Theoretical Computer Science 423 (2012) 30–49

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Monge properties of sequence alignment

Luís M.S. Russo *

Instituto Superior Técnico - Universidade Técnica de Lisboa (IST/UTL), Av. Rovisco Pais, 1049-001 Lisboa, Portugal
INESC-ID / KDBIO R. Alves Redol 9, 1000-029 Lisboa, Portugal

ARTICLE INFO

Article history:
Received 21 January 2011
Received in revised form 22 November 2011
Accepted 25 December 2011
Communicated by R. Giancarlo

Keywords:
Cyclic strings
Edit distance
Longest common subsequence
Computational complexity
String matching
Monge property
Matrix multiplication
Alignment
Incremental string comparison

ABSTRACT

Alignment is an important sequence comparison measure. Algorithms that compute alignments have a wide range of applications, namely in bioinformatic tools. Alignments can be computed as maximum scoring paths in Alignment DAGs. In this paper we study the properties of matrices that contain alignment scores between a string and all the sub-strings of another string. We focus on the fact that these matrices have the Monge property and are sparse in some sense. Related studies were recently presented for HSM and DIST matrices, leading to $O(n \log n)$ procedure for multiplying those matrices, where $O(n)$ bounds the sizes of the strings. Our results strictly generalize previous solutions. We measure the sparseness of the matrices with variable δ and present an algorithm for matrix multiplication in $O((n+\delta)\log^3(n+\delta))$ time, which we improve to $O((n+\delta)\log^2(n+\delta))$, within the same space. We discuss applications of this algorithm, namely fully incremental alignment and alignment update. We study, experimentally, the performance of the methods we propose.

1. Introduction

This paper presents algorithms to multiply Monge [1] matrices, more precisely for the max-plus product of anti-Monge matrices. We use the anti-Monge property, but drop the prefix and refer to them as Monge. These matrices have a long history of algorithmic applications [1]. They play an important role in optimization theory. For example in the traveling salesman problem [2], among other applications. They are also important for several polynomial algorithms, particularly related to dynamic programming. The SMAWK algorithm [3] is a particularly significant example of how the Monge property can be used to speed-up a combinatorial search. The original motivation was the geometric problem of determining all the farthest neighbors of a set of convex points.

Monge matrices have applications to string processing problems, namely computing sequence alignments [4] and determining RNA secondary structure [4, Section 7.6.2] and [5]. Moreover, Monge matrices appear naturally when storing several related alignment values, known as DIST tables [6,7], when using the edit distance, or as Highest-Scoring Matrices (HSMs) [8], when considering Longest Common Subsequences. The fact that these matrices are Monge has a significant impact on string problems such as Cyclic LCS [9–12], Longest Repeated subsequence, Fully-Incremental LCS [8,11,13,14], etc.

Alves et al. [15] proposed an online algorithm to compute an implicit representation of HSMs, in $O(n^2)$ time and $O(n)$ working space, where $O(n)$ bounds the sizes of the strings being processed. Subsequently Tiskin presented an alternative way to represent these matrices in $O(n)$ space [8]. In this paper we generalize this representation, with the notion of core. Given this representation the natural question is: can we multiply HSMs in linear time? This was proposed as an

* Correspondence to: INESC-ID / KDBIO R. Alves Redol 9, 1000-029 Lisboa, Portugal. Tel.: +351 21 31 00272; fax: +351 21 31 45843.
E-mail address: lsr@kdbio.inesc-id.pt.

0304-3975/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2011.12.068

open problem by Landau[1] [16]. Tiskin made significant contributions by proposing $O(n^{1.5})$ and $O(n \log n)$ time algorithms [8,17,18]. However the algorithms mentioned were limited to the Unit-Monge case, meaning that the implicit representation is regular, in the sense that its entries can only be 0 or 1. This restricts the application of previous results to the edit distance, LCS or even weighted LCS problems. Alignments resulting from general score matrices will yield more complex implicit representations, involving real numbers.

We consider the general multiplication problem of core-sparse Monge matrices, i.e., $o(n^2)$ core size, and present a core sensitive algorithm, the multiple maxima trees algorithm (MMT), along with an amortized version, AMMT. Tiskin's algorithm can be partially generalized, although affected by a variable factor ν, this factor is explained in Section 4.2. A comprehensive comparison is given in Section 4. The experimental results show that our prototype obtains the performance bounds of AMMT. We use our general multiplication algorithm to propose the first, as far as we know, non-trivial solution for sequence alignment update and circular alignment computation.

The structure of the paper is the following: Section 2 defines fundamental concepts; Section 3 describes the MMT and AMMT algorithms; Section 4 gives a theoretical and experimental analysis of the several algorithms, including applications; Section 5 concludes the paper.

2. Fundamental concepts

In this section we present some concepts related to Monge matrices, then we introduce the implicit multiplication problem and finish with the HSM class of matrices.

2.1. Basic notions and properties of monge matrices

In this paper $\log n$ is $\log_2 n$. Matrix row and column indices start at 0. For matrix $\mathbf{A}$ consider the expression:

$$\Delta\mathbf{A}(i, i'; j, j') = \mathbf{A}(i', j') - \mathbf{A}(i', j) - \mathbf{A}(i, j') + \mathbf{A}(i, j). \quad \text{對角-反對角} \geq 0 \tag{1}$$

A matrix $\mathbf{A}$, of size $r \times c$, is **Monge**[2] iff $\Delta\mathbf{A}(i, i'; j, j') \geq 0$, for any indices i, i', j and j' such that $0 \leq i \leq i' < r$ and $0 \leq j \leq j' < c$. Fig. 1 shows an example of a Monge matrix, where $\Delta\mathbf{A}(3, 5; 0, 2) = (-3) - (-10) - (-20) + (-3) = 24$. Fig. 2 shows the same matrix, along with the non-zero $\Delta\mathbf{A}(i, i+1; j, j+1)$ values inside boxes, we also refer to these values as the density matrix, which we define shortly. The leftmost argument maximum of a given row is denoted as lax, *i.e.*, the smallest column index were the maximum of a row occurs. In the matrix figures, such as Fig. 1, the leftmost maxima per row are in bold. For example $\operatorname{lax}\mathbf{A}[4, \_] = 1$, the respective maximum is $\mathbf{A}(4, \operatorname{lax}\mathbf{A}[4, \_]) = 6$. A matrix is **monotone** when $i \leq i'$ implies $\operatorname{lax}\mathbf{A}[i, \_] \leq \operatorname{lax}\mathbf{A}[i', \_]$. In other words the lax values increase as the row index increases. The Monge property implies that the matrices are also monotone, this can be observed in Figs. 1 and 6 by noticing that the bold values move to the right when descending by the rows.

Monge matrices occur naturally in several computer science problems. The Monge property, Eq. (1), usually entails a significant efficiency boost in the problems where it occurs. Historically this property dates back to Gaspard Monge (1781), who studied the continuous transportation problem, Monge is in fact known as the "father of Descriptive geometry". Alan Hofffman later used this property to solve Hitchcock's transportation with a greedy algorithm. The Monge property became popular, among computer scientists, due to the seminal paper by Aggarwal et al. [3] on searching in totally monotone matrices, which presented the SMAWK algorithm, hence originating a new focus for research on Monge matrices. The original SMAWK algorithm article presented a linear time solution for the all-farthest neighbor problem, for the vertices of a convex polygon. Nowadays the following problems are known have efficient algorithms that explore the Monge property.

- The Uncapacitated transportation problem consists of determining, for a given Monge cost network $\mathbf{A}(i, j)$, the distribution $x(i, j) \geq 0$ which minimizes the total cost $\sum_{0 \leq i < r} \sum_{1 \leq j \leq c} \mathbf{A}(i, j)x(i, j)$ for transporting goods from the supply vector $a = (a(0), \ldots, a(r-1))$ into the demand vector $b = (b(0), \ldots, b(c-1))$, such that $\sum_{0 \leq j < c} x(i, j) = a(i)$ and $\sum_{1 \leq i \leq r} x(i, j) = b(j)$. An $O(c + r)$ solution follows from a general result of Hoffman [19], whereas for the general case, non Monge, the best solutions require more than $O((c + r)^2)$ time [20]. The Monge property is also relevant for several variations of the transportation problem [1].
- Searching and selecting in Monge matrices, *i.e.*, the determining all the maximum or minimum values, per rows or per columns, can be computed, with the SMAWK [3] algorithm, in $O(c \max\{1, \log(r/c)\})$ time. This is opposed the naive $O(rc)$ algorithm for general matrices.
- Dynamic programming problems, *i.e.*, algorithms that avoid computing repeated recursive calls by storing the values in a table. Early relations to the Monge property were presented by Wilber [21]. This technique is valuable in bioinformatic problems, such a RNA folding and sequence alignment [22]. Larmore and Schieber studied applications of the Monge property to this kind of problems [23], see the exposition by Giancarlo in [4, Chapter 7]. A surprising such result, involving data compression was proposed by Crochemore et al. [24]
- The traveling salesman problem, with a Monge cost matrix, becomes $O(c^2)$, where c is the number of cities [25].

[1] The problem was formulated for Dist tables but it is essentially equivalent.

[2] The usual Monge definition is $\Delta\mathbf{A}(i, i'; j, j') \leq 0$, in which case $-\mathbf{A}$ verifies our condition.

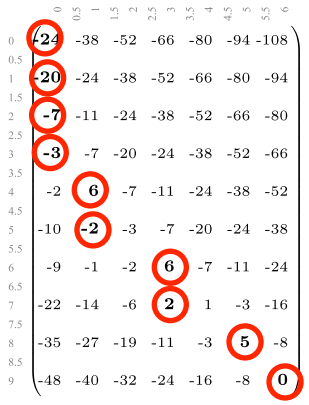


Fig. 1. The **A** Monge matrix. The row and column indices are shown outside. The leftmost maximum, per row, is in bold.

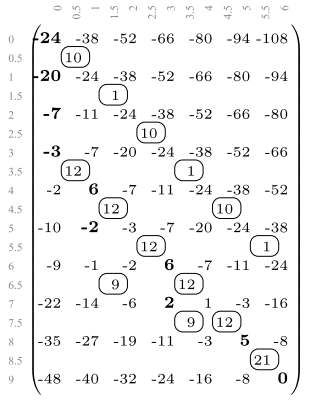


Fig. 2. The **A** Monge matrix and the underlying density matrix. The row and column indices are shown outside. The leftmost maxima, per row, are in bold.

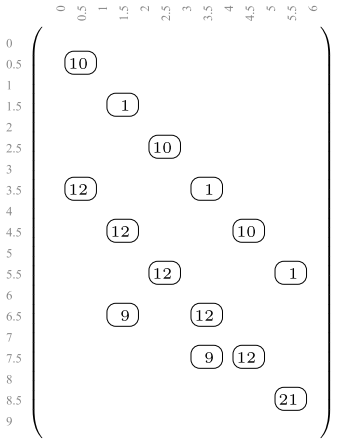


Fig. 3. A density matrix, the 0 entries are omitted. The row and column indices are shown outside.

For a detailed explanation of these and other problems related to Monge matrices, see Burkard et al. [1]. A recent application of Monge matrices to Hierarchial modeling was presented by Imaev and Judd [26].

The notion of core follows from an alternative characterization of Monge matrices. Let D denote a matrix, of size $(r-1) \times (c-1)$, with non-negative values, referred to as a **density matrix**. The rows and columns of these matrices are indexed over half integers, *i.e.*, the first row and the first column are indexed by 0.5 instead of 0. Fig. 3 shows an example of such a matrix. The matrices consist of the values enclosed in boxes, the omitted boxes represent a 0. The respective

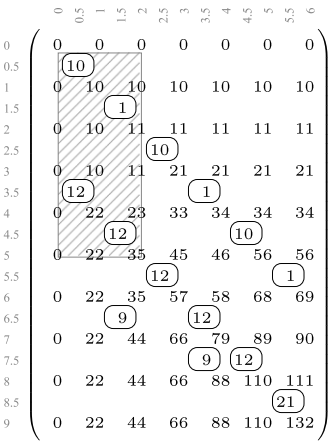


Fig. 4. A density matrix and the respective distribution matrix. The shaded rectangle illustrates the relation between distribution matrices and density matrices, namely $\mathbf{A}(5, 2) = 10 + 1 + 12 + 12 = 35$.

distribution matrix d, of size $r \times c$, is defined, over the integers, as:

$$d(i, j) = \sum_{0<i'<i; 0<j'<j} D(i', j') \quad \text{for all } 0 \le i < r \text{ and } 0 \le j < c. \quad \text{左上方和} \tag{2}$$

A density and a distribution matrix are shown, simultaneously, in Fig. 4.

The next lemma shows an alternative characterization of Monge matrices. The Lemma also establishes a relation between the values and the density matrix, namely $D(i + 0.5, j + 0.5) = \Delta\mathbf{A}(i, i + 1; j, j + 1)$.

Lemma 1 (*Lemma 2.1 in [1]*). *A matrix* $\mathbf{A}$*, of size* $r \times c$*, is Monge iff there is an* $r \times c$ *distribution matrix* d *and two vectors* $u \in \mathbb{R}^r$ *and* $t \in \mathbb{R}^c$ *such that*

$$\mathbf{A}(i, j) = d(i, j) + u(i) + t(j) \quad \text{for all } 0 \le i < r \text{ and } 0 \le j < c. \tag{3}$$

Proof. The proof follows by observing that $\Delta\mathbf{A}$ is additive, *i.e.*, $\Delta\mathbf{A}(i, i'; j, j') + \Delta\mathbf{A}(i', i''; j, j') = \Delta\mathbf{A}(i, i''; j, j')$ and likewise for j. Moreover define $t(j) = \mathbf{A}(0, j)$ and $u(i) = \mathbf{A}(i, 0) - t(0)$. □

2.2. Problem statement and related results

The non-zero entries of D form the core of $\mathbf{A}$, where $\delta(\mathbf{A})$ denotes its size. A matrix is considered sparse if $\delta(\mathbf{A}) = o(rc)$. Notice that $\Delta\mathbf{A}(i, i'; j, j') = \Delta d(i, i'; j, j')$, therefore computing the expression in Eq. (1) consists of summing the core entries inside $[i, i'] \times [j, j']$, for example $\Delta\mathbf{A}(3, 5; 0, 2) = 12 + 12$, as previously shown with direct computation.

