Multiplication Algorithms for Monge Matrices

Luís M. S. Russo[1,2]

[1] CITI / Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Quinta da Torre, 2829-516 Caparica, Portugal
lsr@di.fct.unl.pt

[2] INESC-ID, Knowledge Discovery and Bioinformatics Group, R. Alves Redol, 9, 1000-029 Lisbon, Portugal

Abstract. In this paper we study algorithms for the max-plus product of Monge matrices. These algorithms use the underlying regularities of the matrices to be faster than the general multiplication algorithm, hence saving time. A non-naive solution is to iterate the SMAWK algorithm. For specific classes there are more efficient algorithms. We present a new multiplication algorithm (MMT), that is efficient for general Monge matrices and also for specific classes. The theoretical and empirical analysis shows that MMT operates in near optimal space and time. Hence we give further insight into an open problem proposed by Landau. The resulting algorithms are relevant for bio-informatics, namely because Monge matrices occur in string alignment problems.

1 Introduction and Related Work

In this paper we study algorithms to multiply Monge matrices, more precisely the max-plus product of anti-Monge matrices, although we still refer to them as Monge. These matrices have a long history of algorithmic applications [1]. In Particular they occur in string processing problems, as DIST tables [2, 3] or as Highest-Scoring Matrices (HSMs) [4]. Their applications to string processing problems, include: Cyclic LCS, Longest Repeated subsequence, Fully-Incremental LCS [5, 6, 4], etc.

Alves et al. [7] proposed an online algorithm to compute an implicit representation of HSMs in $O(nm)$ time and $O(m+n)$ working space, where m and n are the sizes of the strings being processed. Subsequently Tiskin observed that the core of these matrices has $O(n)$ size [4]. The core provides an alternative way to represent these matrices.

Given this representation the natural question is: can we multiply HSMs in linear time? proposed as an open problem by Landau[3] [8]. Tiskin made significant contributions by proposing $O(n^{1.5})$ and $O(n\log n)$ time algorithms [4, 9, 10]. The latter algorithm becomes $O(n\log^2 n/\log\log n)$ when considering the $O(\log n/\log\log n)$ access time, to the underlying representation structure [11].

This paper generalizes Landau's problem to core-sparse Monge matrices, i.e. $o(n^2)$ core size, and presents a core sensitive algorithm, MMT. For some of these problems Tiskin's algorithm can be applied, although affected by a variable factor ν. In the conditions of Landau's problem MMT runs in $O(n\log^3 n)$ time, including access costs. A

[3] The problem was formulated for DIST tables but it is essentially equivalent.

comprehensive comparison is given in section 4. The experimental results show, Section 4.1, that MMT runs in around $O(n\log^2 n)$ time, i.e., faster than the theoretical bound. Moreover MMT is a general multiplication algorithm that can be applied to general Monge matrices, namely related to alignment problems. Hence we obtain the first non-trivial solution for Fully-Incremental Alignment.

The structure of the paper is the following: Section 2 defines basic concepts; Section 3 describes the MMT algorithm; Section 4 gives a theoretical analysis of the several algorithms and experimental results of MMT; Section 5 concludes the paper.

2 Basic Concepts

This section presents basic concepts related to Monge matrices. In this paper $\log n$ is $\log_2 n$. Matrix row and column indexes start at 0. For matrix $\mathbf{A}$ consider the expression:

$$\Delta\mathbf{A}(i,i';j,j') = \mathbf{A}(i',j') - \mathbf{A}(i',j) - \mathbf{A}(i,j') + \mathbf{A}(i,j) \tag{1}$$

A matrix $\mathbf{A}$, of size $r \times c$, is **Monge**[4] iff $\Delta\mathbf{A}(i,i';j,j') \geq 0$, for any indexes i, i', j and j' such that $0 \leq i \leq i' < r$ and $0 \leq j \leq j' < c$. Figures 1 and 3 show examples of Monge matrices. Fig. 1 shows the non-zero $\Delta\mathbf{A}(i,i+1;j,j+1)$ values inside rectangles, and likewise for matrices $\mathbf{B}$ and $\mathbf{C}$. For example $\Delta\mathbf{A}(3,5;0,2) = (-3)-(-10)-(-20)+(-3) = 24$. 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 Fig. 1 the leftmost maximums 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 Fig. 1 by noticing that the bold values move to the right when descending by the rows.

