

Fast Distance Multiplication of Unit-Monge Matrices

Alexander Tiskin

Received: 3 November 2011 / Accepted: 21 August 2013 / Published online: 19 September 2013
© The Author(s) 2013. This article is published with open access at Springerlink.com

Abstract Monge matrices play a fundamental role in optimisation theory, graph and string algorithms. Distance multiplication of two Monge matrices of size n can be performed in time $O(n^2)$. Motivated by applications to string algorithms, we introduced in previous works a subclass of Monge matrices, that we call *simple unit-Monge* matrices. We also gave a distance multiplication algorithm for such matrices, running in time $O(n^{1.5})$. Landau asked whether this problem can be solved in linear time. In the current work, we give an algorithm running in time $O(n \log n)$, thus approaching an answer to Landau's question within a logarithmic factor. The new algorithm implies immediate improvements in running time for a number of algorithms on strings and graphs. In particular, we obtain an algorithm for finding a maximum clique in a circle graph in time $O(n \log^2 n)$, and a surprisingly efficient algorithm for comparing compressed strings. We also point to potential applications in group theory, by making a connection between unit-Monge matrices and Coxeter monoids. We conclude that unit-Monge matrices are a fascinating object and a powerful tool, that deserve further study from both the mathematical and the algorithmic viewpoints.

Keywords String comparison · Unit-Monge matrices · Seaweed braids · 0-Hecke monoid · Circle graphs

1 Introduction

A matrix is called *Monge*, if its density matrix is nonnegative. Monge matrices play a fundamental role in optimisation theory, graph and string algorithms. Distance multiplication (also known as min-plus or tropical multiplication) of two Monge matrices

Research supported by the Centre for Discrete Mathematics and its Applications (DIMAP), University of Warwick, EPSRC award EP/D063191/1.

A. Tiskin (✉)

Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK
e-mail: tiskin@dcs.warwick.ac.uk

of size n can be performed in time $O(n^2)$. Motivated by applications to string comparison, we introduced (using different terminology) in [60, 61] the following subclass of Monge matrices. A matrix is called *unit-Monge*, if its density matrix is a permutation matrix; we further restrict our attention to a subclass of *simple unit-Monge* matrices, which satisfy a straightforward boundary condition. In [60, 61], we gave an algorithm for distance multiplication of such matrices, running in time $O(n^{1.5})$. Landau [43] asked (again using different terminology) whether this problem can be solved in linear time. In the current work, we give an algorithm for distance multiplication of simple unit-Monge matrices, running in time $O(n \log n)$, thus approaching an answer to Landau's question within a logarithmic factor.

Our study of unit-Monge matrices is motivated primarily by their applications to string comparison and approximate pattern matching in strings. We presented a number of such algorithmic applications in [41, 42, 61–63, 65], and two biological applications in [7, 52]. Our new distance multiplication algorithm implies immediate improvements in running time for a number of string comparison and graph algorithms: semi-local longest common subsequences between permutations; longest increasing subsequence in a cyclic permutation; maximum clique in a circle graph; longest common subsequence between a grammar-compressed string and an uncompressed string. In the current work, we give a brief overview of these applications and the improvements brought about by faster distance multiplication of simple unit-Monge matrices.

This paper is a revised and extended version of [64]. Further details and applications can be found in [57].

2 Terminology and Notation

For indices, we will use either integers, or half-integers

$$\{\dots, -2, -1, 0, 1, 2, \dots\}$$

$$\left\{ \dots, -\frac{5}{2}, -\frac{3}{2}, -\frac{1}{2}, \frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \dots \right\}$$

For ease of reading, half-integer variables will be indicated by hats (e.g. \hat{i} , \hat{j}). It will be convenient to denote

$$i^- = i - \frac{1}{2} \quad i^+ = i + \frac{1}{2}$$

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

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

We denote integer and half-integer *intervals* by

$$[i : j] = \{i, i+1, \dots, j-1, j\}$$

$$\langle i : j \rangle = \left\{ i + \frac{1}{2}, i + \frac{3}{2}, \dots, j - \frac{3}{2}, j - \frac{1}{2} \right\}$$

A function of an integer argument will be called *unit-monotone increasing*, if for every successive pair of values, the difference between the successor and the predecessor is either 0 or 1.

We will make extensive use of matrices with integer elements, and with integer or half-integer indices.¹ Given two index ranges I, J , it will be convenient to denote their Cartesian product by $(I \mid J)$. We extend this notation to Cartesian products of intervals:

$$[i_0 : i_1 \mid j_0 : j_1] = ([i_0 : i_1] \mid [j_0 : j_1])$$

$$\langle i_0 : i_1 \mid j_0 : j_1 \rangle = (\langle i_0 : i_1 \rangle \mid \langle j_0 : j_1 \rangle)$$

Given index ranges I, J , a *matrix over* $(I \mid J)$ is indexed by $i \in I, j \in J$. A matrix is *nonnegative*, if all its elements are nonnegative.

The matrices we consider can be *implicit*, i.e. represented by a compact data structure that supports access to every matrix element in a specified (typically small, but not necessarily constant) time. If the query time is not given, it is assumed by default to be constant.

We will use the parenthesis notation for indexing matrices, e.g. $A(i, j)$. We will also use a straightforward notation for selecting subvectors and submatrices: for example, given a matrix A over $[0 : n \mid 0 : n]$, we denote by $A[i_0 : i_1 \mid j_0 : j_1]$ the submatrix defined by the given sub-intervals. A star $*$ will indicate that for a particular index, its whole range is used, e.g. $A[* \mid j_0 : j_1] = A[0 : n \mid j_0 : j_1]$.

We will denote by A^T the transpose of matrix A , and by A^R the matrix obtained from A by counterclockwise 90-degree rotation. Given a matrix A over $[0 : n \mid 0 : n]$ or $\langle 0 : n \mid 0 : n \rangle$, we have

$$A^T(i, j) = A(j, i) \quad A^R(i, j) = A(j, n - i)$$

for all i, j .

We now introduce two fundamental combinatorial operations on matrices. The first operation obtains an integer-indexed matrix from a half-integer-indexed matrix by summing up, for each of the integer points, all matrix elements that lie below and to the left of the given point.

Definition 1 Let D be a matrix over $\langle i_0 : i_1 \mid j_0 : j_1 \rangle$. Its *distribution matrix* D^Σ over $[i_0 : i_1 \mid j_0 : j_1]$ is defined by

$$D^\Sigma(i, j) = \sum_{\hat{i} \in \langle i : i_1 \rangle, \hat{j} \in \langle j_0 : j \rangle} D(\hat{i}, \hat{j})$$

for all $i \in [i_0 : i_1], j \in [j_0 : j_1]$.

¹When integers and half-integers are used as matrix indices, it is convenient to imagine that the matrices are written on squared paper. The entries of an integer-indexed matrix are at integer points of line intersections; the entries of a half-integer-indexed matrix are at half-integer points within the squares.

The second operation obtains a half-integer-indexed matrix from an integer-indexed matrix, by taking a four-point difference around each given point.

Definition 2 Let A be a matrix over $[i_0 : i_1 \mid j_0 : j_1]$. Its *density matrix* A^\square over $\langle i_0 : i_1 \mid j_0 : j_1 \rangle$ is defined by

$$A^\square(\hat{i}, \hat{j}) = A(\hat{i}^+, \hat{j}^-) - A(\hat{i}^-, \hat{j}^-) - A(\hat{i}^+, \hat{j}^+) + A(\hat{i}^-, \hat{j}^+)$$

for all $\hat{i} \in \langle i_0 : i_1 \rangle$, $\hat{j} \in \langle j_0 : j_1 \rangle$.

The operations of taking the distribution and the density matrix are close to be mutually inverse. For any finite matrices D , A as above, and for all i, j , we have

$$D^{\Sigma\square} = D$$

$$A^{\square\Sigma}(i, j) = A(i, j) - A(i_1, j) - A(i, j_0) + A(i_1, j_0)$$

When matrix A is restricted to have all zeros on its bottom-left boundary (i.e. in the leftmost column and the bottom row), the two operations become truly mutually inverse. We introduce special terminology for such matrices.

Definition 3 Matrix A will be called *simple*, if $A(i_1, j) = A(i, j_0) = 0$ for all i, j . Equivalently, A is simple if $A^{\square\Sigma} = A$.

The following class of matrices plays an important role in optimisation theory (see Burkard et al. [13] for an extensive survey), and also arises in graph and string algorithms.

Definition 4 Matrix A is called a *Monge matrix*, if

$$A(i, j) + A(i', j') \leq A(i, j') + A(i', j)$$

for all $i \leq i'$, $j \leq j'$. Equivalently, matrix A is a Monge matrix, if A^\square is nonnegative.

Our techniques will rely on structures that have permutations as their basic building blocks. We will be dealing with permutations in matrix form, exploiting the symmetry between indices and elements of a permutation.

Definition 5 A *permutation* (respectively, *subpermutation*) *matrix* is a zero-one matrix containing exactly one (respectively, at most one) nonzero in every row and every column.

Typically, permutation and subpermutation matrices will be indexed by half-integers. An *identity matrix* is a permutation matrix Id over an interval range $\langle i_0 : i_1 \mid i_0 : i_1 \rangle$, such that $Id(\hat{i}, \hat{j}) = 1$, iff $\hat{i} = \hat{j}$.

When dealing with (sub)permutation matrices, we will write “nonzeros” for “index pairs corresponding to nonzeros”, as long as this does not lead to confusion.

Due to the extreme sparsity of (sub)permutation matrices, it would obviously be wasteful and inefficient to store them explicitly. Instead, we will normally assume that a permutation matrix P of size n is given implicitly by the underlying permutation and its inverse, i.e. by a pair of arrays π, π^{-1} , such that $P(\hat{i}, \pi(\hat{i})) = 1$ for all \hat{i} , and $P(\pi^{-1}(\hat{j}), \hat{j}) = 1$ for all \hat{j} . This compact representation has size $O(n)$, and allows constant-time access to each nonzero of P by its row index, as well as by its column index. The implicit representation for subpermutation matrices is analogous.

The following subclasses of Monge matrices will play a crucial role in this work.

Definition 6 Matrix A is called a *unit-Monge* (respectively, *subunit-Monge*) matrix, if A^\square is a permutation (respectively, subpermutation) matrix.

By Definitions 4, 6, any unit-Monge matrix is subunit-Monge, and any subunit-Monge matrix is Monge (since the corresponding density matrix A^\square is a (sub)permutation matrix, and hence nonnegative).

Our particular focus will be on square matrices A that are both simple and unit-Monge. By Definitions 3, 6, this holds if and only if $A = P^\Sigma$, where P is a permutation matrix. Matrix P can be regarded as an implicit representation of the simple unit-Monge matrix $A = P^\Sigma$.

Example The following matrix is simple unit-Monge:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}^\Sigma = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

In our algorithms, we will need to access matrix elements via *incremental queries*. Given an element of an implicit simple (sub)unit-Monge matrix, such a query returns the value of a specified adjacent element.

Theorem 1 Given a permutation matrix P of size n , and the value $P^\Sigma(i, j)$, $i, j \in [0 : n]$, the values $P^\Sigma(i \pm 1, j)$, $P^\Sigma(i, j \pm 1)$, where they exist, can be queried in time $O(1)$.