This lemma is presented to illustrate the fact that the values of $\mathbf{A}$ can be obtained from a smaller representation. Substituting the definitions of u and t into Eq. (3) we obtain that $\mathbf{A}(i, j) = d(i, j) + \mathbf{A}(0, j) + \mathbf{A}(i, 0) - \mathbf{A}(0, 0)$. Hence we refer to the triple with the density matrix d and the $\mathbf{A}(i, 0)$ and $\mathbf{A}(0, j)$ values as the implicit representation of $\mathbf{A}$. In Fig. 5 the matrices use this representation, and moreover represent the implicit matrix multiplication problem. Let us start by defining the notion of Monge matrix multiplication.

Definition 2. For matrices $\mathbf{A}$, $\mathbf{B}$, of sizes $r \times c$ and $(c = r') \times c'$, the max-plus product matrix, $\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$, is $\mathbf{C}(i, j) = \max_{0 \le k < c}\{\mathbf{A}(i, k) + \mathbf{B}(k, j)\}$.

Fig. 6 shows a sample max-plus matrix product. In this paper we present an algorithm to multiply Monge matrices, represented in implicit form, *i.e.*, given $\mathbf{A}$ and $\mathbf{B}$ in implicit form determine the implicit representation of $\mathbf{C}$, see Fig. 5. Therefore the input requires $O(r + c + c' + \delta(\mathbf{A}) + \delta(\mathbf{B}))$ space and the output requires $O(r + c' + \delta(\mathbf{C}))$ space. We are interested in algorithms that operate within this overall space, and ideally within the same time bound. State of the art solutions partially solve the problem. Tiskin [17] presented an algorithm for Unit-Monge matrices, *i.e.*, when the entries of the density matrix are only 0 or 1. For general Monge matrices it is also possible to iterate the SMAWK [3] algorithm, although the resulting algorithm is slow, and a direct implementation requires more than the implicit space.

In this paper we study the multiplication of implicit Monge matrices, we adapt the iterated SMAWK algorithm, so that it operates in implicit space. Since the resulting algorithm is still slow we propose the Multiple Maxima Trees (MMT) algorithm which is faster than the iterated SMAWK approach, but slower than the algorithm of Tiskin, for Unit-Monge matrices. We present a theoretical and experimental analysis of the performance of MMT in Section 4, where a larger variety of matrices is considered.

We start by studying the iterated SMAWK algorithm. It may seem that this algorithm is not useful for the implicit multiplication problem, since it will require random access to the entries of $\mathbf{A}$ and $\mathbf{B}$. Storing these entries explicitly requires

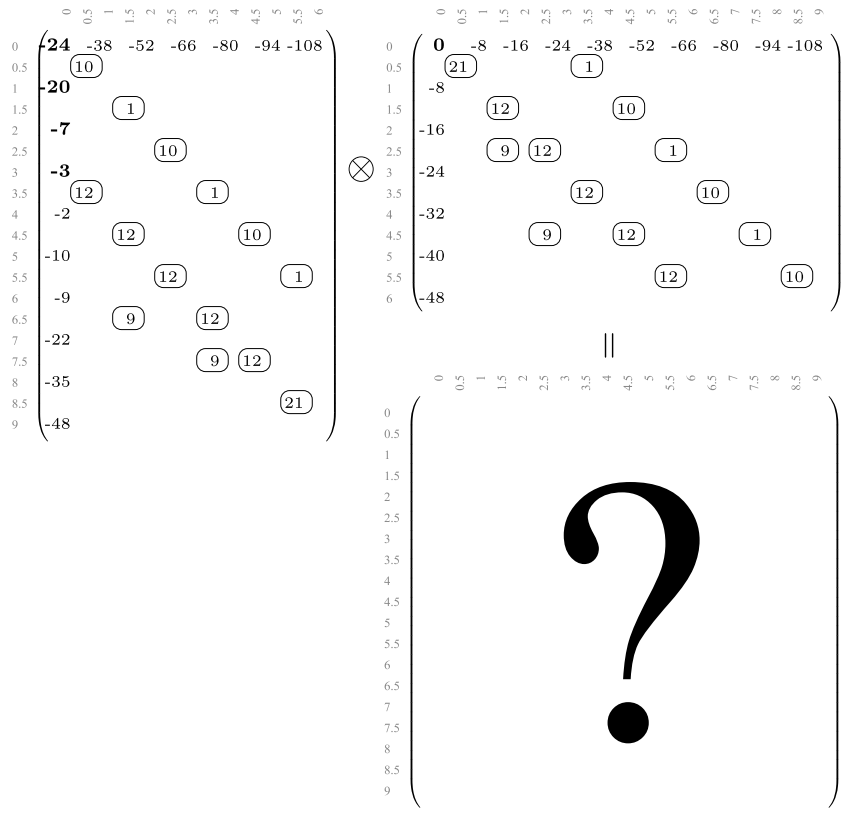


Fig. 5. The **A** and **B** matrices in implicit form. An illustration of the implicit matrix multiplication problem.

$O(rc + r'c')$ space, which is much more space than the one we are considering. For now we will ignore this issue and still study the SMAWK algorithm. In Section 4 we discuss representations of **A** and **B**, that require much less space and still provide random access to the underlying matrices.

To perform the multiplication we organize it into a sequence of *intermediate computation matrices*, $\mathbf{M}_i$, of size $c' \times r'$, such that $\mathbf{M}_i(j,k) = \mathbf{A}(i,k) + \mathbf{B}(k,j)$, for $0 \le i < r,\ 0 \le j < c',\ 0 \le k < r' = c$. Fig. 7 shows $\mathbf{M}_3$. These matrices are used to compute the values of **C**, since $\mathbf{C}(i,j) = \mathbf{M}_i(j, \operatorname{lax}\mathbf{M}_i[j,\_])$, which by definition of lax is the same as $\max_k \mathbf{M}_i(j,k)$. For example row 3 of **C** can be obtained from $\mathbf{M}_3$. Observe that the bold numbers of Fig. 7 correspond to row 3 of **C** in Fig. 6. An important reason to consider the $\mathbf{M}_i$ matrices is that they inherit the Monge property from **B**.

Lemma 3. *Let* **A** *and* **B** *be Monge matrices, of sizes* $r \times c$ *and* $(c = r') \times c'$*, then the* $\mathbf{M}_i$ *matrices are Monge, for any* $0 \le i < c$.

Proof. Use Lemma 1 to conclude that $\mathbf{B}(k,j) = d_{\mathbf{B}}(k,j) + u_{\mathbf{B}}(k) + t_{\mathbf{B}}(j)$, where $d_{\mathbf{B}}$ is a distribution matrix. Recall that $\mathbf{M}_i(j,k) = \mathbf{A}(i,k) + \mathbf{B}(k,j) = d_{\mathbf{B}}(k,j) + (\mathbf{A}(i,k) + u_{\mathbf{B}}(k)) + t_{\mathbf{B}}(j)$. Note that i is not an argument of $\mathbf{M}_i(j,k)$, therefore we can group the $\mathbf{A}(i,k)$ term in the vertical vector. The resulting matrix is Monge because it has the structure described in Lemma 1. □

The previous proof points out that the underlying distribution matrix of the different $\mathbf{M}_i$ matrices is the same, in fact the one of **B**. This regularity partially explains our algorithm.

We will use the Monge property of the $\mathbf{M}_i$ matrices to compute **C** in a non-naive way.

Lemma 4. *Let* **A** *and* **B** *be Monge matrices, of sizes* $r \times c$ *and* $(c = r') \times c'$*, then* $\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$ *can be obtained in* $O(rr' \max\{1, \log(c'/r')\})$ *time.*

Proof. Applying SMAWK [3] to the $\mathbf{M}_i$ matrices returns all the row maxima. For a given $\mathbf{M}_i$, this procedure takes $O(r' \max\{1, \log(c'/r')\})$ time. Iterating this procedure for all possible i's yields the bound in the Lemma and returns all the lax $\mathbf{M}_i[j,\_]$ values. □

This paper does not explain the SMAWK algorithm, an interested reader should consult Aggarwal et al. [3]. A simple, yet less efficient alternative is to recursively find the maximum of the middle row and divide **M** into two smaller sub-problems. For **M** of size $c' \times r'$ this process takes only $O(r' \log c')$ time. Therefore an iterated version of this algorithm would require $O(rr' \log c')$ time. The MMT algorithm uses the regularities of the $\mathbf{M}_i$ matrices and a variation of this algorithm.

The iterated SMAWK algorithm is also inadequate for the implicit Monge matrix multiplication problem, because there we obtain an explicit representation of **C**, instead of its core. In Section 3.2, we explain how to compute the core values, and hence the implicit representation of **C**, within reasonable working space.

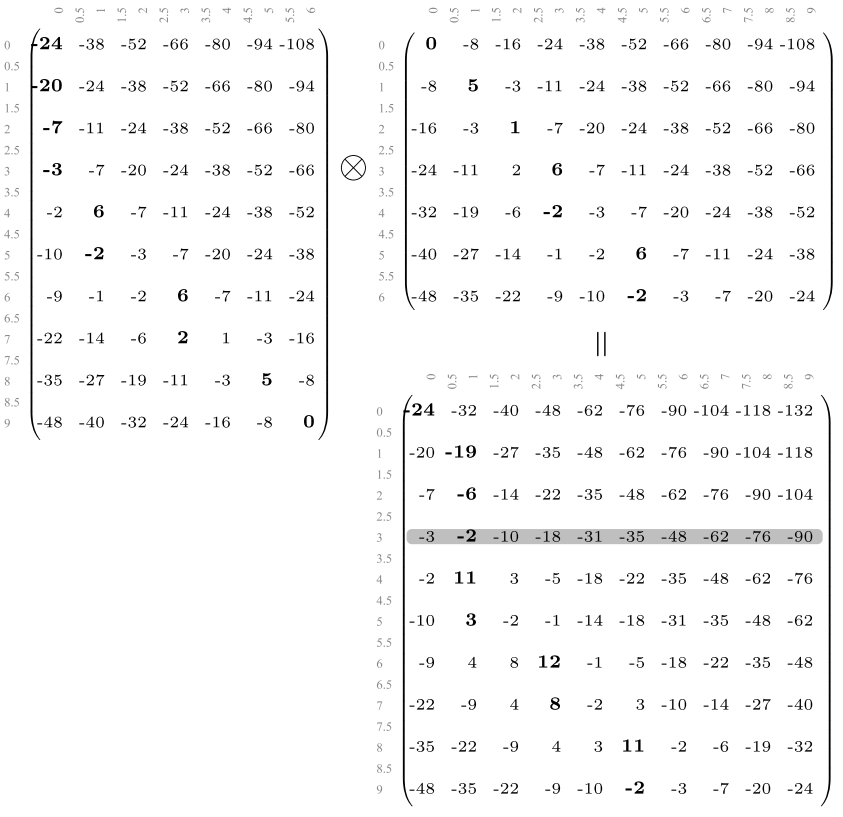


Fig. 6. Illustration of the Monge max-plus product. Three Monge matrices, **A** (top-left), **B** (top-right) and $\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$ (bottom). The row and column indices are shown outside. Row 3 of **C** is highlighted by a grey background.

	0	1	2	3	4	5	6
0	**-3,**	-15,	-36,	-48,	-70,	-92,	-114,
1	-11,	**-2,**	-23,	-35,	-57,	-79,	-101,
2	-19,	**-10,**	-19,	-22,	-44,	-66,	-88,
3	-27,	**-18,**	-27,	-18,	-40,	-53,	-75,
4	-41,	**-31,**	-40,	-31,	-41,	-54,	-76,
5	-55,	-45,	-44,	**-35,**	-45,	-46,	-68,
6	-69,	-59,	-58,	**-48,**	-58,	-59,	-69,
7	-83,	-73,	-72,	**-62,**	-62,	-63,	-73,
8	-97,	-87,	-86,	**-76,**	-76,	-76,	-86,
9	-111,	-101,	-100,	**-90,**	-90,	-90,	-90,

Fig. 7. The figure shows the $\mathbf{M}_3$ matrix.

M3的最大值和C的row3一至

Due to the additive properties of $\Delta\mathbf{A}$ the average density of $\mathbf{A}$ can be computed as:

$$\overline{\Delta\mathbf{A}} = \sum_{0\le i<r;\ 0\le j<c} \Delta\mathbf{A}(i, i+1; j, j+1)/\delta(\mathbf{A}) = \Delta\mathbf{A}(0, r; 0, c)/\delta(\mathbf{A}). \tag{4}$$

This value measures the average density values, counting only the positive entries. Re-writing Eq. (4) we have $\delta(\mathbf{A}) = \Delta\mathbf{A}(0, r; 0, c)/\overline{\Delta\mathbf{A}}$. When considering integer valued matrices we have that $\overline{\Delta\mathbf{A}} \ge 1$, therefore $\delta(\mathbf{A}) \le \Delta\mathbf{A}(0, r; 0, c)$, becomes a bound on the core size. We show experimentally, in Table 2, that $\overline{\Delta\mathbf{A}}$ assumes higher values in practice (when the density matrices are not restricted to 0 or 1 values).

The MMT algorithm is effective, as long as the intervening matrices are sparse, *i.e.*, $\delta(\mathbf{A}) = o(cr)$, $\delta(\mathbf{B}) = o(cr)$ and $\delta(\mathbf{C}) = o(cr)$. In fact $\delta(\mathbf{C})$ depends on $\delta(\mathbf{A})$ and $\delta(\mathbf{B})$. We prove that applying the max-plus product to two Monge matrices yields a Monge matrix, and also relate $\delta(\mathbf{C})$ with characteristics of the matrices $\mathbf{A}$ and $\mathbf{B}$.