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 **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. 1 also contains examples of such matrices. The matrices consist of the values enclosed in rectangles, the omitted rectangles represent a 0. The respective **distribution** matrix d, of size $r \times c$, is defined, over the integers, as:

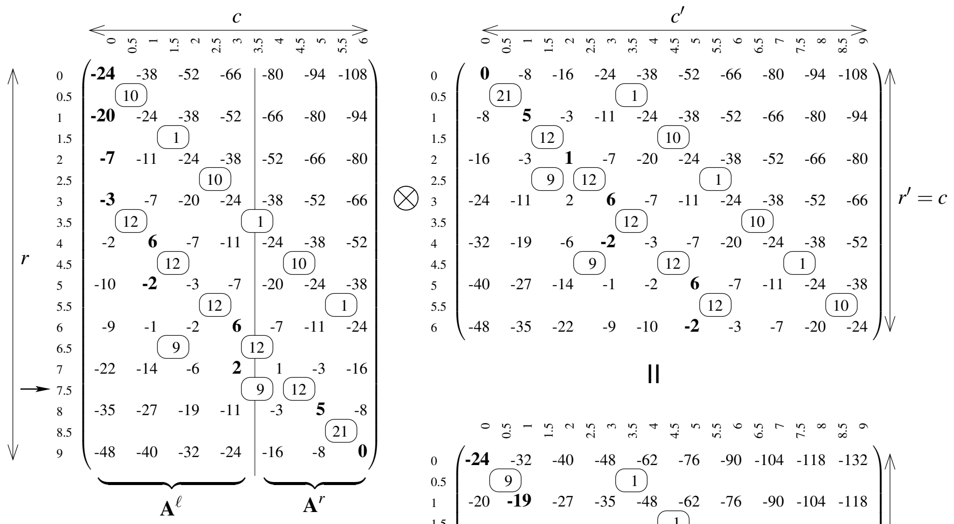
$$d(i,j) = \sum_{0<i'<i;\ 0<j'<j} D(i',j') \qquad \text{for all } 0 \leq i < r \text{ and } 0 \leq j < c \tag{2}$$

The next Lemma shows an alternative characterization of Monge matrices.

Lemma 1 ([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*

$$\mathbf{A}(i,j) = d(i,j) + u(i) + t(j) \qquad \textit{for all } 0 \leq i < r \textit{ and } 0 \leq j < c \tag{3}$$

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



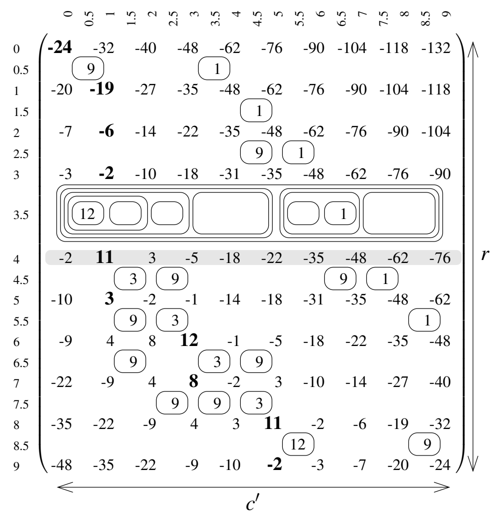


Fig. 1. Three Monge matrices, $\mathbf{A}$ (top-left), $\mathbf{B}$ (top-right) and $\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$ (bottom). The row and column indexes are shown outside. The leftmost maximums, per row, are in bold. The non-zero $\Delta\mathbf{A}(i,i+1;j,j+1) = D(i+0.5, j+0.5)$ values are inside rectangles. The same information is shown for $\mathbf{B}$ and $\mathbf{C}$. Row 4 of $\mathbf{C}$ is over a darker background. Matrix $\mathbf{A}$ is divided into $\mathbf{A}^\ell$ and $\mathbf{A}^r$ by a vertical line and $\mathrm{TR}(\mathbf{A}) = 7.5$ is pointed by an arrow. The rectangles between rows 3 and 4 of $\mathbf{C}$ illustrate the procedure in Lemma 7.

The proof of the lemma follows by defining $D(i+0.5, j+0.5) = \Delta\mathbf{A}(i,i+1;j,j+1)$ and observing that the $\Delta\mathbf{A}$ values are additive. The non-zero entries of D form the core of $\mathbf{A}$, were $\delta(\mathbf{A})$ denotes its size. A matrix is considered sparse when $\delta(\mathbf{A}) = o(rc)$. Notice that $\Delta\mathbf{A}(i,i';j,j') = \Delta d(i,i';j,j')$, therefore computing the expression consists in summing the core entries inside $[i,i'] \times [j,j']$, for example $\Delta\mathbf{A}(3,5;0,2) = 12+12$.