Proof Straightforward from the definitions; see [57] for details. \square

3 Matrix Distance Multiplication

3.1 Distance Multiplication Monoids

The $(\min, +)$ -semiring of integers is one of the fundamental structures in algorithm design. In this semiring, the operators \min and $+$, denoted by \oplus and \odot , play the role of addition and multiplication, respectively. The $(\min, +)$ -semiring is often called *distance* (or *tropical*) algebra. For a detailed introduction into this and related topics,

see e.g. Rote [54], Gondran and Minoux [30], Butković [14]. An application of the distance algebra to string comparison has been previously suggested by Comet [16].

Throughout this section, vectors and matrices will be indexed by integers beginning from 0, or half-integers beginning from $0^+ = \frac{1}{2}$. All our definitions and statements can easily be generalised to indexing over arbitrary integer or half-integer intervals.

Multiplication in the $(\min, +)$ -semiring of integers can be naturally extended to integer matrices.

Definition 7 Let A, B, C be matrices over $[0 : n_1 \mid 0 : n_2]$, $[0 : n_2 \mid 0 : n_3]$, $[0 : n_1 \mid 0 : n_3]$ respectively. The *matrix distance product* $A \odot B = C$ is defined by

$$C(i, k) = \bigoplus_{j \in [0 : n_2]} (A(i, j) \odot B(j, k)) = \min_{j \in [0 : n_2]} (A(i, j) + B(j, k))$$

for all $i \in [0 : n_1]$, $k \in [0 : n_3]$.

We now consider three different monoids of integer matrices with respect to matrix distance multiplication.

Monoid of All Nonnegative Matrices Consider the set of all square matrices with elements in $[0 : +\infty]$ over a fixed index range. This set forms a monoid with zero with respect to distance multiplication. The identity and the zero element in this monoid are respectively the matrices

$$Id_{\odot}(i, j) = \begin{cases} 0 & \text{if } i = j \\ +\infty & \text{otherwise} \end{cases} \quad O_{\odot}(i, j) = +\infty$$

for all i, j . For any matrix A , we have

$$A \odot Id_{\odot} = Id_{\odot} \odot A = A \quad A \odot O_{\odot} = O_{\odot} \odot A = O_{\odot}$$

Monge Monoid It is well-known (see e.g. [6]) that the set of all Monge matrices is closed under distance multiplication.

Theorem 2 Let A, B, C be matrices, such that $A \odot B = C$. If A, B are Monge, then C is also Monge.

Proof Let A, B be over $[0 : n_1 \mid 0 : n_2]$, $[0 : n_2 \mid 0 : n_3]$, respectively. Let $i', i'' \in [0 : n_1]$, $i' \leq i''$, and $k', k'' \in [0 : n_3]$, $k' \leq k''$. By definition of matrix distance multiplication, we have

$$C(i', k'') = \min_j (A(i', j) + B(j, k''))$$

$$C(i'', k') = \min_j (A(i'', j) + B(j, k'))$$

Let j', j'' respectively be the values of j on which these minima are attained. Suppose $j' \leq j''$. We have

$$\begin{aligned}
 & C(i', k') + C(i'', k'') \quad (\text{definition of } \odot) \\
 &= \min_j (A(i', j) + B(j, k')) + \min_j (A(i'', j) + B(j, k'')) \quad (\text{minimisation over } j) \\
 &\leq (A(i', j') + B(j', k')) + (A(i'', j'') + B(j'', k'')) \quad (\text{term rearrangement}) \\
 &= (A(i', j') + A(i'', j'')) + (B(j', k') + B(j'', k'')) \quad (A \text{ is Monge}) \\
 &\leq (A(i', j'') + A(i'', j')) + (B(j', k') + B(j'', k'')) \quad (\text{term rearrangement}) \\
 &= (A(i', j'') + B(j'', k'')) + (A(i'', j') + B(j', k')) \quad (\text{definition of } j', j'') \\
 &= C(i', k'') + C(i'', k')
 \end{aligned}$$

The case $j' \geq j''$ is treated symmetrically, making use of the Monge property of B . Hence, matrix C is Monge. \square

Theorem 2 implies that the set of all square nonnegative Monge matrices over a fixed index range forms a submonoid (the *Monge monoid*) in the distance multiplication monoid of all nonnegative matrices (where the range of elements has to be formally extended by $+\infty$).

The ambient monoid's identity Id_\odot and zero O_\odot are inherited by the Monge monoid. Indeed, in the expansion of their density matrices Id_\odot^\square and O_\odot^\square by Definition 2, all indeterminate expressions of the form $+\infty - \infty$ can be formally considered to be nonnegative. Therefore, matrices Id_\odot and O_\odot can be formally considered to be Monge matrices.

Unit-Monge Monoid It is somewhat surprising, but crucial for the development of our techniques, that the set of all simple (sub)unit-Monge matrices is also closed under distance multiplication.

Theorem 3 *Let A, B, C be matrices, such that $A \odot B = C$. If A, B are simple unit-Monge (respectively, simple subunit-Monge), then C is also simple unit-Monge (respectively, simple subunit-Monge).*

Proof Let A, B be simple unit-Monge matrices over $[0 : n \mid 0 : n]$. We have $A = P_A^\Sigma$, $B = P_B^\Sigma$, where P_A, P_B are permutation matrices. It is easy to check that matrix C is simple, therefore $C = P_C^\Sigma$ for some matrix P_C .

We now have $P_A^\Sigma \odot P_B^\Sigma = P_C^\Sigma$, and we need to show that P_C is a permutation matrix. Clearly, matrices C and P_C are both integer. Furthermore, matrix C is Monge by Theorem 2, and therefore matrix $C^\square = P_C$ is nonnegative.

Since P_B is a permutation matrix, we have

$$P_B^\Sigma(j, 0) = 0 \quad P_B^\Sigma(j, n) = n - j$$

for all $j \in [0 : n]$. Hence

$$C(i, 0) = \min_j (P_A^\Sigma(i, j) + P_B^\Sigma(j, 0)) = \min_j (P_A^\Sigma(i, j) + 0) = 0$$

$$C(i, n) = \min_j (P_A^\Sigma(i, j) + P_B^\Sigma(j, n)) = \min_j (P_A^\Sigma(i, j) + n - j) = n - i$$

for all $i \in [0 : n]$, since the minimum is attained respectively at $j = 0$ and $j = n$. Therefore, we have

$$\begin{aligned} & \sum_{\hat{k}} P_C(\hat{i}, \hat{k}) \quad (\text{definition of } \Sigma \text{ and } \square) \\ &= \sum_{\hat{k}} (C(\hat{i}^+, \hat{k}^-) - C(\hat{i}^-, \hat{k}^-) - C(\hat{i}^+, \hat{k}^+) + C(\hat{i}^-, \hat{k}^+)) \quad (\text{term cancellation}) \\ &= C(\hat{i}^+, 0) - C(\hat{i}^-, 0) - C(\hat{i}^+, n) + C(\hat{i}^-, n) \\ &= 0 - 0 - (n - \hat{i}^+) + (n - \hat{i}^-) = 1 \end{aligned}$$

for all $\hat{i} \in \langle 0 : n \rangle$. Symmetrically, we have

$$\sum_{\hat{i}} P_C(\hat{i}, \hat{k}) = 1$$

for all $\hat{k} \in \langle 0 : n \rangle$. Taken together, the above properties imply that matrix P_C is a permutation matrix. Therefore, C is a simple unit-Monge matrix.

Finally, let A, B be simple subunit-Monge matrices over $[0 : n_1 \mid 0 : n_2], [0 : n_2 \mid 0 : n_3]$, respectively. We have $A = P_A^\Sigma, B = P_B^\Sigma$, where P_A, P_B are subpermutation matrices. As before, let $C = P_C^\Sigma$, for some matrix P_C ; we have to show that P_C is a subpermutation matrix. Suppose that for some \hat{i} , row $P_A(\hat{i}, *)$ contains only zeros. Then, it is easy to check that the corresponding row $P_C(\hat{i}, *)$ also contains only zeros, and that upon deleting rows $P_A(\hat{i}, *)$ and $P_C(\hat{i}, *)$ from the respective matrices, the equality $P_A^\Sigma \odot P_B^\Sigma = P_C^\Sigma$ still holds. Symmetrically, a zero column $P_B(*, \hat{k})$ results in a zero column $P_C(*, \hat{k})$, and upon deleting both these columns from the respective matrices, the equality $P_A^\Sigma \odot P_B^\Sigma = P_C^\Sigma$ still holds. Therefore, we may assume without loss generality that $n_1 \leq n_2, n_2 \geq n_3$, and that subpermutation matrix P_A (respectively, P_B) does not have any zero rows (respectively, zero columns).

Let us now extend matrix P_A to a square $n_2 \times n_2$ matrix $\begin{bmatrix} * & * \\ P_A & * \end{bmatrix}$, where the top $n_2 - n_1$ rows are filled by zeros and ones so that the resulting matrix is a permutation matrix. Likewise, let us extend matrix P_B to an $n_2 \times n_2$ permutation matrix $\begin{bmatrix} * & * \\ P_B & * \end{bmatrix}$. We now have

$$\begin{bmatrix} * & * \\ P_A & * \end{bmatrix}^\Sigma \odot \begin{bmatrix} P_B & * \end{bmatrix}^\Sigma = \begin{bmatrix} * & * \\ P_C & * \end{bmatrix}^\Sigma$$

where $\begin{bmatrix} * & * \\ P_C & * \end{bmatrix}$ is an $n_2 \times n_2$ permutation matrix, with matrix P_C occupying its lower-left corner. Hence, matrix P_C is a subpermutation matrix, and the original matrix C is a simple subunit-Monge matrix. \square

Theorem 3 implies that the set of all simple unit-Monge matrices over a fixed index range forms a submonoid (the *unit-Monge monoid*) in the Monge monoid.

Without loss of generality, let the matrices be over $[0 : n \mid 0 : n]$. The Monge monoid's identity Id_{\odot} and zero O_{\odot} are neither simple nor unit-Monge matrices, and therefore are not inherited by the unit-Monge monoid. Instead, its identity and zero elements are given respectively by the matrices

$$Id^{\Sigma}(i, j) = \max(j - i, 0) \quad Id^{R\Sigma}(i, j) = \min(n - i, j)$$

(recall that Id^R is the matrix obtained by 90-degree rotation of the identity permutation matrix Id). For any permutation matrix P , we have

$$P^{\Sigma} \odot Id^{\Sigma} = Id^{\Sigma} \odot P^{\Sigma} = P^{\Sigma} \quad P^{\Sigma} \odot Id^{R\Sigma} = Id^{R\Sigma} \odot P^{\Sigma} = Id^{R\Sigma}$$

Theorem 3 gives us the basis for performing distance multiplication of simple (sub)unit-Monge matrices implicitly, by taking the density (sub)permutation matrices as input, and producing a density (sub)permutation matrix as output. It will be convenient to introduce special notation for such implicit distance matrix multiplication.

Definition 8 Let P_A, P_B, P_C be (sub)permutation matrices. The *implicit matrix distance product* $P_A \boxdot P_B = P_C$ is defined by $P_A^{\Sigma} \odot P_B^{\Sigma} = P_C^{\Sigma}$.

The set of all permutation matrices over $\langle 0 : n \mid 0 : n \rangle$ is therefore a monoid with respect to implicit distance multiplication \boxdot . This monoid has identity element Id and zero element Id^R , and is isomorphic to the unit-Monge monoid. Note that, although defined on the set of all permutation matrices of size n , this monoid is substantially different from the symmetric group S_n , defined by standard permutation composition (equivalently, by standard multiplication of permutation matrices). In particular, the implicit distance multiplication monoid has a zero element Id^R , whereas S_n , being a group, cannot have a zero. More generally, the implicit distance multiplication monoid has plenty of idempotent elements (defined by involutive permutations), whereas S_n has the only trivial idempotent Id . However, both the implicit distance multiplication monoid and the symmetric group S_n still share the same identity element Id .