Lemma 5. *Let $\mathbf{A}$ and $\mathbf{B}$ be Monge matrices, of sizes $r \times c$ and $(c = r') \times c'$, let also $0 \le i \le i' < r$ and $0 \le j \le j' < c$ be indices, and $k_{i,j} = \operatorname{lax}\mathbf{M}_i[j, \_]$ indices that correspond to maxima, the following properties hold:*

- $\max\{\Delta\mathbf{A}(i, i'; k_{i',j}, k_{i,j'}), \Delta\mathbf{B}(k_{i,j'}, k_{i',j}; j, j')\} \le \Delta\mathbf{C}(i, i'; j, j')$
- $\Delta\mathbf{C}(i, i'; j, j') \le \min\{\Delta\mathbf{A}(i, i'; k_{i,j}, k_{i',j'}), \Delta\mathbf{B}(k_{i,j}, k_{i',j'}; j, j')\}$.
- $\mathbf{C}$ *is Monge.*
- $\delta(\mathbf{C}) \le \min\{\Delta\mathbf{A}(0, r; 0, c), \Delta\mathbf{B}(0, r'; 0, c')\}/\overline{\Delta\mathbf{C}}$
- *For integer valued matrices* $\delta(\mathbf{C}) \le \min\{\Delta\mathbf{A}(0, r; 0, c), \Delta\mathbf{B}(0, r'; 0, c')\}$

Proof. The first inequalities follow from a similar derivation. We present one such derivation and point out the differences for the remaining:

$$\begin{aligned}
&\Delta\mathbf{C}(i, i'; j, j') \\
&\quad = \mathbf{C}(i', j') - \mathbf{C}(i', j) - \mathbf{C}(i, j') + \mathbf{C}(i, j) && (5)\\
&\quad = \mathbf{C}(i', j') - \mathbf{M}_{i'}(j, k_{i',j}) - \mathbf{M}_i(j', k_{i,j'}) + \mathbf{C}(i, j) && (6)\\
&\quad \ge \mathbf{M}_{i'}(j', k_{i,j'}) - \mathbf{M}_{i'}(j, k_{i',j}) - \mathbf{M}_i(j', k_{i,j'}) + \mathbf{M}_i(j, k_{i',j}) && (7)\\
&\quad = \mathbf{A}(i', k_{i,j'}) - \mathbf{A}(i', k_{i',j}) - \mathbf{A}(i, k_{i,j'}) + \mathbf{A}(i, k_{i',j}) && (8)\\
&\quad = \Delta\mathbf{A}(i, i'; k_{i',j}, k_{i,j'}). && (9)
\end{aligned}$$

The equality (5) follows from the definition of $\Delta\mathbf{C}(i, i'; j, j')$. Equality (6) follows from the definition of $k_{i,j'}$ and $k_{i',j}$. Inequality (7) follows by choosing $k_{i,j'}$ in $\mathbf{C}(i', j')$, instead of the index $k_{i',j'}$, which would maximize the value. Likewise we also choose $k_{i',j}$ in $\mathbf{C}(i, j)$. To obtain equality (8) note that $\mathbf{M}_{i'}(j', k_{i,j'}) - \mathbf{M}_i(j', k_{i,j'}) = \mathbf{A}(i', k_{i,j'}) - \mathbf{A}(i, k_{i,j'})$ and that $\mathbf{M}_i(j, k_{i',j}) - \mathbf{M}_{i'}(j, k_{i',j}) = \mathbf{A}(i, k_{i',j}) - \mathbf{A}(i', k_{i',j})$, *i.e.*, in this case we can eliminate the influence of $\mathbf{B}$. To obtain $\Delta\mathbf{C}(i, i'; j, j') \ge \Delta\mathbf{B}(k_{i,j'}, k_{i',j}; j, j')$ in inequality (7) swap the chosen indices, *i.e.*, choose $k_{i',j}$ in $\mathbf{C}(i', j')$ and $k_{i,j'}$ in $\mathbf{C}(i, j)$.

For the second inequality use the indices $k_{i',j'}$ and $k_{i,j}$ in Eq. (6). Now we choose these indices in $\mathbf{C}(i', j)$ and $\mathbf{C}(i, j')$ to obtain a new inequality (7), which becomes a $\le$ inequality.

To prove that $\mathbf{C}$ is Monge assume that $k_{i',j} \le k_{i,j'}$, in which case $0 \le \Delta\mathbf{A}(i, i'; k_{i',j}, k_{i,j'})$, because $\mathbf{A}$ is Monge. Otherwise $k_{i,j'} < k_{i',j}$ and $0 \le \Delta\mathbf{B}(k_{i,j'}, k_{i',j}; j, j')$, because $\mathbf{B}$ is Monge. Hence, using the first property, we can conclude that $0 \le \Delta\mathbf{C}(i, i'; j, j')$ and therefore $\mathbf{C}$ is Monge.

For the fourth property uses the second inequality and Eq. (4) to obtain $\delta(\mathbf{C}) = \Delta\mathbf{C}(0, r; 0, c')/\overline{\Delta\mathbf{C}} \le \min\{\Delta\mathbf{A}(\ldots;\ldots), \Delta\mathbf{B}(\ldots;\ldots)\}/\overline{\Delta\mathbf{C}}$. Moreover $\Delta\mathbf{A}(0, r; k_{0,0}, k_{r,c}) + \Delta\mathbf{A}(0, r; 0, k_{0,0}) + \Delta\mathbf{A}(0, r; k_{r,c}, c) = \Delta\mathbf{A}(0, r; 0, c')$ and all the parcels are positive. Using a similar reasoning for $\mathbf{B}$ we conclude that $\Delta\mathbf{C}(0, r; 0, c') \le \min\{\Delta\mathbf{A}(0, r; 0, c), \Delta\mathbf{B}(0, r'; 0, c')\}$.

The last property follows from the previous one by observing that for integer valued matrices we have that $\overline{\Delta\mathbf{C}} \ge 1$. □

Note that for the Unit-Monge matrices, considered by Tiskin, the last property implies the result that $\delta\mathbf{C} \le \min\{\delta\mathbf{A}, \delta\mathbf{B}\}$. Unit-Monge matrices are such that the non-zero entries of the density matrix are always 1. In this case $\overline{\Delta\mathbf{A}} = \overline{\Delta\mathbf{B}} = \overline{\Delta\mathbf{C}} = 1$, the result follows by using Eq. (4).

2.3. Highest score matrices

Monge matrices occur in several dynamic programming problems, in particular related to sequence alignment. We denote by S, S' and T **strings** of size m, m' and n respectively; by Σ the **alphabet** of size σ; by $S[i]$ the symbol at position i, assuming that positions start at 0; by $S.S'$ **concatenation**; by $S = S[..i-1].S[i..j].S[j+1..]$ respectively a **prefix**, a **substring** and a **suffix**; note that $S[i..j]$, with $j < i$, denotes the empty string; a **subsequence** of S is obtained by deleting zero or more letters; a **Longest Common Subsequence** $\mathrm{LCS}(S, T)$ is a largest subsequence that can be obtained from both strings S and T. Consider the following example $S = ba$, $S' = ab$ and $T = baba$, where $m = m' = 2$ and $n = 4$. In this example $\mathrm{LCS}(S, T) = ba$, $\mathrm{LCS}(S', T) = ab$ and $\mathrm{LCS}(S.S', T) = bab$.

The size of $\mathrm{LCS}(S, T)$ can be computed as a highest-scoring path in a DAG. The DAG is a grid of horizontal and vertical edges with score 0, it contains diagonal edges, with score 1, for every pair of matching characters between the strings. For

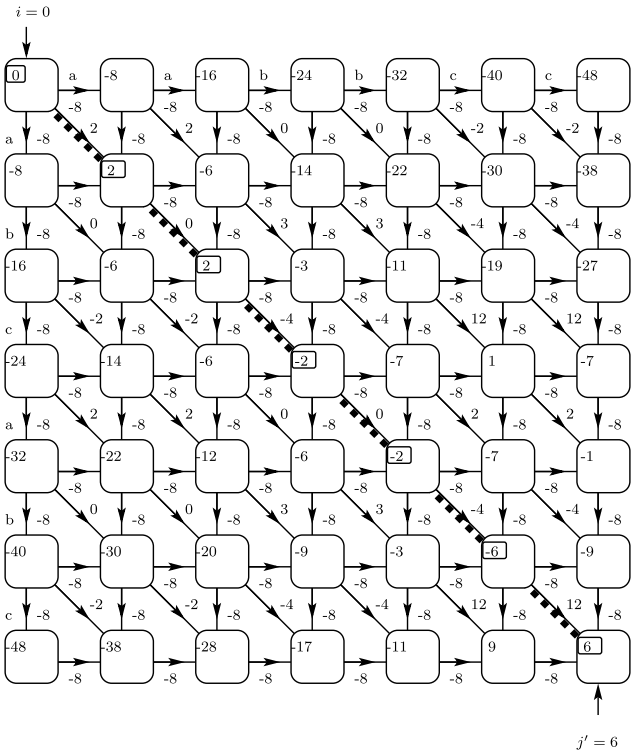


Fig. 8. The general alignment DAG of $S = abcabc$ and $T = aabbcc$, using the PAM matrix. The nodes contain the computation of one highest scoring path, which is also illustrated with a dashed line and boxes.

a general alignment ALIGN(S, T) the scores can have arbitrary real values. Moreover the scores do not have to be unique for a type of edge, *i.e.*, all the edges can have different values. In bioinformatic applications, alignments that are computed with PAM [27] matrices usually have scores that are letter dependent, *i.e.*, the scores associated with an edge usually depend on the letters that are associated with that edge. A general alignment DAG for strings "abcabc" and "aabbcc", that uses the PAM scoring is shown in Fig. 8, for this alignment we are assuming the following correspondences to amino acids: 'a' to Alanine; 'b' to Asparagine or Aspartic acid; 'c' to Cysteine. An even more regular type of alignment, but less regular than LCS is the **Weighted Longest Common Subsequence** (WLCS). In this variant the weights of the edges depend on the type of edge, horizontal, vertical, diagonal match or diagonal mismatch. Fig. 9 shows an WLCS alignment DAG.

Depending on the starting and ending nodes of the path we can determine several LCS values, between S and a substring of T. The highest-score matrix (HSM) $\mathbf{H}_{S,T}$ stores these LCS values, *i.e.*, $\mathbf{H}_{S,T}(i, j) = \text{LCS}(S, T[i..j-1])$. In fact the full HSM matrix $\mathbf{H}_{S,T}(i, j)$ contains the values of the highest scoring paths on an infinite extended alignment DAG LCS$(S[i..], T[..j-1])$ and LCS$(S[..j-1], T[i..])$. Using this simplification we can obtain $\mathbf{H}_{S.S',T}$ as $\mathbf{H}_{S,T} \otimes \mathbf{H}_{S',T}$, the general multiplication is possible, but more complex. A similar matrix was denominated as DIST(S, T) by Apostolico et al. [6], for edit distance problems. In this paper we follow the theory by Tiskin [17,18] but the results are essentially equivalent.

In this context the max-plus product, Definition 2, represents the fact that a highest-score path of $\mathbf{H}_{S.S',T}$ can be decomposed into a path of $\mathbf{H}_{S,T}$ followed by a path of $\mathbf{H}_{S',T}$. HSMs, from non-weighted LCS, are Monge [6], core-sparse and unit, *i.e.*, $\delta(\mathbf{H}_{S,T}) = \min\{m, n\} = o(n^2)$ and the non-empty core entries are always 1. HSMs from WLCS are not necessarily unit, for example the matrix $\mathbf{A}$ of Fig. 2. To use Tiskin's algorithm for these matrices it is necessary to expand them into unit matrices, this entails a performance penalty that we discuss in Section 4. To see why HSMs are Monge choose two pairs of indices $i \leq i'$ and $j \leq j'$. Note that $\mathbf{H}_{S,T}(i, j)$ and $\mathbf{H}_{S,T}(i', j')$ correspond to two, possibly, non-crossing paths. On the other hand $\mathbf{H}_{S,T}(i, j')$ and $\mathbf{H}_{S,T}(i', j)$ correspond to two paths that must necessarily cross each other [7].

We illustrate this property over WLCS derived HSM matrices $\mathbf{H}^W_{S,T}$, *i.e.*, matrices defined as above, but for WLCS. Fig. 10 shows an example of the simultaneous computation of WLCS values. Therefore $\mathbf{H}^W_{S,T}(i, j) = \text{WLCS}(S, T[i..j-1])$. Now consider Fig. 11, the paths in this figure are obtained by following the highest scores, presented in Fig. 10. Choosing $i = 0$, $i' = 2$ and $j = 4$ and $j' = 6$. Note that the $\mathbf{H}^W_{S,T}(i, j') = \text{WLCS}(S, T)$ and $\mathbf{H}^W_{S,T}(i', j) = \text{WLCS}(S, ab)$, by decomposing the first path we obtain that $\mathbf{H}^W_{S,T}(i, j') = \text{WLCS}(ab, aa) + \text{WLCS}(abab, abbb)$. Likewise by decomposing the second path we obtain that $\mathbf{H}^W_{S,T}(i, j') = \text{WLCS}(ab, \epsilon) + \text{WLCS}(abab, ab)$. Notice that $\mathbf{H}^W_{S,T}(i, j) \leq \text{WLCS}(ab, aa) + \text{WLCS}(abab, ab)$ and that $\mathbf{H}^W_{S,T}(i', j') \leq \text{WLCS}(ab, \epsilon) + \text{WLCS}(abab, abbb)$. Therefore we can conclude that $\mathbf{H}_{S,T}(i, j) + \mathbf{H}_{S,T}(i', j') \geq \mathbf{H}_{S,T}(i, j') + \mathbf{H}_{S,T}(i', j)$ by reorganizing the terms in the expression and using the previous inequalities.