Definition 1. *For arbitrary 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. 1 shows a sample max-plus matrix product. To perform the calculation we organize it into a sequence of *intermediate computation matrices*, $\mathbf{M}$s, 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. 3 shows $\mathbf{M}_2$, $\mathbf{M}_3$, $\mathbf{M}_4$ and $\mathbf{M}_5$, ignore the tree-like structure on top. Each cell in the table contains an entry of the four $\mathbf{M}$s. The bottom shows the calculation of the $\mathbf{M}$ values of cell $(4,2)$. These matrices are used to compute the values of $\mathbf{C}$, since $\mathbf{C}(i,j) = \mathbf{M}_i(j, \operatorname{lax} \mathbf{M}_i[j, \_])$. For example row 4 of $\mathbf{C}$ can be obtained from $\mathbf{M}_4$. Observe that the bottom-left values of

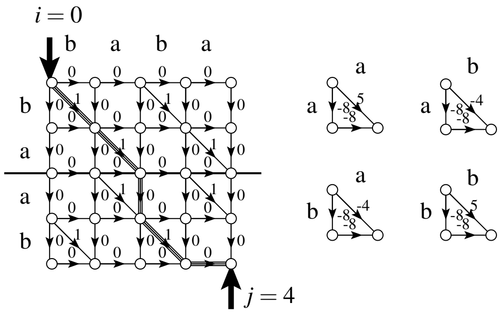


Fig. 2. (Left) Alignment DAGs of $S = ba$ and $T = baba$ and of $S' = ab$ with T. The two DAGs are united, horizontally, into the DAG of $S.S'$ and T. The horizontal line indicates the union. A highest-scoring path is represented by outlined arrows. This path corresponds $\mathrm{LCS}(S.S',T) = bab$, of size 3. (Right) Sample weights for Weighted Longest Common Subsequences.

each cell that are over a darker background correspond to row 4 of $\mathbf{C}$. The $\mathbf{M}_i$ matrices are Monge, because $\mathbf{B}$ is Monge, therefore there is a non-naive way to compute $\mathbf{C}$.

Lemma 2. *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be Monge matrices, of sizes* $r \times c$ *and* $(c = r') \times c'$, *then the values of* $\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$ *can be obtained in* $O(rr' \max\{1, \log(c'/r')\})$ *time.*

Proof. Since the $\mathbf{M}_i$ matrices are Monge the SMAWK [12] algorithm obtains all the row maximums, of each $\mathbf{M}_i$, in $O(r' \max\{1, \log(c'/r')\})$ time. □

This paper does not explain the SMAWK algorithm, the interested reader should consult Aggarwall et al. [12]. A simple recursive algorithm finds the maximum of the middle row and divides $\mathbf{M}$ into two smaller sub-problems. For $\mathbf{M}$ of size $c' \times r'$ this process takes only $O(r' \log c')$ time. The MMT algorithm uses regularities of the $\mathbf{M}_i$ matrices and a variation of this algorithm.

2.1 Highest Score Matrices

String alignment problems are a source of Monge matrices. 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 $\mathrm{LCS}(S.S',T) = bab$.

$\mathrm{LCS}(S,T)$ 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. See Fig. 2 for an example of such a DAG, notice that there is a diagonal edge on the top-left corner because S and T both start by b. A path corresponding to an LCS between $S.S'$ and T is highlighted. 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[5], *i.e.*, $\mathbf{H}_{S,T}(i,j) = \mathrm{LCS}(S, T[i..j-1])$. This matrix was denominated

[5] General $\mathbf{H}_{S,T}(i,j)$ values correspond to highest scoring paths on an infinite EADAG [9].

as $\textsc{Dist}(S,T)$ by Apostolico et al. [2], for edit distance problems. In this paper we follow the theory by Tiskin [9, 10] but the results are essentially equivalent.

In this context the max-plus product, Def. 1, 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 are Monge [2], 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. Therefore they can be multiplied efficiently.

Theorem 1 ([9]). *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 [13], namely Cyclic LCS, Longest Repeated Subsequence, Fully-Incremental LCS [5, 6], etc. The latter problem consists in 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 programing direction and therefore a naive approach requires $O(mn)$ time. Using Theorem 1 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 [6] of $O(n)$ time.