Example Figure 1a shows a triple of 6×6 permutation matrices P_A, P_B, P_C , such that $P_A \boxdot P_B = P_C$. Nonzeros are indicated by green² circles.

3.2 Seaweed Braids

Further understanding of the unit-Monge monoid (and, by isomorphism, of the implicit distance multiplication monoid of permutation matrices) can be gained via an

²For colour illustrations, the reader is referred to the online version of this work. If the colour version is not available, all references to colour can be ignored.

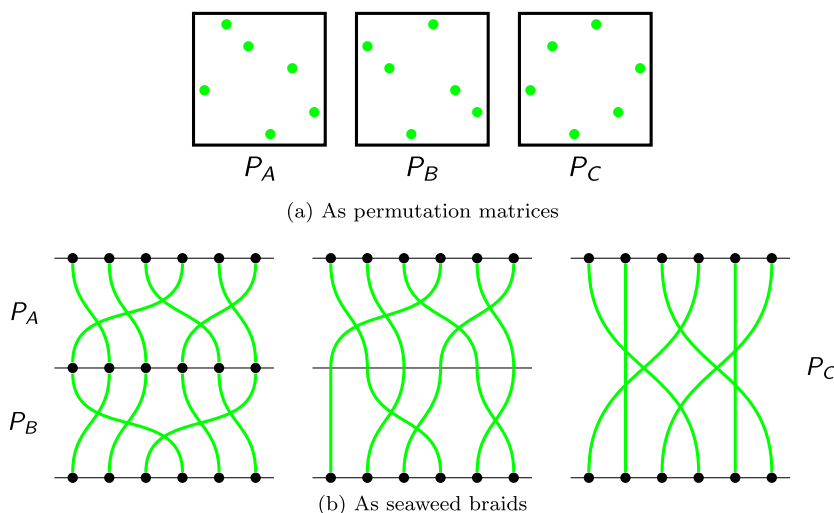


Fig. 1 Implicit matrix distance product $P_A \square P_B = P_C$

algebraic formalism closely related to braid theory. While this formalism is not necessary for obtaining the main results of this paper, we believe it to be of substantial independent interest. We refer the reader to [38] for the background on classical braid theory.

Consider two sets of n nodes each, drawn on two parallel horizontal lines in the Euclidean plane. We put the two node sets into one-to-one correspondence by connecting them pairwise, in some order, with continuous monotone curves. (Here, a curve is called monotone, if its vertical projection is always directed downwards.) These curves will be called *seaweeds*.³ We call the resulting configuration a *seaweed braid* of width n .

Example Figure 1b shows three different seaweed braids.

There is remarkable similarity between seaweed braids and classical braids. However, there is also a crucial difference: all crossings between seaweeds are “level crossings”, i.e. a pair of crossing seaweeds are not assumed to pass under/over one another as in classical braids. We will also assume that all crossings are between exactly two seaweeds, hence three or more seaweeds can never meet at a single point.

In a seaweed braid, a given pair of seaweeds may cross an arbitrary number of times. We call a seaweed braid *reduced*, if every pair of its seaweeds cross at most once (i.e. either once, or not at all).

Similarly to classical braids, two seaweed braids of the same width can be *multiplied*. The product braid is obtained as follows. First, we draw one braid above the other, identifying the bottom nodes of the top braid with the top nodes of the bottom

³A tongue-in-cheek justification for this term is that seaweed braids are like ordinary braids, except that they are sticky: a pair of seaweeds, once they have crossed, cannot be fully untangled.

braid. Then, we join up each pair of seaweeds that became incident in the previous step. Note that, even if both original seaweed braids were reduced, their product may in general not be reduced.

Example In Fig. 1b, the left-hand side is a product of two reduced seaweed braids. In this product braid, some pairs of seaweeds cross twice, hence it is not reduced.

Seaweed braids can be transformed (and, in particular, unreduced braids can be reduced) according to a specific set of algebraic rules. These rules are incorporated into the following formal definition.

Definition 9 The *seaweed monoid* \mathcal{T}_n is a finitely presented monoid on n generators: id (the identity element), g_1, g_2, \dots, g_{n-1} . The presentation of monoid \mathcal{T}_n consists of the *idempotence relations*

$$g_t^2 = g_t \quad t \in [1 : n - 1] \quad (1)$$

the *far commutativity relations*

$$g_t g_u = g_u g_t \quad t, u \in [1 : n - 1], u - t \geq 2 \quad (2)$$

and the *braid relations*

$$g_t g_u g_t = g_u g_t g_u \quad t, u \in [1 : n - 1], u - t = 1 \quad (3)$$

Traditionally, this structure is also known as the *0-Hecke monoid of the symmetric group* $H_0(\mathcal{S}_n)$, or the *Richardson–Springer monoid* (for details, see e.g. Denton et al. [20], Mazorchuk and Steinberg [49], Deng et al. [19]).

The correspondence between elements of the seaweed monoid and seaweed braids is as follows. The monoid multiplication (i.e. concatenation of words in the generators) corresponds to the multiplication of seaweed braids. The identity element id corresponds to a seaweed braid where the top nodes are connected to the bottom nodes in the left-to-right order, without any crossings. Each of the remaining generators g_t corresponds to an *elementary crossing*, i.e. to a seaweed braid where the only crossing is between a pair of neighbouring seaweeds in half-integer positions t^- and t^+ . Figure 2 shows the defining relations of the seaweed monoid (1)–(3) in terms of seaweed braids.

Example In Fig. 1b, the left-hand side is an unreduced product of two seaweed braids. We now *comb* the seaweeds by running through all their crossings, respecting the top-to-bottom partial order of the crossings. For each crossing, we check whether the two crossing seaweeds have previously crossed above the current point. If this is the case, then we undo the current crossing by removing it from the braid and replacing it by two non-crossing seaweed pieces. The correctness of this combing procedure is easy to prove by the seaweed monoid relations (1)–(3). After all the crossings have been combed, we obtain a reduced seaweed braid shown in the middle of Fig. 1b. Another equivalent reduced seaweed braid is shown in the right-hand side.

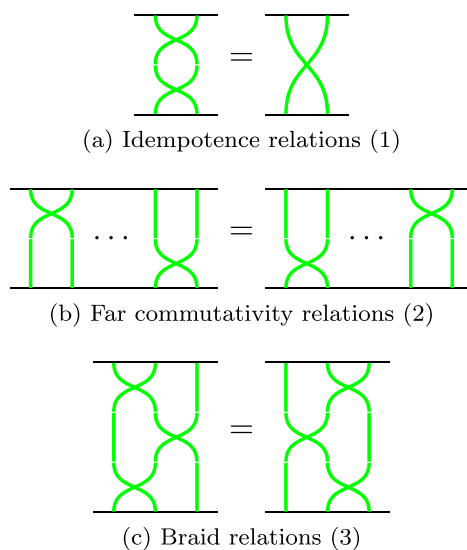


Fig. 2 Defining relations of the seaweed monoid

A permutation matrix P over $\langle 0 : n \mid 0 : n \rangle$ can be represented by a seaweed braid as follows. The row and column indices correspond respectively to the top and the bottom nodes, ordered from left to right. A nonzero $P(\hat{i}, \hat{j}) = 1$ corresponds to a seaweed connecting top node \hat{i} and bottom node \hat{j} . For a given permutation, it is always possible to draw the seaweeds so that the resulting seaweed braid is reduced. In general, this reduced braid will not be unique; however, it turns out that all the reduced braids corresponding to the same permutation are equivalent. We formalise this observation by the following lemma.

Lemma 1 *The seaweed monoid \mathcal{T}_n consists of at most $n!$ distinct elements.*

Proof It is straightforward to see that any seaweed braid can be transformed into a reduced one, using relations (1)–(3). Then, any two reduced seaweed braids corresponding to the same permutation can be transformed into one another, using far commutativity (2) and the braid relations (3). Therefore, each permutation corresponds to a single element of \mathcal{T}_n . This mapping is surjective, therefore the number of elements in \mathcal{T}_n is at most the total number of permutations $n!$. \square

We now establish a direct connection between elements of the seaweed monoid and permutation matrices. The identity generator id corresponds to the identity matrix Id . Each of the remaining generators g_t corresponds to an *elementary transposition matrix* G_t , defined as

$$G_t(\hat{i}, \hat{j}) = \begin{cases} 1 - Id(\hat{i}, \hat{j}) & \text{if } \hat{i}, \hat{j} \in \{t^-, t^+\} \\ Id(\hat{i}, \hat{j}) & \text{otherwise} \end{cases}$$

Lemma 2 *The set $\{G_t^\Sigma\}$, $t \in [1 : n - 1]$, generates the full distance multiplication monoid of simple unit-Monge matrices.*

Proof Let P be a permutation matrix. Consider an arbitrary reduced seaweed braid corresponding to P , and let t be the position of its first elementary crossing. Consider the truncated seaweed braid, obtained by removing this seaweed crossing. This braid is still reduced, and such that the pair of seaweeds originating in t^- , t^+ do not cross. Let this pair of seaweeds terminate at indices \hat{k}_0 , \hat{k}_1 , where $\hat{k}_0 < \hat{k}_1$. Let Q be the permutation matrix corresponding to the truncated seaweed braid. We have

$$\begin{aligned} P(t^-, \hat{k}_1) &= P(t^+, \hat{k}_0) = 1 \\ Q(t^-, \hat{k}_0) &= Q(t^+, \hat{k}_1) = 1 \end{aligned}$$

We will now show that $P = G_t \square Q$ or, equivalently $P^\Sigma = G_t^\Sigma \odot Q^\Sigma$. The lemma statement then follows by induction.

Note that $G_t^\Sigma(i, j) = Id^\Sigma(i, j)$ and $Q^\Sigma(i, j) = P^\Sigma(i, j)$ for all $i \in [0 : n]$, $i \neq t$, and for all $j \in [0 : n]$. Therefore, we have

$$(G_t^\Sigma \odot Q^\Sigma)(i, k) = (Id^\Sigma \odot Q^\Sigma)(i, k) = Q^\Sigma(i, k) = P^\Sigma(i, k)$$

for all $i \in [0 : n]$, $i \neq t$, and for all $k \in [0 : n]$.

It remains to consider the case $i = t$. Note that $G_t^\Sigma(t, j) = Id^\Sigma(t, j)$ for all $j \in [0 : n]$, $j \neq t$. Let $k \in [0 : n]$. We have

$$(G_t^\Sigma \odot Q^\Sigma)(t, k) = \min_{j \in [0 : n]} (G_t^\Sigma(t, j) + Q^\Sigma(j, k)) \quad (4)$$

By definition of the distribution matrix (Definition 1), we have

$$\begin{aligned} G_t^\Sigma(t, t - 1) &= 0 \\ G_t^\Sigma(t, t) &= G_t^\Sigma(t, t + 1) = 1 \\ 0 &\leq Q^\Sigma(t - 1, k) - Q^\Sigma(t, k) \leq 1 \\ 0 &\leq Q^\Sigma(t, k) - Q^\Sigma(t + 1, k) \leq 1 \end{aligned}$$

Hence, we have

$$\begin{aligned} G_t^\Sigma(t, t) + Q^\Sigma(t, k) &= 1 + Q^\Sigma(t, k) \\ &\geq 0 + Q^\Sigma(t - 1, k) = G_t^\Sigma(t, t - 1) + Q^\Sigma(t - 1, k) \end{aligned}$$

and, analogously,

$$G_t^\Sigma(t, t) + Q^\Sigma(t, k) \geq G_t^\Sigma(t, t + 1) + Q^\Sigma(t + 1, k)$$

We have established that the value under the minimum operator in (4) for $j = t$ is always no less than the values for both $j = t - 1$ and $j = t + 1$. Therefore, the

minimum is never attained solely at $j = t$, so we may assume $j \neq t$. We now consider two cases: either $j \in [0 : t - 1]$, or $j \in [t + 1 : n]$.