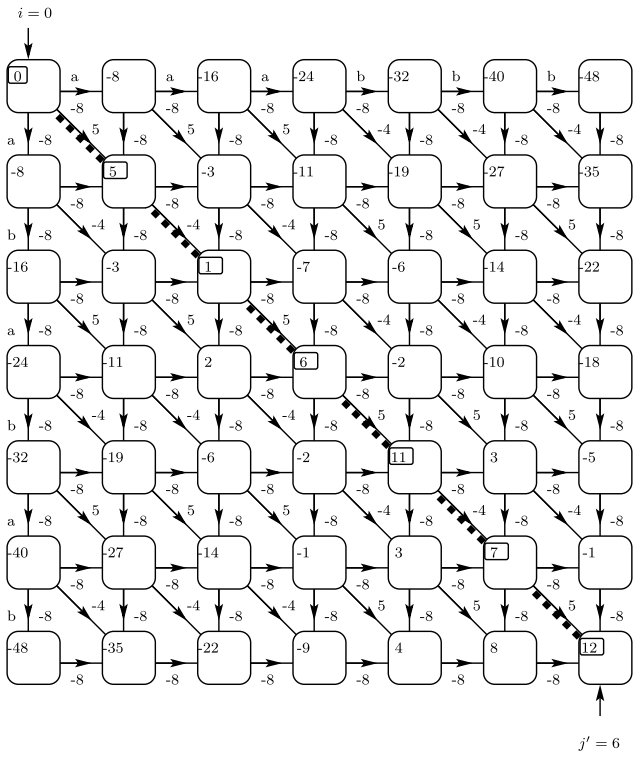


Fig. 9. The WLCS alignment DAG of $S = ababab$ and $T = aaabbb$. The nodes contain the computation of one highest scoring path, which is also illustrated with a dashed line and boxes.

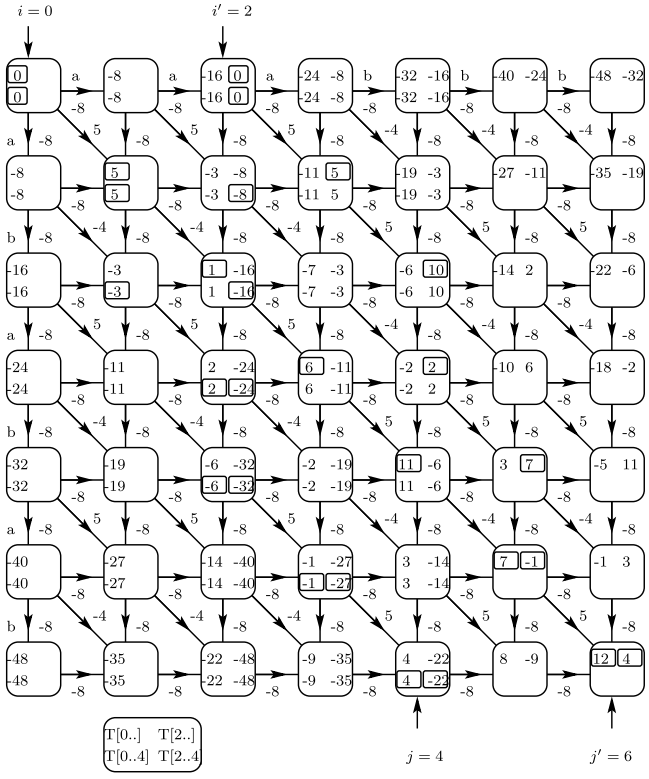


Fig. 10. The WLCS alignment DAG of $S = ababab$ and $T = aaabbb$ (top-left in the boxes). The nodes also contain the computation of the highest scoring paths for the suffix $T[2..] = abbb$ (top-right), *i.e.*, the computation starting with $i = 2$. Moreover the alignments ending with $j = 4$ are also considered, *i.e.*, $T[2..4] = ab$ (bottom-right) and $T[0..4] = aaab$ (bottom-left). A box that illustrates relative positions is given below.

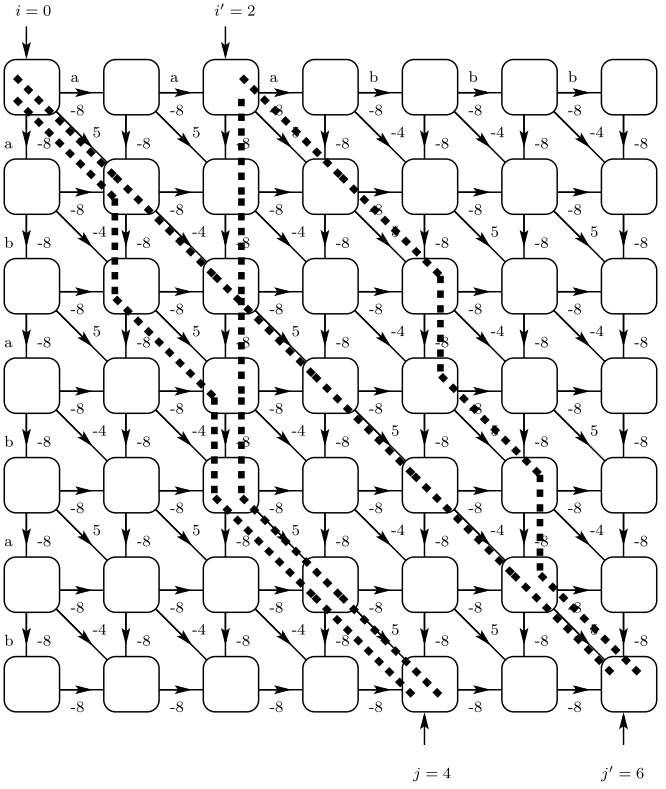


Fig. 11. The WLCS alignment DAG of $S = ababab$ and $T = aaabbb$. We show the four highest scoring paths, for $i = 0, i' = 2, j = 4, j' = 6$. This figure illustrates the discussion on the Monge Property of HSM matrices, on the next page. Note that the crossing paths can be rearranged by the crossing point, resulting in paths from i to j and from i' to j', which must necessarily score less than the respective non-crossing paths.

Unit-Monge matrices can be multiplied efficiently as follows:

Theorem 6 ([17]). *Given Unit-Monge matrices* $\mathbf{A}, \mathbf{B}$*, both of size* $n \times n$*, the core entries of* $\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$ *can be obtained in* $O(n \log n)$ *time and* $O(n)$ *space.*

This result has a significant impact on string problems [28], namely Cyclic LCS, Longest Repeated Subsequence, Fully-Incremental LCS [11,13], etc. The latter problem consists of maintaining a data structure that returns the size of $\mathrm{LCS}(S, T)$ and supports updates to $\mathrm{LCS}(c.S, T)$, $\mathrm{LCS}(S.c, T)$, $\mathrm{LCS}(S, c.T)$ and $\mathrm{LCS}(S, T.c)$, where c is a new character. Notice that the first update is against the usual dynamic programming direction and therefore a naive approach requires $O(mn)$ time. Using Theorem 6 with $\mathbf{A} = \mathbf{H}_{c,T}$ the update to $\mathrm{LCS}(c.S, T)$ runs in $O(n \log n)$ time, which is competitive against state of the art solutions [13,14] of $O(n)$ time.

3. Core sensitive multiplication

This Section describes the Multiple Maxima Trees (MMT) algorithm that solves the implicit Monge matrix multiplication problem. The algorithm receives the cores of $\mathbf{A}$ and $\mathbf{B}$ and outputs the core of $\mathbf{C}$. We describe the algorithm in two phases. First we propose a structure that *represents* the individual values of $\mathbf{C}$, *i.e.*, that we can use to access an individual value of $\mathbf{C}$; Second we explain how to use those values to determine the core of $\mathbf{C}$.

3.1. Representing $\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$

We divide matrix $\mathbf{A}$, in half, into $\mathbf{A}^\ell$ and $\mathbf{A}^r$. The left sub-matrix $\mathbf{A}^\ell$ contains columns 0 to $\lceil c/2 \rceil - 1$. The right sub-matrix $\mathbf{A}^r$ contains columns $\lceil c/2 \rceil$ to $c - 1$. For example in Fig. 12 $\mathbf{A}^\ell$ contains columns 0, 1, 2, 3 and $\mathbf{A}^r$ contains columns 4, 5, 6. Notice that there will be an index for which the leftmost maximum changes from $\mathbf{A}^\ell$ to $\mathbf{A}^r$.

Definition 7. The **transition point** $\mathrm{TR}(\mathbf{A})$, of a monotone matrix $\mathbf{A}$, is a half integer such that $\mathrm{lax}\,\mathbf{A}[i, \_]$ is $\mathrm{lax}\,\mathbf{A}^\ell[i, \_]$ for $i < \mathrm{TR}(\mathbf{A})$ and $\lceil c/2 \rceil + \mathrm{lax}\,\mathbf{A}^r[i, \_]$ for $i > \mathrm{TR}(\mathbf{A})$.

整體和左邊max一樣

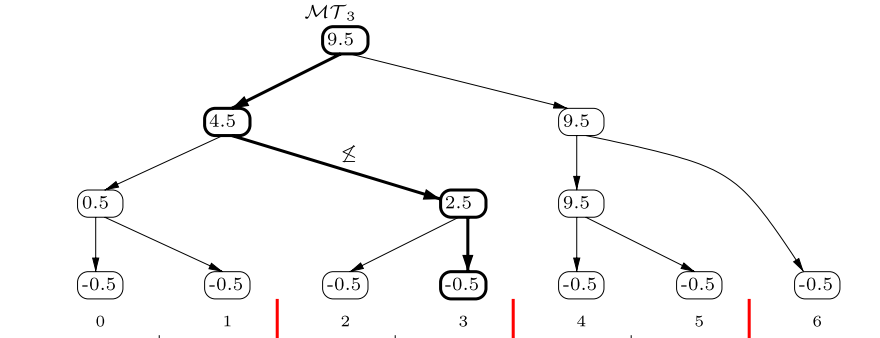
Fig. 12 indicates that $\mathrm{TR}(\mathbf{A}) = 7.5$ with an arrow. Fig. 13 shows that $\mathrm{TR}(\mathbf{M}_3) = 9.5$. The figure also shows $\mathrm{TR}(\mathbf{M}_3^\ell) = 4.5$ and that $\mathrm{TR}(\mathbf{M}_3^r) = 9.5$. We use the convention that $\mathrm{TR}(\mathbf{A}) = -0.5$ if $\mathrm{lax}\,\mathbf{A}[i, \_] = \lceil c/2 \rceil + \mathrm{lax}\,\mathbf{A}^r[i, \_]$, for all i, and that $\mathrm{TR}(\mathbf{A}) = r - 0.5$ if $\mathrm{lax}\,\mathbf{A}[i, \_] = \mathrm{lax}\,\mathbf{A}^\ell[i, \_]$, for all i. For example $\mathrm{TR}(\mathbf{M}_3) = 10 - 0.5$.

	0	0.5	1	1.5	2	2.5	3	3.5	4	4.5	5	5.5	6
0	**-24**	-38	-52	-66	-80	-94	-108
0.5													
1	**-20**	-24	-38	-52	-66	-80	-94
1.5													
2	**-7**	-11	-24	-38	-52	-66	-80
2.5													
3	**-3**	-7	-20	-24	-38	-52	-66
3.5													
4	-2	**6**	-7	-11	-24	-38	-52
4.5													
5	-10	**-2**	-3	-7	-20	-24	-38
5.5													
6	-9	-1	-2	**6**	-7	-11	-24
6.5													
7	-22	-14	-6	**2**	1	-3	-16
7.5													
8	-35	-27	-19	-11	-3	**5**	-8
8.5													
9	-48	-40	-32	-24	-16	-8	**0**

$\mathbf{A}^{\ell}$ $\mathbf{A}^{r}$

TR()為區分max的邊界

Fig. 12. The **A** Monge matrix. The row and column indices are shown outside. The matrix is divided into $\mathbf{A}^{\ell}$ and $\mathbf{A}^{r}$ by a vertical line and $\text{TR}(\mathbf{A}) = 7.5$ is pointed by an arrow.



	0	1	2	3	4	5	6
0	**-3,**	-15,	-36,	-48,	-70,	-92,	-114,
1	-11,	**-2,**	-23,	-35,	-57,	-79,	-101,
2	-19,	**-10,**	-19,	-22,	-44,	-66,	-88,
3	-27,	**-18,**	-27,	-18,	-40,	-53,	-75,
4	-41,	**-31,**	-40,	-31,	-41,	-54,	-76,
5	-55,	-45,	-44,	**-35,**	-45,	-46,	-68,
6	-69,	-59,	-58,	**-48,**	-58,	-59,	-69,
7	-83,	-73,	-72,	**-62,**	-62,	-63,	-73,
8	-97,	-87,	-86,	**-76,**	-76,	-76,	-86,
9	-111,	-101,	-100,	**-90,**	-90,	-90,	-90,

Tr(M3) = 9.5
Tr(Ml3) = 4.5
Tr(Mr3) = 9.5

Fig. 13. The figure shows the $\mathbf{M}_3$ matrix (Bottom) and the respective maxima tree (Top). The edges of the tree are labeled ℓ and r depending on whether the corresponding sub-tree is left or right. The arrows, on the left and on the right, indicate that $\text{TR}(\mathbf{M}_3) = 9.5 = \text{TR}(\mathbf{M}_3^r)$ and that $\text{TR}(\mathbf{M}_3^{\ell}) = 4.5$. The $\not\leq$ symbol illustrates that assuming $\text{TR}(\mathbf{A}^{\ell}) \leq \text{TR}(\mathbf{A}) \leq \text{TR}(\mathbf{A}^r)$, is **incorrect**. The highlighted path in the tree shows the computation of lax $\mathbf{M}_3[5, \_]$.

Definition 8. The **maxima tree** $\mathcal{MT}_{\mathbf{A}}$, of a Monge matrix $\mathbf{A}$, is a balanced binary tree. If $\mathbf{A}$ is empty $\mathcal{MT}_{\mathbf{A}}$ is also empty, otherwise the sub-trees that start at the children of the Root are $\mathcal{MT}_{\mathbf{A}^\ell}$ and $\mathcal{MT}_{\mathbf{A}^r}$. The Root stores $\textsc{Tr}(\mathbf{A})$, the remaining nodes store the corresponding transition points.

Fig. 13 shows the maxima tree[3] of $\mathbf{M}_3$. Maxima trees are not binary search trees, therefore it is **incorrect** to assume that $\textsc{Tr}(\mathbf{A}^\ell) \leq \textsc{Tr}(\mathbf{A}) \leq \textsc{Tr}(\mathbf{A}^r)$, Fig. 13 shows a counter example, indicated by $\not\leq$. The maxima tree can be used to compute lax values.