The concept of LCS can be extended to generic alignments, where the weight of the edges is an integer that depends on the characters involved. The resulting HSMs are generic Monge matrices, not necessarily unit. A simple case, Weighted Longest Common Subsequences (WLCS), occurs when the weight of the edges depends only on the type of edge, see the right part of Fig. 2.

3 Core Sensitive Multiplication

This section describes the Multiple Maxima Trees (MMT) algorithm for the max-plus product of Monge matrices. 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 $\mathbf{C}$.

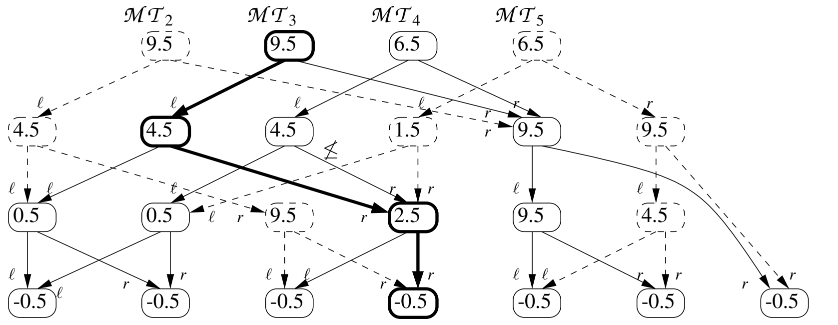
3.1 Representing $\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$

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. 1 $\mathbf{A}^\ell$ 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 2. *The* ***transition point*** $\textsc{Tr}(\mathbf{A})$, *of a monotone matrix* $\mathbf{A}$, *is a half integer such that* $\operatorname{lax}\mathbf{A}[i,\_]$ *is* $\operatorname{lax}\mathbf{A}^\ell[i,\_]$ *for* $i < \textsc{Tr}(\mathbf{A})$ *and* $\lceil c/2 \rceil + \operatorname{lax}\mathbf{A}^r[i,\_]$ *for* $i > \textsc{Tr}(\mathbf{A})$.

Fig. 1 indicates that $\textsc{Tr}(\mathbf{A}) = 7.5$ with an arrow. Fig. 3 shows that $\textsc{Tr}(\mathbf{M}_4) = 6.5$.

Definition 3. *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* $\textsc{Root}$ *are* $\mathcal{MT}_{\mathbf{A}^\ell}$ *and* $\mathcal{MT}_{\mathbf{A}^r}$. *The* $\textsc{Root}$ *stores* $\textsc{Tr}(\mathbf{A})$, *the remaining nodes store the corresponding transition points.*



	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,

$\mathbf{M}_2(4,2){=}\mathbf{A}(2,2)+\mathbf{B}(2,4){=}-24-20{=}-44;\quad \mathbf{M}_3(4,2){=}\mathbf{A}(3,2)+\mathbf{B}(2,4){=}-20-20{=}-40;$
$\mathbf{M}_4(4,2){=}\mathbf{A}(4,2)+\mathbf{B}(2,4){=}\ -7-20{=}-27;\quad \mathbf{M}_5(4,2){=}\mathbf{A}(5,2)+\mathbf{B}(2,4){=}\ -3-20{=}-23;$

Fig. 3. (Bottom) Calculation of the $\mathbf{M}$ values of cell $(4,2)$. (Middle) Matrices $\mathbf{M}_2$, $\mathbf{M}_3$, $\mathbf{M}_4$, $\mathbf{M}_5$ and the respective maxima trees. Each cell in the table shows a value from the four $\mathbf{M}$ matrices, $\mathbf{M}_2$ (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 maximums of $\mathbf{M}_4$ are over a darker background, these values correspond to row 4 of $\mathbf{C}$ in Fig. 1. (Top) the maxima trees of these matrices. The nodes and branches that are exclusive to trees $\mathcal{MT}_2$ or $\mathcal{MT}_5$ are dashed. Each node contains the transition point of the respective sub-matrix. The representation is compact, similar sub-trees are not repeated. The edges of the trees 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.

Fig. 3 shows the maxima trees for $\mathbf{M}_2$, $\mathbf{M}_3$, $\mathbf{M}_4$ and $\mathbf{M}_5$. At this point the reader should focus on an individual tree, $\mathcal{MT}_3$ for example[6]. It is a **fallacy** to assume that $\text{TR}(\mathbf{A}^\ell) \leq \text{TR}(\mathbf{A}) \leq \text{TR}(\mathbf{A}^r)$, Fig. 3 shows a counter example, indicated by $\not\leq$. The maxima tree can be used to compute lax values.