For $j \in [0 : t - 1]$, we have $G_t^\Sigma(t, j) = 0$. Therefore,

$$\begin{aligned} & \min_{j \in [0:t-1]} (G_t^\Sigma(t, j) + Q^\Sigma(j, k)) \\ &= \min_{j \in [0:t-1]} (0 + Q^\Sigma(j, k)) \quad (\text{attained at } j = t - 1) \\ &= Q^\Sigma(t - 1, k) = P^\Sigma(t - 1, k) \end{aligned}$$

Similarly, for $j \in [t + 1 : n]$, we have $G_t^\Sigma(t, j) = j - t$. Therefore,

$$\begin{aligned} & \min_{j \in [t+1:n]} (G_t^\Sigma(t, j) + Q^\Sigma(j, k)) \\ &= \min_{j \in [t+1:n]} (j - t + Q^\Sigma(j, k)) \quad (\text{attained at } j = t + 1) \\ &= 1 + Q^\Sigma(t + 1, k) = 1 + P^\Sigma(t + 1, k) \end{aligned}$$

Substituting into (4), we now have

$$(G_t^\Sigma \odot Q^\Sigma)(t, k) = \min(P^\Sigma(t - 1, k), 1 + P^\Sigma(t + 1, k))$$

Recall that $P(t^-, \hat{k}_1) = P(t^+, \hat{k}_0) = 1$. We have

$$\begin{aligned} P^\Sigma(t - 1, k) &= P^\Sigma(t, k) = P^\Sigma(t + 1, k) \quad \text{for } k < \hat{k}_0 \\ P^\Sigma(t - 1, k) &= P^\Sigma(t, k) = 1 + P^\Sigma(t + 1, k) \quad \text{for } \hat{k}_0 < k < \hat{k}_1 \\ P^\Sigma(t - 1, k) - 1 &= P^\Sigma(t, k) = 1 + P^\Sigma(t + 1, k) \quad \text{for } \hat{k}_1 < k \end{aligned}$$

In all three above cases, we have

$$\min(P^\Sigma(t - 1, k), 1 + P^\Sigma(t + 1, k)) = P^\Sigma(t, k)$$

which completes the proof. \square

We are now able to establish a formal connection between the unit-Monge monoid and the seaweed monoid.

Theorem 4 *The distance multiplication monoid of $n \times n$ simple unit-Monge matrices is isomorphic to the seaweed monoid \mathcal{T}_n .*

Proof We have already established a bijection between the generators of both monoids: a generator simple unit-Monge matrix G_t^Σ corresponds to a generator g_t of the seaweed monoid \mathcal{T}_n . It is straightforward to check that relations (1)–(3) are verified by matrices G_t^Σ , therefore the bijection on the generators defines a homomorphism from the seaweed monoid to the unit-Monge matrix monoid. By Lemma 2, this homomorphism is surjective, hence the cardinality of \mathcal{T}_n is at least the number

of all simple unit-Monge matrices of size n , equal to $n!$. However, by Lemma 1, the cardinality of \mathcal{T}_n is at most $n!$. Thus, the cardinality of \mathcal{T}_n is exactly $n!$, and the two monoids are isomorphic. \square

Example In Fig. 1, the seaweed braids shown in Fig. 1b correspond to the implicit matrix distance product $P_A \square P_B = P_C$ in Fig. 1a.

The seaweed monoid is closely related to some other well-known algebraic structures:

- by replacing the idempotence relations (1) with involution relations $g_t^2 = id$, we obtain the *Coxeter presentation* of the symmetric group;
- by removing the idempotence relations (1), and keeping far commutativity (2) and braid relations (3), we obtain the classical *positive braid monoid* (see e.g. [38, Sect. 6.5]);
- by removing the braid relations (3), and keeping idempotence (1) and far commutativity (2), we obtain the *locally free idempotent monoid* [67] (see also [24]);
- by introducing the generators' inverses g_t^{-1} , and replacing the idempotence relations (1) with cancellation relations $g_t g_t^{-1} = id$, we obtain the classical *braid group*.

A generalisation of the seaweed monoid is given by 0-Hecke monoids of general Coxeter groups, also known as *Coxeter monoids*. These monoids arise naturally as subgroup monoids in groups. The theory of Coxeter monoids can be traced back to Bourbaki [11], and was developed in [12, 26, 53, 66]. A further generalisation to \mathcal{J} -trivial monoids has been studied by Denton et al. [20]. The contents of this and the following sections can be regarded as the first step in the algorithmic study of such general classes of monoids.

3.3 Fast Implicit Distance Multiplication

For generic, explicitly presented matrices, direct application of Definition 7 gives an algorithm for matrix distance multiplication of size n , running in time $O(n^3)$. Slightly subcubic algorithms for this problem have also been obtained. The fastest currently known algorithm is by Chan [15], running in time $O(\frac{n^3(\log \log n)^3}{\log^2 n})$.

For Monge matrices, distance multiplication can easily be performed in time $O(n^2)$, using the standard row-minima searching technique of Aggarwal et al. [1] to perform matrix-vector multiplication in linear time (see also [6, 57]). Alternatively, an algorithm with the same quadratic running time can be obtained directly by the divide-and-conquer technique (see e.g. [5]).

While the quadratic running time is trivially optimal for explicit matrices, it is possible to break through this time barrier in the case of implicitly represented matrices. Subquadratic distance multiplication algorithms for implicit simple (sub)unit-Monge matrices were given in [58, 61], running in time $O(n^{1.5})$. We now show an even faster algorithm for this problem.

Theorem 5 Let P_A, P_B, P_C be $n \times n$ permutation matrices, such that $P_A \boxdot P_B = P_C$. Given the nonzeros of P_A, P_B , the nonzeros of P_C can be computed in time $O(n \log n)$.

Proof Let P_A, P_B, P_C be permutation matrices over $\langle 0:n \mid 0:n \rangle$. The algorithm follows a divide-and-conquer approach, in the form of recursion on n .

Recursion base: $n = 1$. The computation is trivial.

Recursive step: $n > 1$. Assume without loss of generality that n is even. Informally, the idea is to split the range of index j in the definition of matrix distance product (Definition 7) into two sub-intervals of size $\frac{n}{2}$. For each of these half-sized sub-intervals of j , we use the sparsity of the input permutation matrix P_A (respectively, P_B) to reduce the range of index i (respectively, k) to a (not necessarily contiguous) subset of size $\frac{n}{2}$; this completes the *divide phase*. We then call the algorithm recursively on the two resulting half-sized subproblems. Using the subproblem solutions, we reconstruct the output permutation matrix P_C ; this is the *conquer phase*.

We now describe each phase of the recursive step in more detail.

Divide phase. By Definition 8, we have

$$P_A^\Sigma \odot P_B^\Sigma = P_C^\Sigma$$

Consider the partitioning of matrices P_A, P_B into subpermutation matrices

$$P_A = \begin{bmatrix} P_{A,lo} & P_{A,hi} \end{bmatrix} \quad P_B = \begin{bmatrix} P_{B,lo} \\ P_{B,hi} \end{bmatrix}$$

where $P_{A,lo}, P_{A,hi}, P_{B,lo}, P_{B,hi}$ are over $\langle 0:n \mid 0:\frac{n}{2} \rangle, \langle 0:n \mid \frac{n}{2}:n \rangle, \langle 0:\frac{n}{2} \mid 0:n \rangle, \langle \frac{n}{2}:n \mid 0:n \rangle$, respectively; in each of these matrices, we maintain the indexing of the original matrices P_A, P_B . We now have two implicit matrix multiplication subproblems

$$P_{A,lo}^\Sigma \odot P_{B,lo}^\Sigma = P_{C,lo}^\Sigma \quad P_{A,hi}^\Sigma \odot P_{B,hi}^\Sigma = P_{C,hi}^\Sigma$$

where $P_{C,lo}, P_{C,hi}$ are of size $n \times n$. Each of the subpermutation matrices $P_{A,lo}, P_{A,hi}, P_{B,lo}, P_{B,hi}, P_{C,lo}, P_{C,hi}$ has exactly $\frac{n}{2}$ nonzeros.

Recall from the proof of Theorem 3 that a zero row in $P_{A,lo}$ (respectively, a zero column in $P_{B,lo}$) corresponds to a zero row (respectively, column) in their implicit distance product $P_{C,lo}$. Therefore, we can delete all zero rows and columns from $P_{A,lo}, P_{B,lo}, P_{C,lo}$, obtaining, after appropriate index remapping, three $\frac{n}{2} \times \frac{n}{2}$ permutation matrices. Consequently, the first subproblem can be solved by first performing a linear-time index remapping (corresponding to the deletion of zero rows and columns from $P_{A,lo}, P_{B,lo}$), then making a recursive call on the resulting half-sized problem, and then performing an inverse index remapping (corresponding to the reinsertion of the zero rows and columns into $P_{C,lo}$). The second subproblem can be solved analogously.

Conquer phase. We now need to combine the solutions for the two subproblems to a solution for the original problem. Note that we cannot simply put together the

nonzeros of the subproblem solutions. The original problem depends on the subproblems in a more subtle way: some elements of P_A^Σ depend on elements of both $P_{A,lo}$ and $P_{A,hi}$, and therefore would not be accounted for directly by the solution to either subproblem on its own. A similar observation holds for elements of P_B^Σ . However, note that the nonzeros in the two subproblems have disjoint index ranges, and therefore the direct combination of subproblem solutions $P_{C,lo} + P_{C,hi}$, although not a solution to the original problem, is still a permutation matrix.