Lemma 9. *Let $\mathbf{A}$ be a Monge matrix, of size $r \times c$. Its maxima tree $\mathcal{MT}_{\mathbf{A}}$ can be stored in $O(c)$ space and supports $\operatorname{lax}\mathbf{A}[i, \_]$ in $O(\log c)$ time.*

Proof. We compute $\operatorname{lax}\mathbf{A}[i, \_]$ recursively as $\operatorname{lax}\mathbf{A}^\ell[i, \_]$ if $i < \textsc{Tr}(\mathbf{A})$ and as $\lceil c/2 \rceil + \operatorname{lax}\mathbf{A}^r[i, \_]$ otherwise, *i.e.*, move to the left child, v_ℓ, or to the right child, v_r. The execution time is bounded by the height of the tree, that is $O(\log c)$. Since we store only one value per node, the space occupied by the tree depends on the number of nodes, which is $O(c)$. □

Suppose we want to compute $\operatorname{lax}\mathbf{M}_3[5, \_]$. We start at the Root and since $5 < 9.5$ we move to the left, now since $5 > 4.5$ we move to the right, since $5 > 2.5$ we move to the right and reach the leaf of column $3 = \operatorname{lax}\mathbf{M}_3[5, \_]$. Notice that we can also compute the lax values of a sub-matrix corresponding to an internal node of $\mathcal{MT}_{\mathbf{A}}$.

Lemma 10. *A maxima tree can be built in $O(c \log r)$ time.*

Proof. The tree can be built bottom-up. The $\textsc{Tr}(\mathbf{A})$ of the Root can be computed with a binary search in $O(\log r)$ steps. If $\mathbf{A}^\ell(i, \operatorname{lax}\mathbf{A}^\ell[i, \_]) \geq \mathbf{A}^r(i, \operatorname{lax}\mathbf{A}^r[i, \_])$ then[4] $\textsc{Tr}(\mathbf{A}) > i$, otherwise $\textsc{Tr}(\mathbf{A}) < i$. For the remaining internal nodes the process is similar. This yields an overall $O(c(\log c)\log r)$ time. An amortized analysis shows that we are not paying $O(\log c)$ to obtain the lax values, as in Lemma 9. Most nodes are close to the leaves. Half of the nodes pay 1 operation for lax. One quarter of the nodes pay 2 operations, and so on. The overall time is $O(c \log r)$. □

Building all the $\mathcal{MT}_{\mathbf{M}_i}$ trees takes $O(rr' \log c')$ time and $O(rr')$ space. The resulting structure provides $O(\log r')$ access time to $\mathbf{C}(i, j)$, recall that $\mathbf{C}(i, j) = \mathbf{M}_i(j, \operatorname{lax}\mathbf{M}_i[j, \_])$. This result is illustrative but it is still inefficient. The $\mathbf{M}_i$ matrices have regularities that can considerably increase the efficiency of this procedure.

Lemma 11. *Given Monge matrices $\mathbf{A}$ and $\mathbf{B}$, when $\Delta\mathbf{A}(i, i'; k, k') = 0$ then $\mathbf{M}_i(j, k) \leq \mathbf{M}_i(j, k')$ iff $\mathbf{M}_{i'}(j, k) \leq \mathbf{M}_{i'}(j, k')$.*

Proof. The following diagram proves the lemma:

$$\mathbf{M}_i(j,k) - \mathbf{M}_i(j,k') = \mathbf{A}(i,k) - \mathbf{A}(i,k') + \mathbf{B}(k,j) - \mathbf{B}(k',j) \tag{10}$$

$$\| \qquad \Delta A(i,i';k,k')=0 \tag{11}$$

$$\mathbf{M}_{i'}(j,k) - \mathbf{M}_{i'}(j,k') = \mathbf{A}(i',k) - \mathbf{A}(i',k') + \mathbf{B}(k,j) - \mathbf{B}(k',j). \tag{12}$$

Eqs. (10) and (12) follow from the definition of $\mathbf{M}_i$. Eq. (11) follows from the hypothesis. □

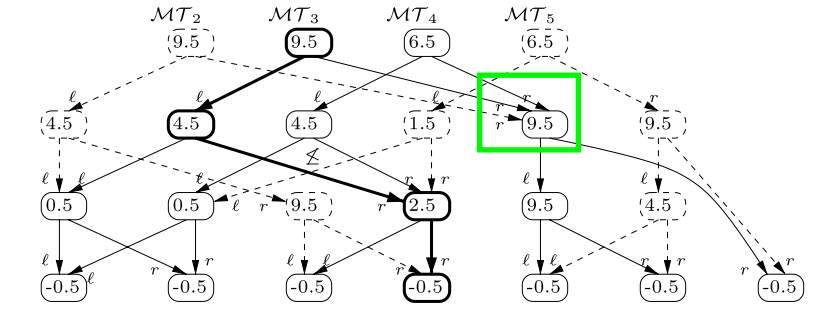
Notice that $\Delta\mathbf{A}(i, i'; k, k') = 0$ implies that $\Delta\mathbf{A}(i_1, i_2; k_1, k_2) = 0$ for any $i \leq i_1 \leq i_2 \leq i'$ and $k \leq k_1 \leq k_2 \leq k'$. Therefore the lemma shows that there is redundant information among the $\mathcal{MT}_{\mathbf{M}_i}$ trees. Namely if $\Delta\mathbf{A}(i, i'; k, k') = 0$ and v, v' are nodes of $\mathcal{MT}_{\mathbf{M}_i}$ and $\mathcal{MT}_{\mathbf{M}_{i'}}$, respectively, whose corresponding rows are the $[k, k']$ interval then the sub-trees of v and v' are identical. Fig. 14 shows an example of this observation, matrices $\mathbf{M}_2$, $\mathbf{M}_3$, $\mathbf{M}_4$ share the right sub-tree, since $\Delta\mathbf{A}(2, 4; 4, 6) = 0$. Moreover $\mathbf{M}_5$ does not share the right sub-tree with $\mathbf{M}_4$, since $\Delta\mathbf{A}(4, 5; 4, 6) = 10 \neq 0$.

Lemma 12. *Given Monge matrices $\mathbf{A}$ and $\mathbf{B}$, of sizes $r \times c$ and $(c = r') \times c'$, there is a representation of $\mathbf{C}$ with $O(\log r')$ access time, that needs $O(r' + \delta(\mathbf{A}) \log r')$ space and $O(r + (r' + \delta(\mathbf{A})(\log r')^2) \log c')$ time to be built.*

Proof. Use Lemma 10 to build $\mathcal{MT}_1$, in $O(r' \log c')$ time. In general use Lemma 11 to build $\mathcal{MT}_{\mathbf{M}_{i+1}}$ from $\mathcal{MT}_{\mathbf{M}_i}$. If $\Delta\mathbf{A}(i, i+1; 0, c-1) = 0$ then $\mathcal{MT}_{\mathbf{M}_i}$ and $\mathcal{MT}_{\mathbf{M}_{i'}}$ are identical and the computation finishes. Otherwise we build a new Root for $\mathcal{MT}_{\mathbf{M}_{i'}}$ and proceed recursively to determine whether the left and right children of $\mathcal{MT}_{\mathbf{M}_{i'}}$ are new or the same as in $\mathcal{MT}_{\mathbf{M}_i}$. Each core value of $\mathbf{A}$ originates at most $\log r'$ new nodes. For each new node we recompute, bottom-up, the respective transition points, in $O((\log r')\log c')$ time each. The $O(r)$ term comes from the i cycle. □

[3] Note that we simplified the notation and use only $\mathcal{MT}_3$ to refer to the maxima tree of $\mathbf{M}_3$.

[4] Note that the previous expression is $\max_k \mathbf{A}^\ell[i, k] \geq \max_k \mathbf{A}^r[i, k]$, the expression illustrates how this value is computed.



	0	1	2	3	4	5	6
0	**-7, -3,** **-2, -10,**	-19, -15, -2, -10,	-40, -36, -23, -19,	-62, -48, -35, -31,	-84, -70, -56, -52,	-106,-92, -78, -64,	-128,-114 -100,-86,
1	-15, -11, -10, -18,	**-6, -2,** **11, 3,**	-27, -23, -10, -6,	-49, -35, -22, -18,	-71, -57, -43, -39,	-93, -79, -65, -51,	-115,-101 -87, -73,
2	-23, -19, -18, -26,	**-14,-10,** **3,** -5,	-23, -19, -6, **-2,**	-36, -22, -9, -5,	-58, -44, -30, -26,	-80, -66, -52, -38,	-102,-88, -74, -60,
3	-31, -27, -26, -34,	**-22,-18,** **-5,** -13,	-31, -27, -14, -10,	-32, -18, -5, **-1,**	-54, -40, -26, -22,	-67, -53, -39, -25,	-89, -75, -61, -47,
4	-45, -41, -40, -48,	**-35,-31,** **-18,**-26,	-44, -40, -27, -23,	-45, -31, -18, **-14,**	-55, -41, -27, -23,	-68, -54, -40, -26,	-90, -76, -62, -48,
5	-59, -55, -54, -62,	-49, -45, -32, -40,	**-48,**-44, -31, -27,	-49, **-35,** **-22,-18,**	-59, -45, -31, -27,	-60, -46, -32, -18,	-82, -68, -54, -40,
6	-73, -69, -68, -76,	-63, -59, -46, -54,	**-62,**-58, -45, -41,	-62, **-48,** **-35,-31,**	-72, -58, -44, -40,	-73, -59, -45, -31,	-83, -69, -55, -41,
7	-87, -83, -82, -90,	-77, -73, -60, -68,	**-76,**-72, -59, -55,	-76, **-62,** -49, -45,	-76, -62, **-48,**-44,	-77, -63, -49, **-35,**	-87, -73, -59, -45,
8	-101,-97, -96, -104,	-91, -87, -74, -82,	**-90,**-86, -73, -69,	-90, **-76,** -63, -59,	-90, -76, **-62,**-58,	-90, -76, -62, **-48,**	-100,-86, -72, -58,
9	-115,-111, -110,-118,	-105,-101, -88, -96,	**-104,**100, -87, -83,	-104,**-90,** -77, -73,	-104,-90, **-76,**-72,	-104,-90, -76, **-62,**	-104,-90, -76, -62,

ΔA(i,i';k,k')=0
->ΔA(2,4;4,6)=0
->M2/3/4共享r
subtree (col4~6)

Fig. 14. Matrices $\mathbf{M}_2, \mathbf{M}_3, \mathbf{M}_4, \mathbf{M}_5$ and the respective maxima trees. Each cell shows $\mathbf{M}_2$ (top-left), $\mathbf{M}_3$ (top-right), $\mathbf{M}_4$ (bottom-left) and $\mathbf{M}_5$ (bottom-right). The bold values highlight the leftmost maximum, per row. The maxima of $\mathbf{M}_3$, over gray, correspond to row 3 of $\mathbf{C}$ in Fig. 6. Nodes and branches exclusive to $\mathcal{MT}_2$ or $\mathcal{MT}_5$ are dashed. Nodes contain the transition point of the sub-matrix. Similar sub-trees are not repeated. The edges are labeled ℓ and r depending on whether the corresponding sub-tree is left or right. The computation of $\operatorname{lax}\mathbf{M}_3[5, \_] = 3$ is indicated by thicker lines.

3.2. Obtaining the core

This section explains how to obtain the core of $\mathbf{C}$ from its representation.

Lemma 13. *Given a Monge matrix* $\mathbf{C}$*, of size* $r \times c'$*, its core entries can be determined by inspecting* $O(r + \delta(\mathbf{C}) \log c')$ *entries of* $\mathbf{C}$*.*

Proof. For every i we compute $\Delta\mathbf{C}(i, i+1; 0, c'-1)$. If the result is 0 we conclude there are no core entries in $[i, i+1] \times [0, c'-1]$ and we stop the search. Otherwise we recursively consider $\Delta\mathbf{C}(i, i+1; 0, \lceil c'/2 \rceil)$ and $\Delta\mathbf{C}(i, i+1; \lceil c'/2 \rceil, c'-1)$. This procedure needs $O(\log c')$ time for each core entry of $\mathbf{C}$ and overall $O(r)$ time to consider all the rows of $\mathbf{C}$. □

Fig. 15 shows an illustration of the procedure described in the lemma, between rows 3 and 4 of $\mathbf{C}$. We can now combine Lemmas 12 and 13 to obtain our main result. Note that in the working space we do not consider the space of the implicit representation of $\mathbf{A}$ and $\mathbf{B}$. This space is discussed in Section 4, along with the kind of information we can obtain in this space, and the time to do so, but is omitted here for simplicity.

Theorem 14. *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be Monge matrices, of sizes* $r \times c$ *and* $(c = r') \times c'$*, the core of* $\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$ *can be computed in* $O(r \log r' + (r' + (\delta(\mathbf{C}) + \delta(\mathbf{A}) \log r') \log r') \log c')$ *time and* $O(\delta(\mathbf{C}) + r')$ *working space.*

Proof. Lemmas 12 and 13 yield an $O(\delta(\mathbf{C}) + r' + \delta(\mathbf{A}) \log r')$ working space solution. It is not necessary to store all the maxima trees $\mathcal{MT}_{\mathbf{M}_i}$. An iteration of the procedure in Lemma 13 inspects only $\mathbf{M}_i$ and $\mathbf{M}_{i+1}$. Hence it is enough to store only two trees in each iteration, which requires at most $O(r')$ space, see Lemma 9 and proof of Lemma 12. □

Using the equation $\mathbf{C} = (\mathbf{B}^T \otimes \mathbf{A}^T)^T$ the previous result gives an algorithm that runs in $O(c' \log c + (c + (\delta(\mathbf{C}) + \delta(\mathbf{B}) \log c) \log c) \log r)$ time and $O(\delta(\mathbf{C}) + c)$ space.