Lemma 3. *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* $\operatorname{lax}\mathbf{A}[i,\_]$ *can be computed in* $O(\log c)$ *time.*

Proof. We compute $\operatorname{lax}\mathbf{A}[i,\_]$ recursively as $\operatorname{lax}\mathbf{A}^\ell[i,\_]$ if $i < \text{TR}(\mathbf{A})$ and as $\lceil c/2 \rceil + \operatorname{lax}\mathbf{A}^r[i,\_]$ otherwise, *i.e.*, move to the left or to the right child. 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, *i.e.*, $O(c)$. □

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 4. *The maxima tree of a Monge matrix* $\mathbf{A}$ *takes* $O(c \log r)$ *time to build.*

Proof. The tree can be built bottom-up. Computing $\text{TR}(\mathbf{A})$ for the ROOT can be done, in $O(\log r)$ steps, with a binary search. If $\mathbf{A}^\ell(i, \operatorname{lax}\mathbf{A}^\ell[i,\_]) \geq \mathbf{A}^r(i, \operatorname{lax}\mathbf{A}^r[i,\_])$ then $\text{TR}(\mathbf{A}) > i$, otherwise $\mathbf{A}^\ell(i, \operatorname{lax}\mathbf{A}^\ell[i,\_]) < \mathbf{A}^r(i, \operatorname{lax}\mathbf{A}^r[i,\_])$ and $\text{TR}(\mathbf{A}) < i$. 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 3. 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 $\mathbf{C}(i,j) = \mathbf{M}_i(j, \operatorname{lax}\mathbf{M}_i[j,\_])$. This is inefficient. The $\mathbf{M}_i$ matrices have regularities that considerably reduce their requirements.

Lemma 5. *Given Monge matrices* $\mathbf{A}$ *and* $\mathbf{B}$*, when* $\Delta\mathbf{A}(i,i';k,k') = 0$ *we have that* $\mathbf{M}_i(j,k) \leq \mathbf{M}_i(j,k')$ *iff* $\mathbf{M}_{i'}(j,k) \leq \mathbf{M}_{i'}(j,k')$.

Proof. Notice the prime in $\mathbf{M}_{i'}$. The following diagram proves the Lemma:

$$\mathbf{M}_i(j,k) - \mathbf{M}_i(j,k') = \mathbf{A}(i,k) - \mathbf{A}(i,k') + \mathbf{B}(k,j) - \mathbf{B}(k',j) \tag{4}$$

$$\shortparallel \tag{5}$$

$$\mathbf{M}_{i'}(j,k) - \mathbf{M}_{i'}(j,k') = \mathbf{A}(i',k) - \mathbf{A}(i',k') + \mathbf{B}(k,j) - \mathbf{B}(k',j) \tag{6}$$

Eqs 4 and 6 follow from the Def. of $\mathbf{M}_i$. Equation 5 follows from the hypothesis. □

Notice that the $\Delta\mathbf{A}(i,i';k,k') = 0$ hypothesis 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 $k \leq k_1 \leq k_2 \leq k'$. This Lemma exposes the redundant information among the $\mathcal{MT}_{\mathbf{M}_i}$ trees. Namely if $\Delta\mathbf{A}(i,i';k,k') = 0$ and ϑ, ϑ' are nodes of $\mathcal{MT}_{\mathbf{M}_i}$ and $\mathcal{MT}_{\mathbf{M}_{i'}}$ whose corresponding rows are the $[k,k']$ interval then the sub-trees of ϑ and ϑ' are identical. Fig. 3 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 $\Delta\mathbf{A}(4,5;4,6) = 10 \neq 0$.

[6] Note that we simplified the notation.

Lemma 6. *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 4 to build the first tree, $\mathcal{MT}_1$, in $O(r' \log c')$ time. In general use Lemma 5 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. □

3.2 Obtaining the Core

Using the representation of $\mathbf{C}$ we can obtain its core. This section explains how.

Lemma 7. *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 is no core entry in $[i, i+1] \times [0, c'-1]$ and abandon 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 $O(\log c')$ for each core entry of $\mathbf{C}$ and $O(r)$ to consider every row of $\mathbf{C}$. □

Fig. 1 shows an illustration of the procedure described in the Lemma, between rows 3 and 4 of $\mathbf{C}$. We can now combine Lemmas 6 and 7 to obtain our main result.

Theorem 2. *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 6 and 7 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 7 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 3 and proof of Lemma 6. □

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.