In order to combine correctly the solutions of the two subproblems, let us consider the relationship between these subproblems in more detail. First, we split the range of index j in the definition of matrix distance product (Definition 7) into a “low” and a “high” sub-interval, each of size $\frac{n}{2}$.

$$\begin{aligned} P_C^\Sigma(i, k) &= \min_{j \in [0:n]} (P_A^\Sigma(i, j) + P_B^\Sigma(j, k)) \\ &= \min \left(\min_{j \in [0:\frac{n}{2}]} (P_A^\Sigma(i, j) + P_B^\Sigma(j, k)), \min_{j \in [\frac{n}{2}:n]} (P_A^\Sigma(i, j) + P_B^\Sigma(j, k)) \right) \end{aligned} \quad (5)$$

for all $i, k \in [0 : n]$. Let us denote the two arguments in (5) by $M_{lo}(i, k)$ and $M_{hi}(i, k)$, respectively:

$$P_C^\Sigma(i, k) = \min(M_{lo}(i, k), M_{hi}(i, k)) \quad (6)$$

for all $i, k \in [0 : n]$. The first argument in (5), (6) can be expressed via the solutions of the two subproblems as follows:

$$\begin{aligned} M_{lo}(i, k) &= \min_{j \in [0:\frac{n}{2}]} (P_A^\Sigma(i, j) + P_B^\Sigma(j, k)) \quad (\text{definition of } \Sigma) \\ &= \min_{j \in [0:\frac{n}{2}]} \left(P_{A,lo}^\Sigma(i, j) + P_{B,lo}^\Sigma(j, k) + P_{B,hi}^\Sigma\left(\frac{n}{2}, k\right) \right) \\ &\quad (\text{term rearrangement}) \\ &= \min_{j \in [0:\frac{n}{2}]} (P_{A,lo}^\Sigma(i, j) + P_{B,lo}^\Sigma(j, k)) + P_{B,hi}^\Sigma\left(\frac{n}{2}, k\right) \\ &\quad (\text{definition of } \odot) \\ &= P_{C,lo}^\Sigma(i, k) + P_{C,hi}^\Sigma(0, k) \end{aligned} \quad (7)$$

Here, the final equality is due to

$$\begin{aligned} P_{C,hi}^\Sigma(0, k) &= \min_{j \in [\frac{n}{2}:n]} (P_{A,hi}^\Sigma(0, j) + P_{B,hi}^\Sigma(j, k)) \\ &= \min_{j \in [\frac{n}{2}:n]} \left(j - \frac{n}{2} + P_{B,hi}^\Sigma(j, k) \right) = P_{B,hi}^\Sigma\left(\frac{n}{2}, k\right) \end{aligned}$$

since the minimum is attained at $j = \frac{n}{2}$. The second argument in (5), (6) can be expressed analogously as

$$\begin{aligned} M_{hi}(i, k) &= \min_{j \in [\frac{n}{2}, n]} (P_A^\Sigma(i, j) + P_B^\Sigma(j, k)) \\ &= P_{C,hi}^\Sigma(i, k) + P_{C,lo}^\Sigma(i, n) \end{aligned} \quad (8)$$

The minimisation operator in (5), (6) is equivalent to evaluating the sign of the difference of its two arguments:

$$\begin{aligned} \delta(i, k) &= M_{lo}(i, k) - M_{hi}(i, k) \quad (\text{by (7), (8)}) \\ &= (P_{C,lo}^\Sigma(i, k) + P_{C,hi}^\Sigma(0, k)) - (P_{C,hi}^\Sigma(i, k) + P_{C,lo}^\Sigma(i, n)) \\ &\quad (\text{term rearrangement}) \\ &= (P_{C,hi}^\Sigma(0, k) - P_{C,hi}^\Sigma(i, k)) - (P_{C,lo}^\Sigma(i, n) - P_{C,lo}^\Sigma(i, k)) \\ &\quad (\text{definition of } \Sigma) \\ &= \sum_{\hat{i} \in (0:i), \hat{k} \in (0:k)} P_{C,hi}(\hat{i}, \hat{k}) - \sum_{\hat{i} \in (i:n), \hat{k} \in (k:n)} P_{C,lo}(\hat{i}, \hat{k}) \\ &\quad (\text{definition of } \Sigma, R) \\ &= P_{C,hi}^{R\Sigma}(n-k, i) - P_{C,lo}^{RRR\Sigma}(k, n-i) \end{aligned}$$

Since $P_{C,lo}$, $P_{C,hi}$ are subpermutation matrices, and $P_{C,lo} + P_{C,hi}$ a permutation matrix, it follows that function δ is unit-monotone increasing in each of its arguments. The sign of function δ determines the positions of nonzeros in P_C as follows. Let us fix some half-integer point $\hat{i}, \hat{k} \in (0:n)$ in P_C , and consider the signs of the four values $\delta(\hat{i}^\pm, \hat{k}^\pm)$ at neighbouring integer points. Due to the unit-monotonicity of δ , only three cases are possible.

Case $\delta(\hat{i}^\pm, \hat{k}^\pm) \leq 0$ for all four sign combinations. We have

$$M_{lo}(\hat{i}^\pm, \hat{k}^\pm) \leq M_{hi}(\hat{i}^\pm, \hat{k}^\pm)$$

for each sign combination taken consistently on both sides of the inequality, and, by (6),

$$P_C^\Sigma(\hat{i}^\pm, \hat{k}^\pm) = M_{lo}(\hat{i}^\pm, \hat{k}^\pm)$$

Hence, we have

$$P_C(\hat{i}, \hat{k}) = P_C^{\Sigma\Box}(\hat{i}, \hat{k}) = M_{lo}^\Box(\hat{i}, \hat{k}) = P_{C,lo}(\hat{i}, \hat{k}) \quad (\text{definition of } \Sigma, \Box, (7), (8))$$

Thus, in this case $P_C(\hat{i}, \hat{k}) = 1$ is equivalent to $P_{C,lo}(\hat{i}, \hat{k}) = 1$. Note that this also implies $\delta(\hat{i}^-, \hat{k}^-) < 0$, since otherwise we would have $\delta(\hat{i}^\pm, \hat{k}^\pm) = 0$ for all four sign combinations, and hence, by symmetry, also $P_{C,hi}(\hat{i}, \hat{k}) = 1$. However, that would imply $P_{C,lo}(\hat{i}, \hat{k}) + P_{C,hi}(\hat{i}, \hat{k}) = 1 + 1 = 2$, which is a contradiction to $P_{C,lo} + P_{C,hi}$ being a permutation matrix.

Case $\delta(\hat{i}^\pm, \hat{k}^\pm) \geq 0$ for all four sign combinations. Symmetrically to the previous case, we have

$$P_C(\hat{i}, \hat{k}) = P_{C,hi}(\hat{i}, \hat{k})$$

Thus, in this case $P_C(\hat{i}, \hat{k}) = 1$ is equivalent to $P_{C,lo}(\hat{i}, \hat{k}) = 1$, and implies $\delta(\hat{i}^+, \hat{k}^+) > 0$.

Case $\delta(\hat{i}^-, \hat{k}^-) < 0$, $\delta(\hat{i}^-, \hat{k}^+) = \delta(\hat{i}^+, \hat{k}^-) = 0$, $\delta(\hat{i}^+, \hat{k}^+) > 0$. By (6), we have

$$P_C^\Sigma(\hat{i}^-, \hat{k}^-) = M_{lo}(\hat{i}^-, \hat{k}^-)$$

$$P_C^\Sigma(\hat{i}^+, \hat{k}^-) = M_{lo}(\hat{i}^+, \hat{k}^-)$$

$$P_C^\Sigma(\hat{i}^-, \hat{k}^+) = M_{lo}(\hat{i}^-, \hat{k}^+)$$

$$P_C^\Sigma(\hat{i}^+, \hat{k}^+) = M_{hi}(\hat{i}^+, \hat{k}^+) < M_{lo}(\hat{i}^+, \hat{k}^+)$$

Hence,

$$P_C(\hat{i}, \hat{k}) = P_C^{\Sigma\Box}(\hat{i}, \hat{k}) > M_{lo}^\Box(\hat{i}, \hat{k}) = P_{C,lo}(\hat{i}, \hat{k}) \quad (\text{definition of } \Sigma, \Box, (7), (8))$$

Since both P_C and $P_{C,lo}$ are zero-one matrices, the strict inequality implies that $P_C(\hat{i}, \hat{k}) = 1$ and $P_{C,lo}(\hat{i}, \hat{k}) = 0$. Symmetrically, also $P_{C,hi}(\hat{i}, \hat{k}) = 0$.

Summarising the above three cases, we have $P_C(\hat{i}, \hat{k}) = 1$, if and only if one of the following conditions holds:

$$\delta(\hat{i}^-, \hat{k}^-) < 0 \quad \text{and} \quad P_{C,lo}(\hat{i}, \hat{k}) = 1 \quad (9)$$

$$\delta(\hat{i}^+, \hat{k}^+) > 0 \quad \text{and} \quad P_{C,hi}(\hat{i}, \hat{k}) = 1 \quad (10)$$

$$\delta(\hat{i}^-, \hat{k}^-) < 0 \quad \text{and} \quad \delta(\hat{i}^+, \hat{k}^+) > 0 \quad (11)$$

By the discussion above, these three conditions are mutually exclusive.

In order to check the conditions (9)–(11), we need an efficient procedure for determining the sign of function δ in points of the integer square $[0 : n \mid 0 : n]$. Informally, low (respectively, high) values of both i and k correspond to negative (respectively, positive) values of $\delta(i, k)$. By unit-monotonicity of δ , there must exist a pair of monotone rectilinear paths from the bottom-left to the top-right corner of the half-integer square $\langle -1 : n + 1 \mid -1 : n + 1 \rangle$, that separate strictly negative and nonnegative (respectively, strictly positive and nonpositive) values of δ .

We now give a simple efficient procedure for finding such a pair of separating paths. By symmetry we only need to consider the *lower separating path*. For all integer points (i, k) above-left (respectively, below-right) of this path, we have $\delta(i, k) < 0$ (respectively, $\delta(i, k) \geq 0$).

We start at the bottom-left corner of the square, with $(\hat{i}, \hat{k}) = (n^+, 0^-)$ as the initial point on the lower separating path. We have $\delta(\hat{i}^-, \hat{k}^+) = \delta(n, 0) = 0$.

Let (\hat{i}, \hat{k}) now denote any current point on the lower separating path, and suppose that we have evaluated $\delta(\hat{i}^-, \hat{k}^+)$. The sign of this value determines the next point

on the path:

$$\begin{aligned} (\hat{i}, \hat{k} + 1) & \quad \text{if } \delta(\hat{i}^-, \hat{k}^+) < 0 \\ (\hat{i} - 1, \hat{k}) & \quad \text{if } \delta(\hat{i}^-, \hat{k}^+) \geq 0 \end{aligned}$$

Following this choice, we then evaluate either $\delta(\hat{i}^-, (\hat{k} + 1)^+)$, or $\delta((\hat{i} - 1)^-, \hat{k}^+)$ from $\delta(\hat{i}^-, \hat{k}^+)$ by an incremental query of Theorem 1 in time $O(1)$. The computation is now repeated with the new current point.

The described path-finding procedure runs until either $\hat{i} = 0^-$, or $\hat{k} = n^+$. We then complete the path by moving in a straight horizontal (respectively, vertical) line to the final destination $(\hat{i}, \hat{k}) = (0^-, n^+)$. The whole procedure of finding the lower separating path runs in time $O(n)$. A symmetric procedure with the same running time can be used to find the *upper separating path*, for which we have $\delta(i, k) \leq 0$ on the above-left, and $\delta(i, k) > 0$ on the below-right.

Given a value $d \in [-n + 1 : n - 1]$, let us now consider the set of points (\hat{i}, \hat{k}) with $\hat{k} - \hat{i} = d$; such a set forms a diagonal in the half-integer square. Let $(\hat{i}_{lo}, \hat{k}_{lo})$, where $\hat{k}_{lo} - \hat{i}_{lo} = d$, be the unique intersection point of the given diagonal with the lower separating path. Let $r_{lo}(d) = \hat{i}_{lo} + \hat{k}_{lo}$. Define $r_{hi}(d)$ analogously, using the upper separating path. Conditions (9)–(11) can now be expressed in terms of arrays r_{lo}, r_{hi} as follows:

$$\hat{i} + \hat{k} \leq r_{lo}(\hat{k} - \hat{i}) \quad \text{and} \quad P_{C,lo}(\hat{i}, \hat{k}) = 1 \quad (12)$$

$$\hat{i} + \hat{k} \geq r_{hi}(\hat{k} - \hat{i}) \quad \text{and} \quad P_{C,hi}(\hat{i}, \hat{k}) = 1 \quad (13)$$

$$\hat{i} + \hat{k} = r_{lo}(\hat{k} - \hat{i}) = r_{hi}(\hat{k} - \hat{i}) \quad (14)$$

Here, we make use of the fact that $\hat{i} + \hat{k} \leq r_{lo}(\hat{k} - \hat{i})$ is equivalent to $\hat{i}^- + \hat{k}^- < r_{lo}(\hat{k} - \hat{i})$, and $\hat{i} + \hat{k} \geq r_{hi}(\hat{k} - \hat{i})$ to $\hat{i}^+ + \hat{k}^+ > r_{hi}(\hat{k} - \hat{i})$.