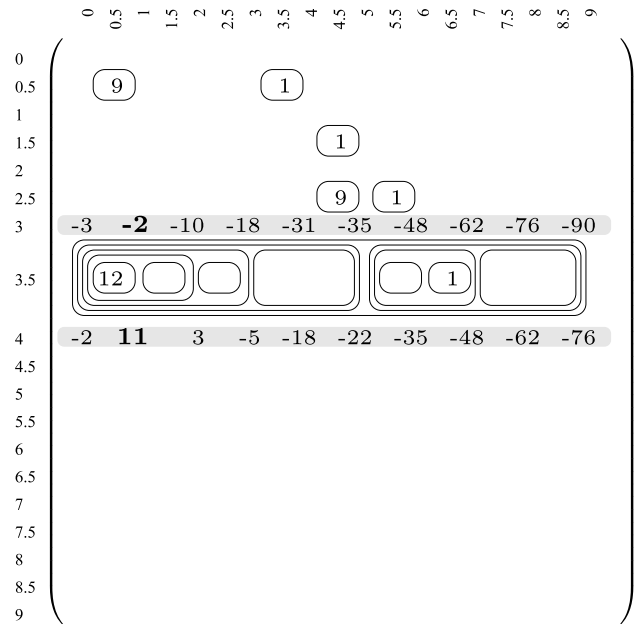


Fig. 15. An illustration of the procedure in Lemma 13. The non-zero $\Delta\mathbf{C}(i, i+1; j, j+1)$ values are inside boxes. Rows 3 and 4 of $\mathbf{C}$ are over a darker background. The boxes between rows 3 and 4 of $\mathbf{C}$ illustrate the procedure in Lemma 13.

3.3. Amortized analysis

This section presents a more efficient version of Theorem 14. The algorithm is essentially the same but we use a slightly different scan and amortized analysis on the core size of the intervening matrices. We start by observing that transition points are monotonic non-increasing.

Lemma 15. *For arbitrary Monge matrices* $\mathbf{A}$, $\mathbf{B}$, *the transition points of the intermediate computation matrices,* $\textsc{Tr}(\mathbf{M}_i)$, *form a monotonic non-increasing sequence,* i.e., $\textsc{Tr}(\mathbf{M}_i) \geq \textsc{Tr}(\mathbf{M}_{i+1})$.

Proof. This property follows from the simpler observation that $\operatorname{lax}\mathbf{M}_i[j, \_] \leq \operatorname{lax}\mathbf{M}_{i+1}[j, \_]$. For example, notice that in Fig. 14 the bold values of $\mathbf{M}_{i+1}$ are to the right of the bold values of $\mathbf{M}_i$.

Suppose by contradiction that for some i and j we have that $k' = \operatorname{lax}\mathbf{M}_i[j, \_] > \operatorname{lax}\mathbf{M}_{i+1}[j, \_] = k$. Consider the following derivation:

$$\underbrace{(\mathbf{M}_i(j,k) - \mathbf{M}_i(j,k'))}_{<0} + \underbrace{((-\mathbf{M}_{i+1}(j,k) + \mathbf{M}_{i+1}(j,k'))}_{\leq 0} < 0 \tag{13}$$

$$\|$$

$$(\mathbf{A}(i,k) + \mathbf{B}(k,j) - \mathbf{A}(i,k') - \mathbf{B}(k',j)) + (-\mathbf{A}(i+1,k) - \mathbf{B}(k,j) + \mathbf{A}(i+1,k') + \mathbf{B}(k',j)) < 0 \tag{14}$$

$$\|$$

$$(\mathbf{A}(i,k) - \mathbf{A}(i,k')) + (-\mathbf{A}(i+1,k) + \mathbf{A}(i+1,k')) < 0 \tag{15}$$

$$\|$$

$$\Delta\mathbf{A}(i, i+1; k, k') < 0. \tag{16}$$

The inequality under the first term of Eq. (13) comes from the hypotheses that $k' = \operatorname{lax}\mathbf{M}_i[j, \_]$ and $k' > k$. A similar reasoning explains the inequality under the second term. These two inequalities justify Eq. (13). The remaining relations are obtained by simplifying this expression. Eq. (14) is obtained from the definition of $\mathbf{M}_i$. Eq. (15) is simplified from Eq. (14). Eq. (15) uses the definition of $\Delta\mathbf{A}(i, i+1; k, k')$. The last equation contradicts the Monge property of $\mathbf{A}$. □

This result will have an impact on Lemma 10. Up to this point the transition points were computed with binary searches. Computing transition points with sequential searches seems, deceptively, less efficient but it will turn out to be good for amortized analysis. Scanning all the rows of $\mathbf{A}$ or $\mathbf{M}_i$ would still be inefficient. We can scan only the rows that have non-zero density values, the next lemma explains why.

Lemma 16. *For an arbitrary Monge matrix* $\mathbf{A}$, *of size* $r \times c$, *we have that* $\Delta\mathbf{A}(\textsc{Tr}(\mathbf{A}) - 0.5, \textsc{Tr}(\mathbf{A}) + 0.5; 0, c) > 0$.

Proof. We prove that if $i \leq i'$ and $\operatorname{lax}\mathbf{A}[i, \_] < \operatorname{lax}\mathbf{A}[i', \_]$ then $\Delta\mathbf{A}(i, i'; 0, c) > 0$. This property implies the result in the lemma because $\textsc{Tr}(\mathbf{A}) - 0.5 \leq \textsc{Tr}(\mathbf{A}) + 0.5$ and $\operatorname{lax}\mathbf{A}[\textsc{Tr}(\mathbf{A}) - 0.5, \_] < \operatorname{lax}\mathbf{A}[\textsc{Tr}(\mathbf{A}) + 0.5, \_]$, otherwise there would be no transition.

Suppose by contradiction that $\Delta\mathbf{A}(i, i'; 0, c) = 0$ and that $j = \operatorname{lax}\mathbf{A}[i, \_] < \operatorname{lax}\mathbf{M}_i[i', \_] = j'$. From the hypotheses we conclude that:

$$\Delta\mathbf{A}(i, i'; j, j') = 0 \tag{17}$$

$$\| \tag{18}$$

$$(\mathbf{A}(i, j) - \mathbf{A}(i, j')) + \underbrace{(-\mathbf{A}(i', j) + \mathbf{A}(i', j'))}_{>0} = 0.$$

Eq. (17) follows from the hypotheses that $\Delta\mathbf{A}(i, i'; 0, c) = 0$. Eq. (18) is derived from Eq. (17) by definition of $\Delta\mathbf{A}(i, i'; j, j')$. The inequality of the second term follows from the fact that $j' = \operatorname{lax}\mathbf{M}_i[i', \_]$ and $j < j'$. From this last relation we can conclude that $\mathbf{A}(i, j) < \mathbf{A}(i, j')$, which is a contradiction because $j = \operatorname{lax}\mathbf{A}[i, \_]$ and therefore $\mathbf{A}(i, j)$ must be a maximum. □

This lemma implies that we can compute a transition point, of $\mathbf{M}_i$, with a sequential search over the core entries of $\mathbf{M}_i$. Combining this search with Lemma 15 means that to compute $\textsc{Tr}(\mathbf{M}_{i'})$ after computing $\textsc{Tr}(\mathbf{M}_i)$ we can simply continue the search. The next lemma uses this approach to obtain an improvement of Lemma 12.

Lemma 17. *Given Monge matrices* $\mathbf{A}$ *and* $\mathbf{B}$*, of sizes* $r \times c$ *and* $(c = r') \times c'$*, there is a representation of* $\mathbf{C}$ *with* $O(\log r')$ *access time, that needs* $O(r' + \delta(\mathbf{A}) \log r')$ *space and* $O(r + r' \log c' + (\delta(\mathbf{A}) + \delta(\mathbf{B}))(\log r')^2)$ *time to be built.*

Proof. We use the same procedure as Lemma 12, but scan over the core entries of $\textsc{Tr}(\mathbf{M}_i)$ sequentially, in linear time, instead of binary searches to determine transition points. Recall that the core of the $\textsc{Tr}(\mathbf{M}_i)$ matrices is the same as the core of $\mathbf{B}$, Lemma 3. Using the Lemmas 15 and 16 we amortize the binary search factor $O(\log c')$ to $O(1)$, by consulting the $O(\delta(\mathbf{B}))$ entries of $\mathbf{B}$.

Let us number the nodes of the maxima tree $\mathcal{MT}_i$ from 1 to r', the $\textsc{Root}$ is 1 and the last leaf is r', denoted as $v_i^{(j)}$. Each of these nodes stores one transition point value. Due to Lemma 16 we are not interested in the value of the transition point itself, instead we are interested in the number of positive density values, in the sub-matrix associated with $v_i^{(j)}$, between 0 and the transition point. We denote these values as $l_i^{(j)}$. Hence $l_i^{(j)} - l_{i+1}^{(j)}$ counts the number of steps that the sequential search executes from iteration j to $j+1$ for a given node. Recall that these values are decreasing, a consequence of Lemma 15. Therefore the overall number of search steps is:

$$\sum_{i=1}^{r} \sum_{j=1}^{r'} l_i^{(j)} - l_{i+1}^{(j)} = \sum_{j=1}^{r'} l_1^{(j)} - l_r^{(j)} \leq 2\delta(\mathbf{B}) \log r'$$

The first equality follows by swapping the $\sum$ operators and using the telescopic property of the sum. The second inequality results from adding the number of positive density values inside all the sub-matrices of $\mathbf{B}$. The bound holds by considering the $l_1^{(j)} - l_r^{(j)}$ values at the smallest sub-matrices, with 2 or 3 columns, and multiplying these values by $\log r'$. These values add-up to at most $\delta(\mathbf{B})$. Each search step uses the lax operation, which accounts for the second $O(\log r')$ factor. Also the $\delta(\mathbf{A})$ still appears in the complexity, for the reasons presented in Lemma 12, which are not accounted for in the previous expression. □

Since Lemma 12 was critical for the performance of Theorem 14, the improved Lemma 17 yields the following result:

Theorem 18. *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be Monge matrices, of sizes* $r \times c$ *and* $(c = r') \times c'$*, the core of* $\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$ *can be computed in* $O(r \log r' + r' \log c' + (\delta(\mathbf{C}) \log c' + (\delta(\mathbf{A}) + \delta(\mathbf{B})) \log r') \log r')$ *time and* $O(\delta(\mathbf{C}) + r')$ *working space.*

Likewise using the equation $\mathbf{C} = (\mathbf{B}^T \otimes \mathbf{A}^T)^T$ the previous result gives an algorithm that runs in $O(c' \log c + c \log r + (\delta(\mathbf{C}) \log r + (\delta(\mathbf{B}) + \delta(\mathbf{A})) \log c) \log c)$ time and $O(\delta(\mathbf{C}) + c)$ space.

4. Analysis

4.1. Theoretical analysis

This section presents a theoretical and empirical analysis of several multiplication algorithms. Until now we have omitted the time to access an entry of $\mathbf{A}$ or $\mathbf{B}$. It is common to assume that this value is $O(1)$, for example, in a geometrical context the entries of $\mathbf{A}$ may represent the Euclidean distance between two points in the plane. Therefore this value can be computed, with a fixed number of operations, instead of having to consult adjacent values in the matrix. In the context of sequence alignment accessing a random element of $\mathbf{A}$ requires a dedicated data structure. Since the algorithms we presented work in $o(rc)$ space, it is not possible to store the values of $\mathbf{A}$ in memory, which would guarantee $O(1)$ access time. Tiskin [18] proposed a dominance counting structure [29] to represent Unit-Monge matrices.[5] In this case computing the sum in Eq. (2) consists of counting the number of points inside a rectangle. This structure supported access to a random element of $\mathbf{A}$ in

[5] More precisely the matrices were assumed to be permutations, but general Unit-Monge matrices can be reduced to this case.

Table 1
Comparison between max-plus multiplication algorithms, accounting for access time to **A** and **B**. The $\langle s(n), t(n)\rangle$ notation means $s(n)$ space and $t(n)$ time requirements.

$\delta = \Theta(n^\epsilon)$	MMT, Theorem 14	AMMT, Theorem 18
$\epsilon = 2$	$\langle O(n^2), O(n^2 \log n)\rangle$	$\langle O(n^2), O(n^2 \log n)\rangle$
$2 < \epsilon \leq 1$	$\langle O(n^\epsilon \log n), O(n^\epsilon \log^3 n)\rangle$	$\langle O(n^\epsilon \log n), O(n^\epsilon \log^2 n)\rangle$
Unit-Monge, $\epsilon = 1$	$\langle O(n), O(n \log^3 n)\rangle$	$\langle O(n), O(n \log^2 n)\rangle$
$1 < \epsilon \leq 0$	$\langle O(n), O(n \log^2 n)\rangle$	$\langle O(n), O(n \log^2 n)\rangle$
$\delta = \Theta(n^\epsilon)$	SMAWK , Lemma 4	Theorem 6
$\epsilon = 2$	$\langle O(n^2), O(n^2)\rangle$	–
$2 < \epsilon \leq 1$	$\langle O(n^\epsilon \log n), O(n^2 \log n)\rangle$	–
Unit-Monge, $\epsilon = 1$	$\langle O(n), O(n^2 \log n)\rangle$	$\langle O(n), O(n \log n)\rangle$
$1 < \epsilon \leq 0$	$\langle O(n), O(n^2 \log n)\rangle$	–

Table 2
Core statistics for the analyzed examples, including average sparsity $(\log \delta(\mathbf{A}))/\log n$, average core density $\overline{\Delta\mathbf{A}}$ and the ν values.

	LCS	WLCS	PAM
$(\log \delta(\mathbf{A}))/\log n$	1.0	1.25	1.30
$\overline{\Delta\mathbf{A}}$	1.0	5.38	15.70
ν	2	14	26

$O((\log \delta(\mathbf{A}))/\log\log \delta(\mathbf{A}))$ time and $O(\delta(\mathbf{A}))$ space. For general Monge matrices use the structure by Willard [30], which requires $O(\log \delta(\mathbf{A}))$ time and $O(\delta(\mathbf{A}) \log \delta(\mathbf{A}))$ space. Note also that besides this type of access, in AMMT, Lemma 17, we need to access the density values of sub-matrices of **B**, this implies accessing the density values inside a vertical strip. This process is also supported by the structure of Willard, or with more recent representations based on the wavelet tree [31,32]. Contrary to the algorithms we propose, the algorithm of Tiskin (Theorem 6) makes regular accesses to the underlying matrix, *i.e.*, it does not require random accesses, instead the values are consulted sequentially and therefore can be computed incrementally in $O(1)$ per access, without a dedicated data structure. These structures are important for the algorithms we present in Theorems 14 and 18. We refer to these algorithms as Multiple Maxima Trees (MMT) and Amortized MMT (AMMT). The MMT algorithms always computes a lax value before accessing an entry, *i.e.*, the accesses of the algorithm are always of the form $\mathbf{C}(i, j) = \mathbf{M}_i(j, \operatorname{lax} \mathbf{M}_i[j, \_])$. Except in the amortized analysis of Lemma 10 the lax operation takes $\Omega(\log r')$ time, see Lemma 9. This property is also confirmed experimentally in this section.