4 Analysis

This section presents a theoretical and empirical analysis of the several multiplication algorithms. Up to now the time to access an entry of $\mathbf{A}$ or $\mathbf{B}$ has been omitted. Since the working space can be $o(rc)$, it is not reasonable to assume $O(1)$ access time. For unit-Monge matrices this problem was solved [9] using a dominance counting structure [11], which works in $O((\log \delta(\mathbf{A}))/\log\log \delta(\mathbf{A}))$ time and $O(\delta(\mathbf{A}))$ space. A similar result can be obtained with a wavelet tree [14]. Moreover by adding accumulated values by level it is possible to obtain the values of general Monge matrices, in $O(\log \delta(\mathbf{A}))$ time and $O(\delta(\mathbf{A}) \log \delta(\mathbf{A}))$ space.

$\delta = \Theta(n^{\varepsilon})$	MMT, Theorem 2	SMAWK , Lemma 2	Theorem 1
$\varepsilon = 2$	$\langle O(n^2), O(n^2 \log n)\rangle$	$\langle O(n^2), O(n^2)\rangle$	—
$2 < \varepsilon \leq 1$	$\langle O(n^{\varepsilon} \log n), O(n^{\varepsilon} \log^3 n)\rangle$	$\langle O(n^{\varepsilon} \log n), O(n^2 \log n)\rangle$	—
unit-Monge, $\varepsilon = 1$	$\langle O(n), O(n \log^3 n)\rangle$	$\langle O(n), O(\frac{n^2 \log n}{\log\log n})\rangle$	$\langle O(n), O(\frac{n \log^2 n}{\log\log n})\rangle$
$1 < \varepsilon \leq 0$	$\langle O(n), O(n \log^2 n)\rangle$	$\langle O(n), O(n^2 \log n)\rangle$	—

Table 1. Comparison between max-plus multiplication algorithms, accounting for access time to **A** and **B**, $O(1)$ for $\varepsilon = 2$, $O(\log n/\log\log n)$ for unit-Monge and $O(\log n)$ otherwise. For $\varepsilon \geq 1$ and $\varepsilon \neq 2$ the performance of MMT is unaltered even if access time is $O(1)$. The $\langle s(n), t(n)\rangle$ notation means $s(n)$ space and $t(n)$ time requirements.

To simplify the analysis let us assume that $O(r) = O(c) = O(r') = O(c') = O(n)$ and that $O(\delta(\mathbf{A})) = O(\delta(\mathbf{B})) = O(\delta(\mathbf{C})) = O(\delta)$. The algorithm of Theorem 1 needs to access entries at each step. Hence the access value becomes a factor of the overall time. The MMT algorithm 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 4 the lax operation is $\Omega(\log n)$, see Lemma 3. Therefore the access time does not become a time factor in MMT. This claim is confirmed experimentally, in Section 4.1.

Table 1 shows a comparison of the different algorithms according to core size, $\delta = \Theta(n^{\varepsilon})$. The analysis considers the best space and time requirements that includes the access time to **A** and **B**. For $\varepsilon = 2$ the access time is $O(1)$, in this case Lemmas 6 and 7 have amortized performance, hence the $o(n^2 \log^3 n)$ time. Lemma 6 becomes an iterated Lemma 4. Lemma 7 loses a $O(\log n)$ factor. For $\varepsilon < 1$ the access time is $O(\log n)$, the dominating time is that of building $\mathcal{MT}_1$, Lemma 4. Because of the amortized analysis we need to count the access time, hence the $\Omega(n \log n)$ time.

Table 1 shows that the MMT algorithm is not always the most efficient, in particular SMAWK is faster by an $O(\log n)$ factor for $\delta = \Theta(n^2)$. This factor quickly disappears for $\varepsilon < 2$, 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 MMT is faster than SMAWK . For the particular case of unit-Monge matrices, with $\delta = \Theta(n)$, the algorithm of Theorem 1 is faster than MMT by a $O(\log n \log\log n)$ factor. However the experimental results show that in practice MMT is faster than what is predicted by theory.

Using MMT we obtain the first non-trivial solution for the Fully-Incremental Alignment problem, which consists in updating a generic alignment score from $\textsc{Align}(S,T)$ to $\textsc{Align}(c.S,T)$, $\textsc{Align}(S,c.T)$, $\textsc{Align}(S.c,T)$ or $\textsc{Align}(S,T.c)$, where $\textsc{Align}$ is the respective score. The resulting procedure takes $O((n+\delta)\log^3(n+\delta))$ time and $O(n + \delta \log \delta)$ space, where δ is the core size of the intervening matrices.