The nonzeros of P_C satisfying either of the conditions (12), (13) can be found in time $O(n)$ by checking directly each of the nonzeros in matrices $P_{C,lo}$ and $P_{C,hi}$. The nonzeros of P_C satisfying condition (14) can be found in time $O(n)$ by a linear sweep of the points (\hat{i}, \hat{k}) on the two separating paths. We have now obtained all the nonzeros of matrix P_C .

(End of recursive step)

Time analysis. The recursion tree is a balanced binary tree of height $\log n$. In the root node, the computation runs in time $O(n)$. In each subsequent level, the number of nodes doubles, and the running time per node decreases by a factor of 2. Therefore, the overall running time is $O(n \log n)$. \square

Example Figure 3 illustrates the proof of Theorem 5 on a problem instance with a solution generated by the Wolfram Mathematica software. Figure 3a shows a pair of input 20×20 permutation matrices P_A, P_B , with nonzeros indicated by green circles. Figure 3b shows the partitioning of the implicit 20×20 matrix distance multiplication problem into two 10×10 subproblems. The nonzeros in the two subproblems are shown respectively by filled red squares and hollow blue squares. Figure 3c shows a recursive step. The lower and the upper separating paths are shown respectively in red

and in blue (note that the lower path is visually above the upper one; the lower/upper terminology refers to the relative values of δ , rather than the visual position of the paths). The nonzeros in the output matrix P_C satisfying (12), (13), (14) are shown respectively by filled red squares, hollow blue squares, and green circles; note that overall, there are 20 such nonzeros, and that they define a permutation matrix. Figure 3d shows the output matrix P_C .

Having proved Theorem 5, it is natural to ask whether a similar fast algorithm exists for implicit distance multiplication of general (not necessarily simple) unit-Monge matrices. Unfortunately, there does not appear to be an easy reduction from this more general problem to the case of simple matrices. A related problem is the distance multiplication of an implicit simple unit-Monge matrix by an arbitrary vector. Even this, more basic problem appears to be highly non-trivial. An optimal algorithm for this problem, running in time $O(n)$, was recently given by Gawrychowski [28].

3.4 Bruhat Order

Given a permutation, it is natural to ask how well-sorted it is. In particular, a permutation may be either fully sorted (the identity permutation), or fully anti-sorted (the reverse identity permutation), or anything in between. More generally, given two permutations, it is natural to ask whether, in some sense, one is “more sorted” than the other.

Let P_A, P_B be permutation matrices over $\langle 0:n \mid 0:n \rangle$. A classical “degree-of-sortedness” comparison is given by the following partial order (see e.g. Bóna [10], Hammett and Pittel [33], and references therein).

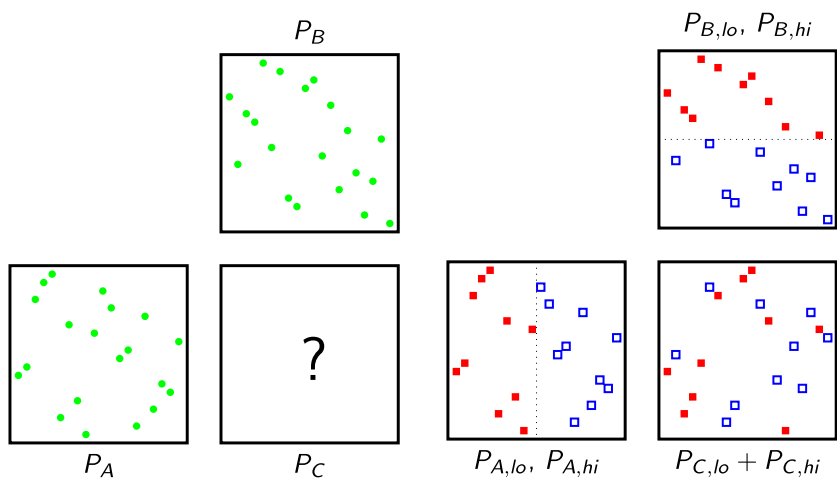
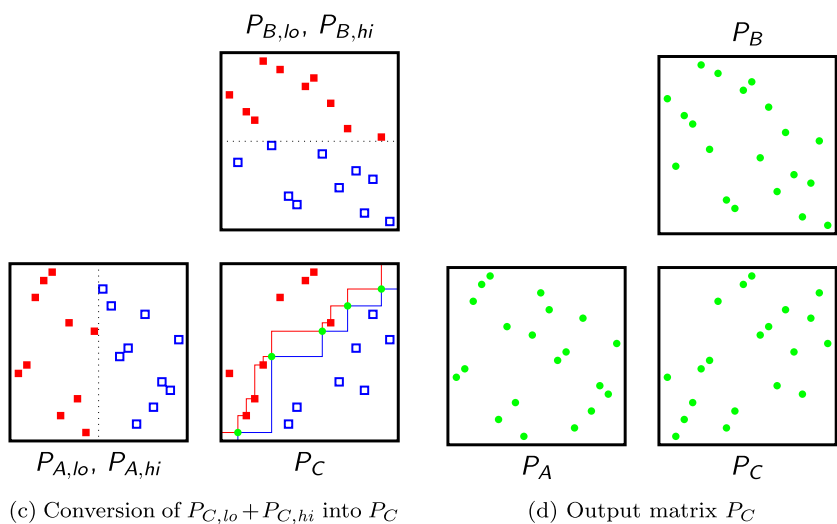
Definition 10 Matrix P_A is lower than matrix P_B in the *Bruhat order*, $P_A \leq P_B$, if P_A can be transformed to P_B by a sequence of *anti-sorting* steps. Each such step substitutes a (not necessarily contiguous) submatrix of the form $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ by a submatrix of the form $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$.

Informally, $P_A \leq P_B$, if P_A defines a “more sorted” permutation than P_B . More precisely, $P_A \leq P_B$, if the permutation defined by P_A can be transformed into the one defined by P_B by successive pairwise anti-sorting between arbitrary pairs of elements. Symmetrically, the permutation defined by P_B can be transformed into the one defined by P_A by successive pairwise sorting (or, equivalently, by an application of a comparison network; see e.g. Knuth [39]).

Bruhat order is an important group-theoretic concept, which can be generalised to arbitrary Coxeter groups (see Björner and Brenti [9], Denton et al. [20] for more details and further references).

Many equivalent definitions of the Bruhat order on permutations are known; see e.g. [9, 22, 31, 36, 71] and references therein. A classical combinatorial characterisation of the Bruhat order, known as *Ehresmann’s criterion* or *dot criterion*, is as follows.

Theorem 6 We have $P_A \leq P_B$, if and only if $P_A^\Sigma \leq P_B^\Sigma$ elementwise.

(a) Input matrices P_A, P_B (b) Subproblems $P_{A,lo} \sqcup P_{B,lo} = P_{C,lo}$ and $P_{A,hi} \sqcup P_{B,hi} = P_{C,hi}$ (c) Conversion of $P_{C,lo} + P_{C,hi}$ into P_C (d) Output matrix P_C **Fig. 3** Proof of Theorem 5: $P_A \sqcup P_B = P_C$

Proof Straightforward from the definitions; see e.g. [9]. □

Example We have

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}^{\sqcup} \sqcup \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 2 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}^{\sqcup} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Note that the permutation matrix on the right can be obtained from the one on the left by anti-sorting the 2×2 submatrix at the intersection of the top two rows with the leftmost and rightmost columns.

We also have

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}^{\square} \quad \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}^{\square}$$

The above two permutation matrices are incomparable in the Bruhat order.

Theorem 6 immediately gives one an algorithm for deciding whether two permutations are Bruhat-comparable in time $O(n^2)$. To the author's knowledge, no asymptotically faster algorithm for deciding Bruhat comparability has been known so far.

To demonstrate an application of our techniques, we now give a new characterisation of the Bruhat order in terms of the unit-Monge monoid (or, equivalently, the seaweed monoid). This characterisation will give us a substantially faster algorithm for deciding Bruhat comparability.

Intuitively, the connection between the Bruhat order and seaweeds is as follows. Consider matrix P_A and the rotated matrix P_A^R . The matrix rotation induces a one-to-one correspondence between the nonzeros of P_A^R and P_A , and therefore also between individual seaweeds in their reduced seaweed braids. A pair of seaweeds cross in a reduced braid of P_A^R , if and only if the corresponding pair of seaweeds do not cross in a reduced braid of P_A . Now consider the product braid $P_A^R \square P_A$, where each seaweed is made up of two mutually corresponding seaweeds from P_A^R and P_A , respectively. Every pair of seaweeds in braid $P_A^R \square P_A$ either cross in the top sub-braid P_A^R , or in the bottom sub-braid P_A , but not in both. Therefore, the product braid is a reduced seaweed braid, in which every pair of seaweeds cross exactly once. Thus, we have $P_A^R \square P_A = Id^R$.

Now suppose $P_A \leq P_B$. By Theorem 6, we have $P_A^\Sigma \leq P_B^\Sigma$ elementwise. Therefore, by Definition 7, $P_A^{R\Sigma} \odot P_A^\Sigma \leq P_A^{R\Sigma} \odot P_B^\Sigma$ elementwise, hence by Theorem 6, we have $P_A^R \square P_A \leq P_A^R \square P_B$. However, as argued above, $P_A^R \square P_A = Id^R$, which is the highest possible permutation matrix in the Bruhat order, corresponding to the reverse identity permutation. Therefore, $P_A^R \square P_B = Id^R$. We thus have a necessary condition for $P_A \leq P_B$. It turns out that this condition is also sufficient, giving us a new, computationally efficient criterion for Bruhat comparability.

Theorem 7 *We have $P_A \leq P_B$, if and only if $P_A^R \square P_B = Id^R$.*

Proof Let $i, j \in [0 : n]$. We have

$$\begin{aligned} & P_A^{R\Sigma}(i, j) + P_B^\Sigma(j, n-i) \quad (\text{definition of } R) \\ &= (n-i - P_A^\Sigma(j, n-i)) + P_B^\Sigma(j, n-i) \quad (\text{term rearrangement}) \\ &= (P_B^\Sigma(j, n-i) - P_A^\Sigma(j, n-i)) + n-i \end{aligned} \tag{15}$$

We now prove the implication separately in each direction.

Necessity. Let $P_A \preceq P_B$. By (15) and Theorem 6, we have

$$P_A^{R\Sigma}(i, j) + P_B^\Sigma(j, n-i) \geq n-i$$

This lower bound is attained at $j = 0$ (and, symmetrically, $j = n$): we have $P_A^{R\Sigma}(i, 0) + P_B^\Sigma(0, n-i) = 0 + (n-i) = n-i$. Therefore,

$$\begin{aligned} & (P_A^R \sqcap P_B)^\Sigma(i, n-i) \quad (\text{definition of } \sqcap) \\ &= \min_j (P_A^{R\Sigma}(i, j) + P_B^\Sigma(j, n-i)) = n-i \quad (\text{attained at } j=0) \end{aligned}$$

It is now easy to prove (e.g. by induction on n) that $P_A^R \sqcap P_B = Id^R$ is the only permutation matrix satisfying the above equation for all i . Sufficiency. Let $P_A^R \sqcap P_B = Id^R$. By Definition 7, we have

$$\min_j (P_A^{R\Sigma}(i, j) + P_B^\Sigma(j, n-i)) = Id^{R\Sigma}(i, n-i) = n-i$$

for all i . Therefore, for all i, j , $P_A^{R\Sigma}(i, j) + P_B^\Sigma(j, n-i) \geq n-i$. By (15), this is equivalent to $P_B^\Sigma(j, n-i) - P_A^\Sigma(j, n-i) \geq 0$, therefore $P_A^\Sigma(j, n-i) \leq P_B^\Sigma(j, n-i)$, hence by Theorem 6, we have $P_A \preceq P_B$. \square

The combination of Theorems 5 and 7 gives us an algorithm for deciding Bruhat comparability of permutations in time $O(n \log n)$.