To simplify the analysis let us assume that $\Theta(r) = \Theta(c) = \Theta(r') = \Theta(c') = \Theta(n)$ and that $\Theta(\delta(\mathbf{A})) = \Theta(\delta(\mathbf{B})) = \Theta(\delta(\mathbf{C})) = \Theta(\delta)$. The algorithm of Theorem 6 needs to access entries at each step. Hence one could expect that the access time became a factor of the overall time. However this would be a pessimistic comparison [33]. The reason for this is that the algorithm makes non-random access to the underlying Monge matrices. The accesses are sequential, in the sense that after accessing $\mathbf{A}(i, j)$ the next access value can be computed in $O(1)$ time by adding or subtracting a value from the core of **A**.

Table 1 shows a comparison of the different algorithms according to core size, $\delta = \Theta(n^\epsilon)$. The analysis considers the best space and time requirements that includes the access time to **A** and **B**.

First notice that the algorithms of Theorems 14 and 18 are indeed different, the result in Theorem 18 is not a tighter analysis of the algorithm in Theorem 14. An example is the multiplication of the following matrices:

$$\begin{pmatrix} 0 & -1 & -2 & -3 \\ 0 & 0 & -1 & -2 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & -1 & -2 & -3 \\ 0 & 0 & -1 & -2 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -1 & -2 & -3 \\ 0 & 0 & -1 & -2 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

The density matrix, of these matrices, is the usual identity matrix, with '1's along the main diagonal. It is straightforward to generalize these matrices to size $n \times n$, in which case $\delta = \Theta(n)$ and MMT runs in $O(n \log^3 n)$ time and AMMT runs in $O(n \log^2 n)$ time.

For $\epsilon = 2$ the access time is $O(1)$, in this case Lemmas 12 and 13 have amortized performance, hence the $o(n^2 \log^3 n)$ time of MMT and AMMT. Lemma 12 becomes an iterated Lemma 10. Lemma 13 loses an $O(\log n)$ factor. For $\epsilon < 1$ the access time is $O(\log n)$, the dominating time is that of building $\mathcal{MT}_1$, Lemma 10. Because of the amortized analysis we need to count the access time, hence the $\Omega(n \log n)$ time of MMT and AMMT.

Table 1 shows that the MMT algorithms are not always the most efficient, in particular SMAWK is faster by an $O(\log n)$ factor for $\delta = \Theta(n^2)$. This situation changes for $\epsilon < 2$, not only because SMAWK is not sensitive to the core size, but also

because access times are no longer $O(1)$. For the remaining core-sizes the MMTs are faster than SMAWK . For the particular case of Unit-Monge matrices, with $\delta = \Theta(n)$, the algorithm of Theorem 6 is faster than MMT by a $O(\log^2 n)$ factor and faster then AMMT by an $O(\log n)$ factor. This hints that AMMT can possibly be improved. In practice this factor is not always dominant, especially when the core entries have large values.

An important class of Monge matrices are obtained from WLCS. The algorithm of Theorem 6 can be adapted to this case by multiplying its performance by a scaling factor ν, *i.e.*, it becomes $O(n\nu)$ space and $O(n\nu \log n)$ time. To compare the two algorithms note that $\delta = O(n\nu)$, because all the entries of the matrix are integers. This bound is not always tight. Hence if δ is close to this bound the algorithm of Theorem 6 is faster than MMT by an $O(\log^2 n)$ factor and faster then AMMT by an $O(\log n)$ factor. On the other hand if $\delta = \Omega(n)$, but close to $O(n)$, then the algorithm in Theorem 6 is slower than MMT by an $O(\nu/\log^2 n)$ factor and slower than AMMT by an $O(\nu/\log n)$ factor. Section 4.3 shows experimental results.

4.2. Applications

Using MMT we obtain a non-trivial solution for the Fully-Incremental Alignment problem, which consists of updating a generic alignment score from $\textsc{Align}(S, T)$ to $\textsc{Align}(c.S, T)$ or $\textsc{Align}(S, c.T)$ or $\textsc{Align}(S.c, T)$ or $\textsc{Align}(S, T.c)$, where Align is the respective score and c is a new character. For $\textsc{Align}(S, T.c)$ we can simply compute $\mathbf{H}_{S,T.c}$ by multiplying $\mathbf{H}_{S,T}$ and $\mathbf{H}_{S,c}$, *i.e.*, $\mathbf{H}_{S,T.c} = \mathbf{H}_{S,T} \otimes \mathbf{H}_{S,c}$, the remaining problems can be solved in a similar way.

The resulting procedure takes $O((n+\delta)\log^3 n)$ time and $O(n+\delta \log n)$ space, where δ is the core size of the intervening matrices. Using AMMT this result becomes $O((n+\delta)\log^2 n)$ time and $O(n+\delta \log n)$ space. Notice that apart from the solution presented here the only alternative is a naive solution, that requires a complete re-computation and therefore takes $O(n^2)$ time. The algorithm in Theorem 6 can not be used in the general case, not even if allowing for a multiplicative factor ν.

We can use Eq. (4) to obtain a bound for δ, stated in terms of alignment scores. Let us first consider $\mathbf{A}$ when trying to compute $\textsc{Align}(S, T.c)$. For a given alignment matrix we define ν, such that $|h - d| < \nu$ for any vertical or horizontal score h and any diagonal score d, *i.e.*, is $\textsc{Align}(c, \epsilon)$ or $\textsc{Align}(\epsilon, c)$ for some letter c, and d is $\textsc{Align}(c, c')$, for any letters c and c'. Moreover the previous bound for ν should also hold when $d = 0$. It is enough to choose the smallest ν that satisfies the previous condition. This was the selection criterion for ν in the previous section. The values $-\nu$ are used in the alignment DAGs as the weight of moving backwards on vertical or horizontal edges. In this case $\Delta\mathbf{A}(0, r; 0, c) \leq 2n\nu - w$, where $w = \textsc{Align}(S, T)$. Therefore adding all the core sizes and using Lemma 5 we can conclude that $\delta = O(n(\nu/\overline{\Delta}))$, where $\overline{\Delta}$ is the smallest average in the matrices being multiplied. Typical average density values are shown in Table 2.

Since the solution to the previous problem consists of multiplying matrices it can solve a more general problem, that is both incremental and decremental.

Definition 19. Given strings S, S', T and T' the alignment update problem consists of maintaining a data structure that represents the alignment score between S and $T.T'$, *i.e.*, $\textsc{Align}(S, T.T')$, such that given a character c this value can be updated to $\textsc{Align}(S, T.c.T')$ and vice-versa.

Note that this problem is restricted to update only one of the strings, not both at the same time. For this problem consider a balanced binary tree, AVL [34]. Every leaf of the AVL tree contains the matrix $\mathbf{H}_{S,a}$ for every letter a in $T.T'$. Moreover the internal nodes contain HSMs related to the substrings of $T.T'$. If $R.R'$ is a substring of $T.T'$ and $\mathbf{H}_{S,R}$ is the matrix of the left child and $\mathbf{H}_{S,R'}$ is the matrix of the right child then $\mathbf{H}_{S,R.R'}$ is the matrix of the node. Note that adjacent leaves in the tree contain consecutive letters of $T.T'$. To insert a letter c we simply insert a new leaf in the tree and update the matrices in the path to the Root, using Monge matrix multiplication. When re-balancing occurs the matrices at the nodes can also be updated with matrix multiplication. Likewise deletions are handled in a similar way.

An update to the tree requires at most $O(\log n)$ multiplications, where $O(n)$ is the size of S and $T.T'$. Therefore an update can be executed with $O(\log n)$ calls to AMMT, *i.e.*, the overall time requirement of an update is $O((n+\delta)\log^2(n+\delta)\log n)$. Storing the tree requires $O(n+\delta')$, where δ' is the accumulated core size of all the intervening matrices.

We also consider the circular alignment problem. Given strings S and T a circular alignment is an alignment between S and a cyclic rotation of T, the problem consists in determining the best alignment between S and any cyclic rotation of S'. Solutions for circular LCS, that require $O(n^2)$ time were proposed by several authors. Moreover Tiskin [18] proposed a solution that requires only $O((\frac{n \log\log n}{\log n})^2)$ time.

We present the first, non-naive, solution to circular alignment. Again, Tiskin's algorithm cannot be used for general alignments. Using AMMT our result becomes $O(n(n+\delta)\log^2 n)$ time and $O(n+\delta \log n)$ space, where δ is the core size of the intervening matrices. A naive solution would require $O(n^3)$ time. The procedure consists in computing the $\mathbf{H}_{S,T.T}$ matrix and reading the entries that correspond to alignments between S and substrings of $T.T$ with the size of T, *i.e.*, $\mathbf{H}_{S,T.T}(i, i+|T|)$. This matrix must be computed from scratch, *i.e.*, if $T'.c$ is a prefix of $T.T$ then $\mathbf{H}_{S,T'.c}$ is computed by mutiplying submatrices of $\mathbf{H}_{S,T'}$ and $\mathbf{H}_{S,c}$. This accounts for the $O(n)$ factor in these algorithms. For small alphabets this factor can be improved to $O(n/\log_\sigma n)$ by computing $\mathbf{H}_{S,Q_i}$ for every q-sample of T. In this case we avoid recomputation of repeated matrices. The last step multiplies all the $O(n/\log_\sigma n)$ matrices of the form $\mathbf{H}_{S,Q_i}$. A similar speed-up can be obtained by using the compressibility of T, resulting in an $O(n/H_k)$ factor, where H_k is the k-th order entropy of S [35]. In this case instead of the q-samples tree use the Trie from the Lempel–Ziv 78 [36] parsing of S.

4.3. Experimental results

We implemented a prototype of the MMT algorithm and tested it on an Intel Core2 Duo @1.33 GHz, with 1.9 GiB of RAM running Xubuntu 9.10, with Linux Kernel 2.6.31. The code was compiled with `gcc 4.4.1 -O9`. The implementation is actually an improvement of AMMT, where the computation of the transition points is an inverse binary search. This means that we start with a small interval and double it at each step, provided we can guarantee that it does not contain the transition point we are searching for. Whenever the interval contains the desired transition point we do a regular binary search. This process obtains, essentially, the amortized performance of AMMT, without the overhead of having to keep the **B** matrix in a form that supports sequential access to the positive density values, within each sub-matrix. We refer to this prototype as BMMT.

Results from this prototype were originally reported in [33]. Unfortunately, in that reference, the prototype was denominated MMT, but actually it obtains the performance of AMMT. At the time we were unaware that the inverse binary search procedure would amortize the worst case performance. Still the performance of the algorithms in Theorems 14 and 18 is in fact different, in the worst case. To obtain AMMT's performance it is necessary to use the scan described in Lemma 17, or the inverse binary search just described. In the average case it may be that the performance of MMT is close to the performance of AMMT, but changing an implementation of MMT to BMMT requires little extra effort and therefore there is no compelling argument to use MMT instead.

The prototype consists only of the multiplication and not of the representation of **A** and **B**, which are stored in memory and hence have $O(1)$ access time, our results show that this simplification has no impact in the asymptotic performance of the algorithm. To test the performance we multiplied HSM matrices, obtained from the LCS. We also tested HSMs that resulted from generic alignments with the PAM weight matrix [27] and HSMs from Weighted Longest Common Subsequences (WLCS), using the weights in Fig. 9. The algorithm of Theorem 6 can be extended to support WLCSs, but the space and time are affected by a factor ν, in this scenario $\nu = 14$. On the other hand for PAM this technique does not apply, *i.e.*, the algorithm from Theorem 6 cannot be adapted to this problem. If it were possible it would have $\nu = 26$. The underlying strings S, S', T were random Bernoulli proteins, *i.e.*, $\sigma = 23$, that differed in a letter with 10% probability. The sizes were $m = m' = 4i$ and $n = 8i$ for i between 1 and 128.

Besides comparing the algorithms we also analyzed the underlying matrices with respect to their core. We verified experimentally that our test matrices are sparse, by computing the average $(\log \delta(\mathbf{A}))/\log n$. This value is usually close to 1, and very distant from 2, hence indicating that the matrices are sparse. We also measured the average density of the underlying matrix. The results, in Table 1, highlight the difference between $\Delta\mathbf{A}(0, r; 0, c)$ and $\delta(\mathbf{A})$, recall Eq. (2). The ν factor of the reduction presented by Tiskin is also presented, moreover the following relation holds $\overline{\Delta\mathbf{A}} \leq \nu$, because this factor is used to expand the density matrices into "equivalent" unit density matrices.

The results from executing the algorithms are shown in Fig. 16. To make the plots readable we show only 20% of the points. The x axis is indexed by a variable $N = \max\{n, \delta\}$. For precision we repeated each query during 10 s. The plots on the left show the running time, in seconds, of the BMMT algorithm and of an $O(\nu n \log(\nu n))$ time algorithm, denominated as Simulated. The ν factor is calculated from Tiskin's reduction procedure [18]. The simulated prototype is a divide and conquer algorithm that allocates an array of size $O(\nu n)$ and increases every cell from 0 to $\log(\nu n)$. At each step all the cells in the array are increased by 1, the array is then divided in half and each part is processed recursively. The recursion processes both sub-arrays, but in random order.

We compute a minimum squares (MSQ) estimates of c and d in the expression $O(cN \log^d N)$. Since the behavior is asymptotic the first points may distort the values. To diminish this effect we compute several MSQ estimates, successively discarding the first points. We pessimistically choose the largest estimate.