4.1 Experimental Results

We implemented MMT and tested it on an Intel Core2 Duo @1.33GHz, 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 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. To test the performance we multiplied HSM matrices, obtained from the LCS, as defined in Section 1. We also tested HSMs that resulted from generic alignments with the PAM weight matrix [15] and HSMs from Weighted Longest Common Subsequences (WLCS), using the weights in Fig. 2. The algorithm of Theorem 1 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 1 cannot be adapted for 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. The results are shown in Fig. 4. 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 seconds.

The plots on the left show the running time, in seconds, of the MMT 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 [10]. 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 for c and d in $O(cN \log^d N)$. Since the behavior is asymptotic the first points may distort the values. To diminish this effect we computed several MSQ estimates, successively discarding the first points. We pessimistically chose the largest estimate.

The results show that for LCS and WLCS the δ values are similar to n, but not for PAM, notice the dispersion of the simulated points. Using $O(1)$ access time MMT is slower than the simulated algorithm for LCS and WLCS but, generally, faster for PAM. Contrary to what is predicted in Table 1, 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 $O(Nlog^2 N)$.

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/Leafs), 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 a $O(\log n)$ 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 $o(\log^1 N)$ because the MMT prototype uses inverse binary search. Instead of dividing an interval, it starts from a point and moves in powers of 2. Hence it usually runs on a small interval. 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.