4 Applications in String Comparison

The longest common subsequence (LCS) problem is a classical problem in computer science. Given two strings a, b of lengths m, n respectively, the LCS problem asks for the length of the longest string that is a subsequence of both a and b . This length is called the strings' LCS score. The classical dynamic programming algorithm for the LCS problem [51, 68] runs in time $O(mn)$; the best known algorithms improve on this running time by some (model-dependent) polylogarithmic factors [8, 17, 47, 70]. We refer the reader to monographs [18, 32] for the background on this problem and further references.

The semi-local LCS problem is a generalisation of the LCS problem, arising naturally in the context of string comparison. Given two strings a, b as before, the semi-local LCS problem asks for the LCS score of each string against all substrings of the other string, and of all prefixes of each string against all suffixes of the other string.

In [60, 61], we introduced the semi-local LCS problem, described its connections with unit-Monge matrices and seaweed braids, and gave a number of its algorithmic applications. Many of these applications use distance multiplication of simple unit-Monge matrices as a subroutine. In these cases, we can immediately obtain improved algorithms by plugging in the new multiplication algorithm given by Theorem 5. The rest of this section gives a few examples of such improvements. For further applications of our method, we refer the reader to [57].

4.1 Semi-Local LCS Between Permutations

An important special case of string comparison is where each of the input strings a , b is a *permutation string*, i.e. a string that consists of all distinct characters. Without loss of generality, we may assume that $m = n$, and that both strings are permutations of a given totally ordered alphabet of size n . The semi-local LCS problem on permutations is equivalent to finding the length of the longest increasing subsequence (LIS) in every substring of a given permutation string.

In [61], we gave an algorithm for the semi-local LCS problem on permutation strings, running in time $O(n^{1.5})$. By using the algorithm of Theorem 5, the running time improves to $O(n \log^2 n)$.

4.2 Cyclic LCS Between Permutations

The cyclic LCS problem on permutation strings is equivalent to the LIS problem on a circular string. This problem has been considered by Albert et al. [2], who gave a Monte Carlo randomised algorithm, running in time $O(n^{1.5} \log n)$ with small error probability.

In [61], we gave an algorithm for the cyclic LCS problem on permutation strings, running in deterministic time $O(n^{1.5})$. By using the algorithm of Theorem 5, the running time improves to $O(n \log^2 n)$.

4.3 Maximum Clique in a Circle Graph

A *circle graph* [25, 29] is defined as the intersection graph of a set of chords in a circle, i.e. the graph where each node represents a chord, and two nodes are adjacent, whenever the corresponding chords intersect. We consider the maximum clique problem on a circle graph.

The *interval model* of a circle graph is obtained by cutting the circle at an arbitrary point and laying it out on a line, so that the chords become (closed) intervals. The original circle graph is isomorphic to the overlap graph of its interval model, i.e. the graph where each node represents an interval, and two nodes are adjacent, whenever the corresponding intervals intersect but do not contain one another.

Figure 4 shows an instance of the maximum clique problem on a six-node circle graph. Figure 4a shows the set of chords defining a circle graph, with one of the maximum cliques highlighted in bold red. The cut point is shown by scissors. Figure 4b shows the corresponding interval model; the dotted diagonal line contains the intervals, each defined by the diagonal of a square. The squares corresponding to the maximum clique are highlighted in bold red.

It has long been known that the maximum clique problem in a circle graph on n nodes is solvable in polynomial time [27]. A number of algorithms have been proposed for this problem [4, 35, 48, 55]; the problem has also been studied in the context of line arrangements in the hyperbolic plane [23, 37]. Given an interval model of a circle graph, the running time of the above algorithms is $O(n^2)$ in the worst case, i.e. when the input graph is dense.

In [59, 61], we gave an algorithm running in time $O(n^{1.5})$. By using the algorithm of Theorem 5, the running time improves to $O(n \log^2 n)$.

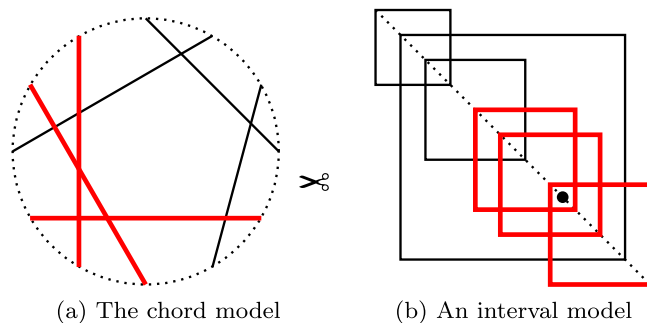


Fig. 4 A circle graph and its maximum clique

4.4 Compressed String Comparison

String compression is a classical paradigm, touching on many different areas of computer science. From an algorithmic viewpoint, it is natural to ask whether compressed strings can be processed efficiently without decompression. Early examples of such algorithms were given e.g. by Amir et al. [3] and by Rytter [56].

We consider the following general model of compression. Let t be a string of length n (typically large). String t will be called a *grammar-compressed string* (*GC-string*), if it is generated by a context-free grammar, also called a *straight-line program* (*SLP*). An SLP of length \bar{n} , $\bar{n} \leq n$, is a sequence of \bar{n} *statements*. A statement numbered k , $1 \leq k \leq \bar{n}$, has one of the following forms:

$$t_k = \alpha \quad \text{where } \alpha \text{ is an alphabet character}$$

$$t_k = t_i t_j \quad \text{where } 1 \leq i, j < k$$

We identify every symbol t_r with the string it expands to; in particular, we have $t = t_{\bar{n}}$. In general, the plain string length n can be exponential in the GC-string length \bar{n} .

As a special case, grammar compression includes the classical LZ78 and LZW compression schemes by Ziv, Lempel and Welch [69, 73]. It should also be noted that certain other compression methods, such as e.g. LZ77 [72] and run-length compression, do not fit directly into the grammar compression model.

The LCS problem on two GC-strings has been considered by Lifshits and Lohrey [45, 46], and proven to be PP-hard (and therefore NP-hard).

We consider the LCS problem between a plain (uncompressed) pattern string p of length m , and a grammar-compressed text string t of length n , generated by an SLP of length \bar{n} . In the special case of LZ77 or LZW compression of the text, the algorithm of Crochemore et al. [17] solves the LCS problem in time $O(m\bar{n})$. Thus, LZ77 or LZW compression of one of the input strings only slows down the LCS computation by a constant factor relative to the classical dynamic programming LCS algorithm, or by a polylogarithmic factor relative to the best known LCS algorithms.

The general case of an arbitrary GC-text appears more difficult. A GC-text is a special case of a context-free language, which consists of a single string. Therefore, the LCS problem between a GC-text and a plain pattern can be regarded as a special

case of the edit distance problem between a context-free language given by a grammar of size \bar{n} , and a pattern string of size m . For this more general problem, Myers [50] gave an algorithm running in time $O(m^3 \bar{n} + m^2 \cdot \bar{n} \log \bar{n})$. In [62], we gave an algorithm for another generalisation of LCS problem between a GC-text and a plain pattern, running in time $O(m^{1.5} \bar{n})$. Lifshits [44] asked whether the LCS problem in the same setting can be solved in time $O(m \bar{n})$. Our new method gives an algorithm for the LCS problem between a GC-string and an uncompressed string running in time $O(m \log m \cdot \bar{n})$, thus approaching an answer to Lifshits' question within a logarithmic factor.

Hermelin et al. [34] and Gawrychowski [28] refined the application of our techniques as follows. They consider the rational-weighted alignment problem (equivalently, the LCS or Levenshtein distance problems) on a pair of GC-strings a, b of total compressed length $\bar{r} = \bar{m} + \bar{n}$, parameterised by the strings' total plain length $r = m + n$. The algorithm of [34] runs in time $O(r \log(r/\bar{r}) \cdot \bar{r})$, which is improved in [28] to $O(r \log^{1/2}(r/\bar{r}) \cdot \bar{r})$. In both cases, our algorithm of Theorem 5 is used as a subroutine.

5 Conclusion

In this work, we have given a fast algorithm for distance multiplication of simple unit-Monge matrices, running in time $O(n \log n)$. The only known lower bound is trivial $\Omega(n)$. Therefore, Landau's question whether the problem can be solved in time $O(n)$ is still open, although we have now approached an answer within a logarithmic factor.

Our approach unifies and gives improved solutions to a number of algorithmic problems. It is likely that the scope of the applications can be widened still further, e.g. by considering new kinds of approximate matching and approximate repeat problems in strings.

The algebraic structure underlying our method is the distance-multiplication monoid of simple unit-Monge matrices, which we call the seaweed monoid. Traditionally, this structure is known as the 0-Hecke monoid of the symmetric group; it is a special case of a Coxeter monoid. Therefore, one may expect implications of our work in computational (semi)group theory. There are also potential connections with tropical mathematics, e.g. the tropical rank theory (see e.g. [21]).

In [40], we used distance multiplication of simple unit-Monge matrices to obtain the first parallel LCS algorithm with scalable communication. It would be interesting to see whether the fast distance multiplication algorithm given in the current work can be efficiently parallelised, and whether this can be used to achieve further improvement in the communication efficiency of parallel LCS computation.

Acknowledgement This work was conceived in a discussion with Gad Landau in Haifa. The imaginative term “seaweeds” was coined by Yuri Matiyasevich during a presentation by the author in St. Petersburg. I thank Elzbieta Babij, Philip Bille, Paweł Gawrychowski, Tim Griffin, Dima Grigoriev, Peter Krusche, Gad Landau, Victor Levandovsky, Sergei Nechaev, Luís Russo, Andrei Sobolevski, Nikolai Vavilov, Oren Weimann, and Michal Ziv-Ukelson for fruitful discussions, and many anonymous referees for their comments that helped to improve this work.

Open Access This article is distributed under the terms of the Creative Commons Attribution License which permits any use, distribution, and reproduction in any medium, provided the original author(s) and the source are credited.