The results show that for LCS and WLCS the δ values are similar to n, but this is not always true in the PAM case. This can be observed by looking at the simulated results, when $\delta = \Theta(\nu n)$ then $N = \Theta(\nu n)$ and the Simulated result show stable $O(N \log N)$ curves. This happens for the LCS and WLCS cases, but not for the PAM dataset. In the PAM dataset δ varies between values that are $o(\nu n)$, up to $O(\nu n)$. When $\delta = o(\nu n)$ the corresponding $O(\nu n \log(\nu n))$ value is plotted with a smaller N value, hence causing a dispersion in the simulated points. The same then happens for the BMMT prototype. This is in fact a pessimist comparison, for a fair comparison we should use $N = n$ for the simulated algorithm. Still BMMT is already faster, for PAM, because of the ν factor, and for the remaining cases the comparison is fair. Using $O(1)$ access time BMMT is slower than the simulated algorithm for LCS and WLCS but, generally, faster for PAM. Note that d is much smaller than 3, for a decent fitting of the model, *i.e.*, $R^2 > 0.95$, the time bound is always lower than $O(N log^2 N)$.

The graphics on the right show the number of operations for different sub-routines. The respective MSQs appear in the same line in the label. We estimate the ratio between node accesses and leaf accesses (Nodes/Leaves), which is always very close to $O(\log N)$. This value is important because it is against this time that the accesses to **A** and **B** add. If this ratio was 1 we would need to add an $O(\log n)$ factor to the final complexity. Since the ratio is close to $O(\log N)$ the accesses add only a constant term. The BinSteps line counts the average number of steps that is necessary in a binary search that computes a transition point. This value is $o(\log N)$ because the BMMT prototype uses inverse binary search.

Transition points are computed lazily, *i.e.*, only if we need to consult them during the algorithm. TotalOps/N measures the total number of operations that the algorithm used, divided by N. The d estimate of this value is usually larger than the d value obtained in the time graphs, on the left. This may be related to cache effects. For LCS and WLCS the bound was at most $O(\log^{2.1} N)$, for PAM the value was higher, but the model fitting was extremely low.

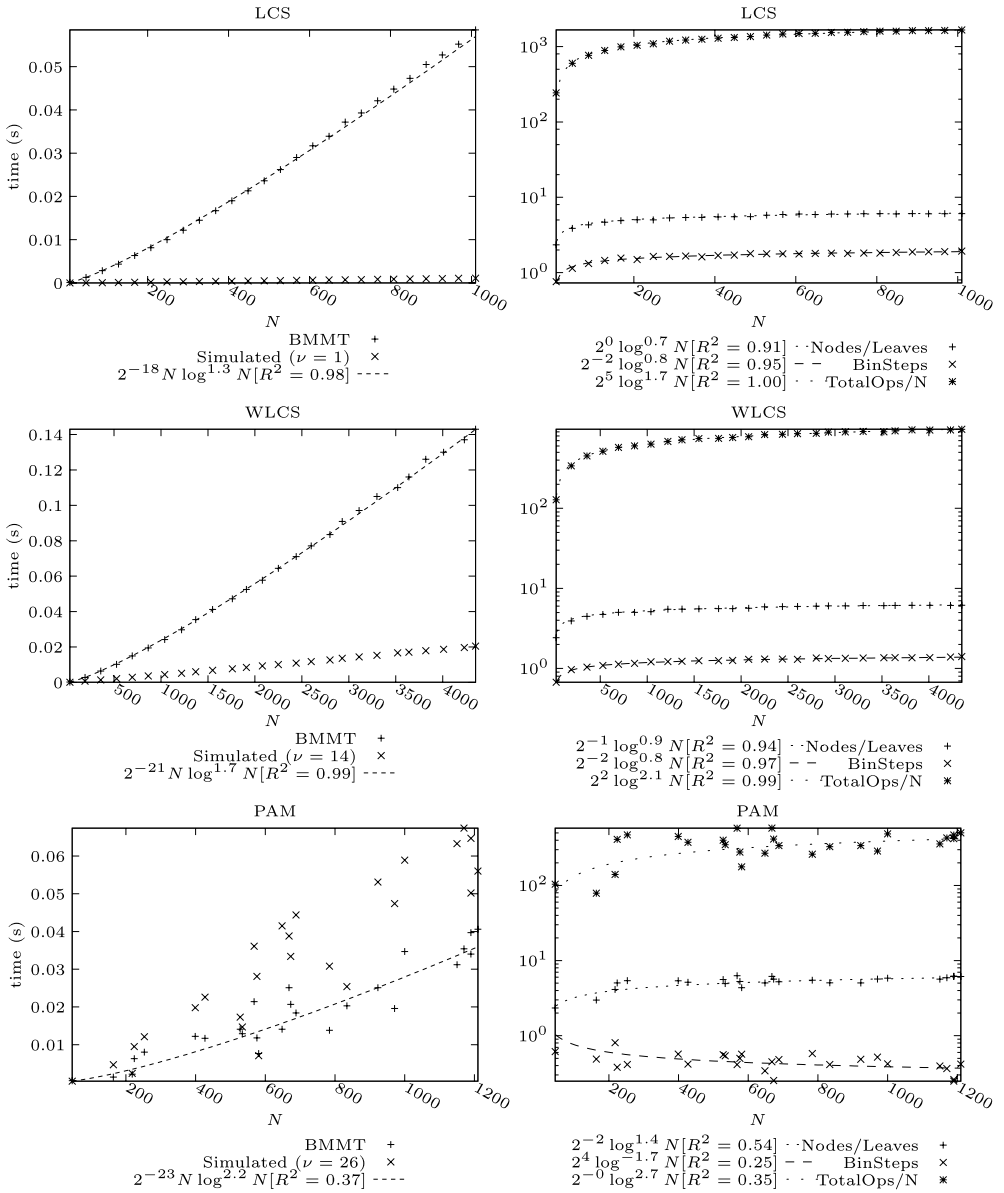


Fig. 16. Experimental testing of the BMMT algorithm. The *x*-axis represents variable $N = \max\{n, \delta\}$. The *y*-axis of the plots on the left indicates the time in seconds. The *y*-axis of the graphs on the right indicates the number of operations.

5. Conclusions

In this paper we studied algorithms for the max-plus product of Monge matrices. We analyzed the existing algorithms considering the core size, Table 1. The analysis showed that the existing algorithms are either sub-optimal or apply only to specific classes of matrices. Alternatively we proposed a core sensitive algorithm, MMT, and an amortized version AMMT. This algorithm is faster than the iterated *SMAWK* algorithm, except when the core is $O(n^2)$. For Unit-Monge matrices the algorithm of Theorem 6 is theoretically faster than AMMT, by an $O(\log n)$ factor. The algorithm of Theorem 6 can also be applied to matrices that result from Weighted LCSs, by using a ν time and space factor.

We also showed experimentally that the MMT algorithms are efficient for general sparse Monge matrices, namely those related to PAM alignments. The MMT and AMMT algorithms are simple and flexible, their main tools are binary trees and binary searches. They are also flexible and provided, as far as we know, the first non-trivial algorithm for the fully-incremental alignment, alignment update and circular alignment problems, string processing problems that are relevant for bio-informatics. We expect MMT and AMMT to have broad applications, since a myriad [17,24] of string processing and optimization problems [1] use Monge matrices.

Acknowledgements

I am grateful to Gad Landau for proposing the DIST multiplication problem, for enlightening discussions on Tiskin's results and references to Monge properties.

I am grateful to Arlindo L. Oliveira for good advise and guidance. I am grateful to the Portuguese Science and Technology Foundation for Funding this research.

Author financed by national funds by FCT/MCTES (PIDDAC, INESC-ID multiannual funding) and projects TAGS PTDC/EIA-EIA/112283/2009, HELIX PTDC/EEA-ELC/113999/2009.

References

[1] R. Burkard, B. Klinz, R. Rudolf, Perspectives of Monge properties in optimization, Discrete Applied Mathematics 70 (2) (1996) 95–161.
[2] J. Park, A special case of the n-vertex traveling-salesman problem that can be solved in $O(n)$ time, Information Processing Letters 40 (5) (1991) 247–254.
[3] A. Aggarwal, M. Klawe, S. Moran, P. Shor, R. Wilber, Geometric applications of a matrix-searching algorithm, Algorithmica 2 (1) (1987) 195–208.
[4] A. Apostolico, Z. Galil, Pattern Matching Algorithms, Oxford University Press, 1997.
[5] M. Waterman, T. Smith, Rapid dynamic programming algorithms for rna secondary structure, Advances in Applied Mathematics 7 (4) (1986) 455–464.
[6] A. Apostolico, M.J. Atallah, L.L. Larmore, S. McFaddin, Efficient parallel algorithms for string editing and related problems, SIAM Journal on Computing 19 (5) (1990) 968–988.
[7] J. Schmidt, All highest scoring paths in weighted grid graphs and their application to finding all approximate repeats in strings, SIAM Journal on Computing 27 (1998) 972.
[8] A. Tiskin, Semi-local longest common subsequences in subquadratic time, Journal of Discrete Algorithms 6 (4) (2008) 570–581.
[9] M. Maes, On a cyclic string-to-string correction problem, Information Processing Letters 35 (2) (1990) 78.
[10] H. Bunke, U. Bühler, Applications of approximate string matching to 2D shape recognition, Pattern Recognition 26 (12) (1993) 1797–1812.
[11] G. Landau, E. Myers, J. Schmidt, Incremental string comparison, SIAM Journal on Computing 27 (1998) 557–582.
[12] A. Marzal, S. Barrachina, Speeding up the computation of the edit distance for cyclic strings, in: Proc. 15th International Conference on Pattern Recognition, vol. 2, 2000, pp. 891 –894.
[13] Y. Ishida, S. Inenaga, A. Shinohara, M. Takeda, Fully Incremental LCS Computation, in: Fundamentals of Computation Theory, in: LNCS, vol. 3623, 2005, pp. 563–574.
[14] G.M. Landau, E.W. Myers, M. Ziv-Ukelson, Two algorithms for lcs consecutive suffix alignment, Journal of Computer and System Sciences 73 (7) (2007) 1095–1117.
[15] C.E.R. Alves, E.N. Cáceres, S.W. Song, An all-substrings common subsequence algorithm, Discrete Applied Mathematics 156 (7) (2008) 1025–1035.
[16] G.M. Landau, Can dist tables be merged in linear time – An open problem, in: Proc. of the Prague Stringology Conference, 2006, p. 1, abstract of invited talk.
[17] A. Tiskin, Fast distance multiplication of Unit-Monge matrices, in: Proc. 21th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, 2010, pp. 1287–1296.
[18] A. Tiskin, Semi-local string comparison: algorithmic techniques and applications, ArXiv e-prints2007arXiv0707.3619T. arXiv:0707.3619.
[19] A. Hoffman, On simple linear programming problems, Selected papers of Alan Hoffman with commentary (2003) 317.
[20] J. Orlin, A faster strongly polynomial minimum cost flow algorithm, in: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, STOC'88, ACM, New York, NY, USA, 1988, pp. 377–387.
[21] R. Wilber, The concave least-weight subsequence problem revisited, Journal of Algorithms 9 (3) (1988) 418–425.
[22] T.F. Smith, M.S. Waterman, Identification of common molecular subsequences, Journal of Molecular Biology 147 (1) (1981) 195–197.
[23] L. Larmore, B. Schieber, On-line dynamic programming with applications to the prediction of RNA secondary structure 1, Journal of Algorithms 12 (3) (1991) 490–515.
[24] M. Crochemore, G. Landau, M. Ziv-Ukelson, A subquadratic sequence alignment algorithm for unrestricted scoring matrices, SIAM journal on computing 32 (6) (2003) 1654–1673.
[25] P.C. Gilmore, E.L. Lawler, D.B. Shmoys, Well-solved special cases of the traveling salesman problem, Tech. Rep. UCB/CSD-84-208, EECS Department, University of California, Berkeley (1984).
[26] A. Imaev, R. Judd, Hierarchial modeling of manufacturing systems using max-plus algebra, in: American Control Conference, 2008, IEEE, 2009, pp. 471–476.
[27] M.O. Dayhoff, R.M. Schwartz, B.C. Orcutt, A model of evolutionary change in proteins, Atlas of Protein Sequence and Structure 5 (suppl 3) (1978) 345–351.
[28] C.S. Iliopoulos, L. Mouchard, M.S. Rahman, A new approach to pattern matching in degenerate DNA/RNA sequences and distributed pattern matching, Mathematics in Computer Science 1 (4) (2008) 557–569.
[29] J. JáJá, C.W. Mortensen, Q. Shi, Space-efficient and fast algorithms for multidimensional dominance reporting and counting, in: Proc. 15th International Symposium on Algorithms and Computation, ISAAC, 2004, pp. 558–568.
[30] D. Willard, New data structures for orthogonal range queries, SIAM Journal on Computing 14 232–253.
[31] G. Navarro, L. Russo, Space-efficient data-analysis queries on grids, in: Proc. 22nd Annual International Symposium on Algorithms and Computation, ISAAC, in: LNCS, vol. 7074, Springer, 2011, pp. 323–332.
[32] N. Brisaboa, M. Luaces, G. Navarro, D. Seco, A new point access method based on wavelet trees, in: Proc. 3rd International Workshop on Semantic and Conceptual Issues in GIS, SeCoGIS, in: LNCS, vol. 5833, Springer, 2009, pp. 297–306.
[33] L.M.S. Russo, Multiplication algorithms for Monge matrices, in: Proc. 17th International Symposium on String Processing and Information Retrieval, SPIRE, in: LNCS, vol. 6393, 2010, pp. 94–105.
[34] G. Adelson-Velskii, E. Landis, An Algorithm for the Organization of Information. Soviet Math, in: Doklady, vol. 3, 1962, pp. 1259–1263.
[35] G. Manzini, An analysis of the Burrows-Wheeler transform, Journal of the ACM 48 (3) (2001) 407–430.
[36] J. Ziv, A. Lempel, Compression of individual sequences via variable-rate coding, IEEE Transactions on Information Theory 24 (5) (1978) 530–536.