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

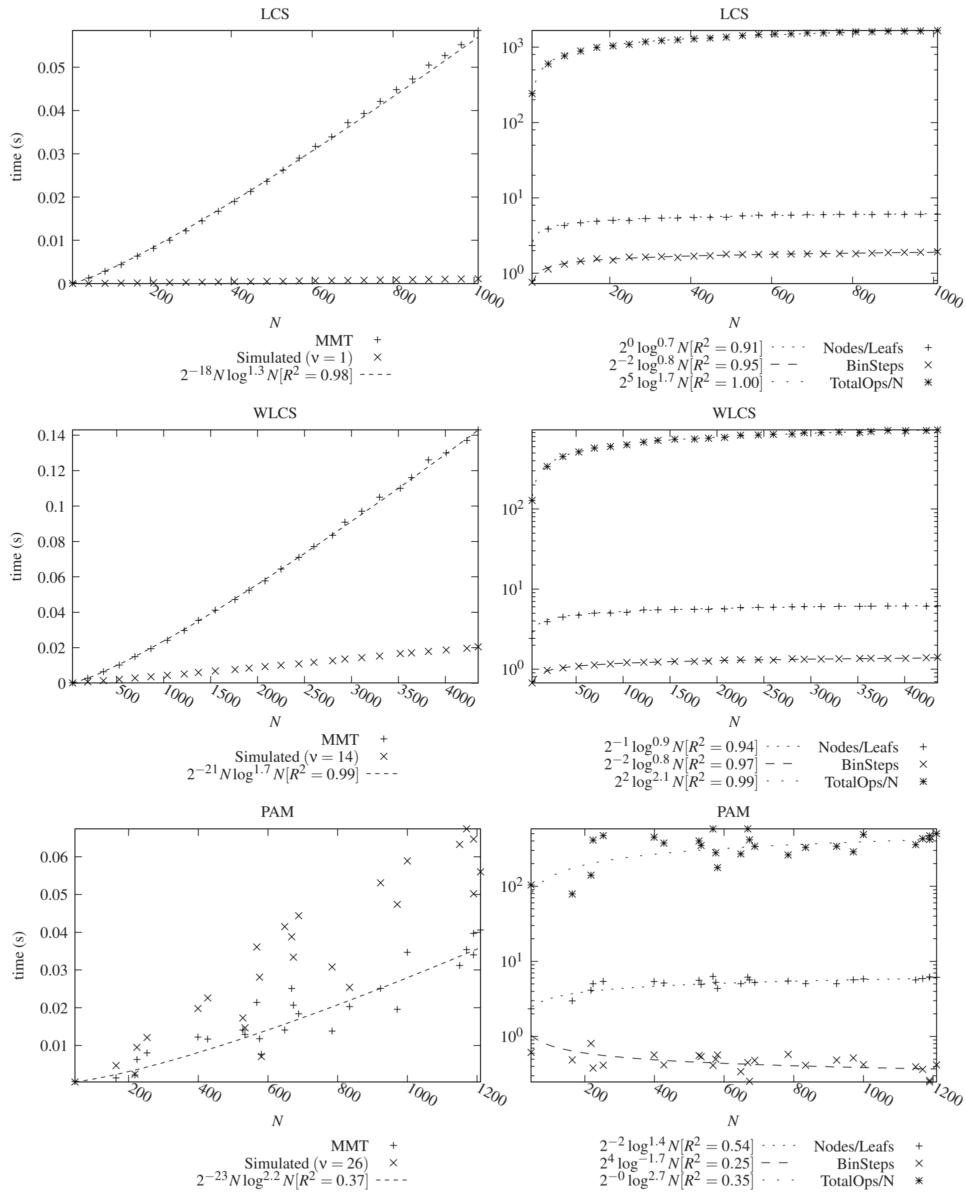


Fig. 4. Experimental testing of the MMT algorithm. The x-axis of the represents variable $N = \max\{n + \delta\}$. The y-axis of the graphs on the left measures the time in seconds. The y-axis of the graphs on the right measures the number of operations.

that the existing algorithms are either sub-optimal or apply only to specific classes of matrices. Alternatively we proposed a core sensitive algorithm, MMT. 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 1 is theoretically faster than MMT, by an $O(\log n \log\log n)$ factor, which does not really hold in practice, *i.e.*, in practice MMT is faster than the theoretical bound. The algorithm of Theorem 1 can also be applied to matrices that result from Weighted LCSs, by using a ν time and space factor.

The MMT algorithm is simple and flexible, its main tools are binary trees and binary searches. Moreover it can uniformly hand all sorts of Monge matrices, namely it can multiply HSMs that result from PAM alignments, whereas the algorithm of Theorem 1 cannot. This flexibility accounted for the first, as far as we know, non-trivial algorithm for the Fully-Incremental Alignment problem, a string processing problem that is relevant for bio-informatics. We expect MMT to have broad applications, since a myriad [9, 16] of string processing and optimization problems [1] use Monge matrices.

Acknowledgments: We thank anonymous reviewers for several insightful remarks.

References

1. Burkard, R., Klinz, B., Rudolf, R.: Perspectives of Monge properties in optimization. Discrete Applied Mathematics **70**(2) (1996) 95–161
2. Apostolico, A., Atallah, M.J., Larmore, L.L., McFaddin, S.: Efficient parallel algorithms for string editing and related problems. SIAM J. Comput. **19**(5) (1990) 968–988
3. Schmidt, J.: 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
4. Tiskin, A.: Semi-local longest common subsequences in subquadratic time. J. Discrete Algorithms **6**(4) (2008) 570–581
5. Landau, G., Myers, E., Schmidt, J.: Incremental string comparison. SIAM Journal on Computing **27** (1998) 557–582
6. Ishida, Y., Inenaga, S., Shinohara, A., Takeda, M.: Fully incremental LCS computation. In: Fundamentals of Computation Theory, Springer 563–574
7. Alves, C.E.R., Cáceres, E.N., Song, S.W.: An all-substrings common subsequence algorithm. Discrete Applied Mathematics **156**(7) (2008) 1025–1035
8. Landau, G.M.: Can dist tables be merged in linear time – An open problem. In: Proceedings of the Prague Stringology Conference '06. (2006)
9. Tiskin, A.: Fast distance multiplication of unit-monge matrices. In: ACM-SIAM SODA. (2010) 1287–1296
10. Tiskin, A.: Semi-local string comparison: algorithmic techniques and applications. ArXiv e-prints (July 2007) 2007arXiv0707.3619T.
11. JáJá, J., Mortensen, C.W., Shi, Q.: Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In: ISAAC. (2004) 558–568
12. Aggarwal, A., Klawe, M., Moran, S., Shor, P., Wilber, R.: Geometric applications of a matrix-searching algorithm. Algorithmica **2**(1) (1987) 195–208
13. Iliopoulos, C.S., Mouchard, L., Rahman, M.S.: A new approach to pattern matching in degenerate DNA/RNA sequences and distributed pattern matching. Mathematics in Computer Science **1**(4) (2008) 557–569
14. Grossi, R., Gupta, A., Vitter, J.S.: High-order entropy-compressed text indexes. In: SODA. (2003) 841–850
15. Dayhoff, M.O., Schwartz, R.M., Orcutt, B.C.: A model of evolutionary change in proteins. Atlas of protein sequence and structure **5**(suppl 3) (1978) 345–351
16. Crochemore, M., Landau, G., Ziv-Ukelson, M.: A subquadratic sequence alignment algorithm for unrestricted scoring matrices. SIAM journal on computing **32**(6) (2003) 1654–1673