References

1. Aggarwal, A., Klawe, M.M., Moran, S., Shor, P., Wilber, R.: Geometric applications of a matrix-searching algorithm. *Algorithmica* **2**(1), 195–208 (1987)
2. Albert, M.H., Atkinson, M.D., Nussbaum, D., Sack, J.-R., Santoro, N.: On the longest increasing subsequence of a circular list. *Inf. Process. Lett.* **101**, 55–59 (2007)
3. Amir, A., Benson, G., Farach, M.: Let sleeping files lie: Pattern matching in Z-compressed files. *J. Comput. Syst. Sci.* **52**(2), 299–307 (1996)
4. Apostolico, A., Atallah, M.J., Hambrusch, S.E.: New clique and independent set algorithms for circle graphs. *Discrete Appl. Math.* **36**, 1–24 (1992)
5. Apostolico, A., Atallah, M.J., Larmore, L.L., MacFaddin, S.: Efficient parallel algorithms for string editing and related problems. *SIAM J. Comput.* **19**(5), 968–988 (1990)
6. Atallah, M.J., Kosaraju, S.R., Larmore, L.L., Miller, G.L., Teng, S.-H.: Constructing trees in parallel. In: *Proceedings of the 1st ACM SPAA*, pp. 421–431 (1989)
7. Baxter, L., Jironkin, A., Hickman, R., Moore, J., Barrington, C., Krusche, P., Dyer, N.P., Buchanan-Wollaston, V., Tiskin, A., Beynon, J., Denby, K., Ott, S.: Conserved noncoding sequences highlight shared components of regulatory networks in dicotyledonous plants. *Plant Cell* **24**(10), 3949–3965 (2012)
8. Bille, P., Farach-Colton, M.: Fast and compact regular expression matching. *Theor. Comput. Sci.* **409**(3), 486–496 (2008)
9. Björner, A., Brenti, F.: *Combinatorics of Coxeter Groups*. Graduate Texts in Mathematics, vol. 231. Springer, Berlin (2005)
10. Bóna, M.: *Combinatorics of Permutations*. Discrete Mathematics and Its Applications. Chapman & Hall, London (2004)
11. Bourbaki, N.: *Groupes et Algèbres de Lie*. Chapitres 4, 5 et 6. Hermann, Paris (1968)
12. Buch, A.S., Kresch, A., Shimozone, M., Tamvakis, H., Yong, A.: Stable Grothendieck polynomials and K-theoretic factor sequences. *Math. Ann.* **340**(2), 359–382 (2008)
13. Burkard, R.E., Klinz, B., Rudolf, R.: Perspectives of Monge properties in optimization. *Discrete Appl. Math.* **70**(2), 95–161 (1996)
14. Butković, P.: *Max-Linear Systems: Theory and Algorithms*. Springer Monographs in Mathematics. Springer, Berlin (2010)
15. Chan, T.M.: More algorithms for all-pairs shortest paths in weighted graphs. In: *Proceedings of the 39th ACM STOC*, pp. 590–598 (2007)
16. Comet, J.-P.: Application of max-plus algebra to biological sequence comparisons. *Theor. Comput. Sci.* **293**(1), 189–217 (2003)
17. Crochemore, M., Landau, G.M., Ziv-Ukelson, M.: A subquadratic sequence alignment algorithm for unrestricted score matrices. *SIAM J. Comput.* **32**(6), 1654–1673 (2003)
18. Crochemore, M., Rytter, W.: *Text Algorithms*. Oxford University Press, Oxford (1994)
19. Deng, B., Du, J., Parshall, B., Wang, J.: *Finite Dimensional Algebras and Quantum Groups*. Mathematical Surveys and Monographs, vol. Number 150. Am. Math. Soc., Providence (2008)
20. Denton, T., Hivert, F., Schilling, A., Thiéry, N.M.: On the representation theory of finite J -trivial monoids. *Sémin. Lothar. Comb.* **64**(B64d) (2011)
21. Develin, M., Santos, F., Sturmfels, B.: On the rank of a tropical matrix. In: *Combinatorial and Computational Geometry*. MSRI Publications, vol. 52, pp. 213–242. Cambridge University Press, Cambridge (2005)
22. Drake, B., Gerrish, S., Skandera, M.: Two new criteria for comparison in the Bruhat order. *Electron. J. Comb.* **11**(1) (2004)
23. Dress, A., Koolena, J.H., Moulton, V.: On line arrangements in the hyperbolic plane. *Eur. J. Comb.* **23**(5), 549–557 (2002)
24. Esyp, E.S., Kazachkov, I.V., Remeslennikov, V.N.: Divisibility theory and complexity of algorithms in free partially commutative groups. In: *Groups, Languages, Algorithms*. Contemporary Mathematics, vol. 378, pp. 319–348. Am. Math. Soc., Providence (2005)

25. Even, S., Itai, A.: Queues, stacks and graphs. In: *Theory of Machines and Computations*, pp. 71–86. Academic Press, San Diego (1971)
26. Fomin, S., Greene, C.: Noncommutative Schur functions and their applications. *Discrete Math.* **193**(1–3), 179–200 (1998). Reprinted in *Discrete Math.* **306**(10–11), 1080–1096 (2006)
27. Gavril, F.: Algorithms for a maximum clique and a maximum independent set of a circle graph. *Networks* **3**, 261–273 (1973)
28. Gawrychowski, P.: Faster algorithm for computing the edit distance between SLP-compressed strings. In: *Proceedings of SPIRE. Lecture Notes in Computer Science*, vol. 7608, pp. 229–236 (2012)
29. Golumbic, M.C.: *Algorithmic Graph Theory and Perfect Graphs*. Number 57 in *Annals of Discrete Mathematics*, 2nd edn. Elsevier, Amsterdam (2004)
30. Gondran, M., Minoux, M.: *Graphs, Dioids and Semirings. Operations Research/Computer Science Interfaces Series*, vol. 47. Springer, Berlin (2008)
31. Grigor'ev, D.Yu.: Additive complexity in directed computations. *Theor. Comput. Sci.* **19**, 39–67 (1982)
32. Gusfield, D.: *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, Cambridge (1997)
33. Hammett, A., Pittel, B.: How often are two permutations comparable? *Trans. Am. Math. Soc.* **360**(9), 4541–4568 (2008)
34. Hermelin, D., Landau, G.M., Landau, S., Weimann, O.: Unified compression-based acceleration of edit-distance computation. *Algorithmica* **65**, 339–353 (2013)
35. Hsu, W.-L.: Maximum weight clique algorithms for circular-arc graphs and circle graphs. *SIAM J. Comput.* **14**(1), 224–231 (1985)
36. Johnson, C.R., Nasserar, S.: $TP_2 = Bruhat$. *Discrete Math.* **310**(10–11), 1627–1628 (2010)
37. Karzanov, A.: Combinatorial methods to solve cut-determined multi-flow problems. In: *Combinatorial Methods for Flow Problems*, vol. 3, pp. 6–69 (1979). VNIISI, in Russian
38. Kassel, C., Turaev, V.: *Braid Groups. Graduate Texts in Mathematics*, vol. 247. Springer, Berlin (2008)
39. Knuth, D.E.: *The Art of Computer Programming: Sorting and Searching* vol. 3. Addison-Wesley, Reading (1998)
40. Krusche, P., Tiskin, A.: Efficient parallel string comparison. In: *Proceedings of ParCo. NIC Series (John von Neumann Institute for Computing)*, vol. 38, pp. 193–200 (2007)
41. Krusche, P., Tiskin, A.: Longest increasing subsequences in scalable time and memory. In: *Proceedings of PPAM 2009, Revised Selected Papers, Part I. Lecture Notes in Computer Science*, vol. 6067, pp. 176–185 (2010)
42. Krusche, P., Tiskin, A.: New algorithms for efficient parallel string comparison. In: *Proceedings of ACM SPAA*, pp. 209–216 (2010)
43. Landau, G.: Can DIST tables be merged in linear time? An open problem. In: *Proceedings of the Prague Stringology Conference*, p. 1 (2006). Czech Technical University in Prague
44. Lifshits, Y.: Processing compressed texts: a tractability border. In: *Proceedings of CPM. Lecture Notes in Computer Science*, vol. 4580, pp. 228–240 (2007)
45. Lifshits, Y., Lohrey, M.: Querying and embedding compressed texts. In: *Proceedings of MFCS. Lecture Notes in Computer Science*, vol. 4162, pp. 681–692 (2006)
46. Lohrey, M.: Algorithmics on SLP-compressed strings: a survey. *Groups Complex. Cryptol.* **4**(2), 241–299 (2012)
47. Masek, W.J., Paterson, M.S.: A faster algorithm computing string edit distances. *J. Comput. Syst. Sci.* **20**, 18–31 (1980)
48. Masuda, S., Nakajima, K., Kashiwabara, T., Fujisawa, T.: Efficient algorithms for finding maximum cliques of an overlap graph. *Networks* **20**, 157–171 (1990)
49. Mazorchuk, V., Steinberg, B.: Double Catalan monoids. *J. Algebr. Comb.* **36**(3), 333–354 (2012)
50. Myers, G.: Approximately matching context-free languages. *Inf. Process. Lett.* **54**, 85–92 (1995)
51. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.* **48**(3), 443–453 (1970)
52. Picot, E., Krusche, P., Tiskin, A., Carré, I., Ott, S.: Evolutionary analysis of regulatory sequences (EARS) in plants. *Plant J.* **164**(1), 165–176 (2010)
53. Richardson, R.W., Springer, T.A.: The Bruhat order on symmetric varieties. *Geom. Dedic.* **35**(1–3), 389–436 (1990)
54. Rote, G.: Path problems in graphs. *Computing, Suppl.* **7**, 155–189 (1990)
55. Rotem, D., Urrutia, J.: Finding maximum cliques in circle graphs. *Networks* **11**, 269–278 (1981)

56. Rytter, W.: Algorithms on compressed strings and arrays. In: Proceedings of SOFSEM. Lecture Notes in Computer Science, vol. 1725, pp. 48–65 (1999)
57. Tiskin, A.: Semi-local string comparison: Algorithmic techniques and applications. Technical report. [arXiv:0707.3619](https://arxiv.org/abs/0707.3619)
58. Tiskin, A.: All semi-local longest common subsequences in subquadratic time. In: Proceedings of CSR. Lecture Notes in Computer Science, vol. 3967, pp. 352–363 (2006)
59. Tiskin, A.: Longest common subsequences in permutations and maximum cliques in circle graphs. In: Proceedings of CPM. Lecture Notes in Computer Science, vol. 4009, pp. 271–282 (2006)
60. Tiskin, A.: Semi-local longest common subsequences in subquadratic time. *J. Discrete Algorithms* **6**(4), 570–581 (2008)
61. Tiskin, A.: Semi-local string comparison: Algorithmic techniques and applications. *Math. Comput. Sci.* **1**(4), 571–603 (2008)
62. Tiskin, A.: Faster subsequence recognition in compressed strings. *J. Math. Sci.* **158**(5), 759–769 (2009)
63. Tiskin, A.: Periodic string comparison. In: Proceedings of CPM. Lecture Notes in Computer Science, vol. 5577, pp. 193–206 (2009)
64. Tiskin, A.: Fast distance multiplication of unit-Monge matrices. In: Proceedings of ACM–SIAM SODA, pp. 1287–1296 (2010)
65. Tiskin, A.: Towards approximate matching in compressed strings: Local subsequence recognition. In: Proceedings of CSR. Lecture Notes in Computer Science, vol. 6651, pp. 401–414 (2011)
66. Tsaranov, S.: Representation and classification of Coxeter monoids. *Eur. J. Comb.* **11**(2), 189–204 (1990)
67. Vershik, A.M., Nechaev, S., Bikbov, R.: Statistical properties of locally free groups with applications to braid groups and growth of random heaps. *Commun. Math. Phys.* **212**(2), 469–501 (2000)
68. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. *J. ACM* **21**(1), 168–173 (1974)
69. Welch, T.A.: A technique for high-performance data compression. *Computer* **17**(6), 8–19 (1984)
70. Wu, S., Manber, U., Myers, G.: A subquadratic algorithm for approximate limited expression matching. *Algorithmica* **15**, 50–67 (1996)
71. Zhao, Y.: On the Bruhat order of the symmetric group and its shellability. Technical report (2007)
72. Ziv, G., Lempel, A.: A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory* **23**, 337–343 (1977)
73. Ziv, G., Lempel, A.: Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theory* **24**, 530–536 (1978)