Theory of Computing Systems (2026) 70:2
https://doi.org/10.1007/s00224-025-10244-9

Generalization of Repetitiveness Measures for Two-Dimensional Strings

Lorenzo Carfagna[1] · Giovanni Manzini[1] · Giuseppe Romana[2] · Marinella Sciortino[2] · Cristian Urbina[3,4]

Received: 17 March 2025 / Accepted: 1 September 2025

Abstract
The problem of detecting and measuring the repetitiveness of one-dimensional strings has been extensively studied in data compression and text indexing. Our understanding of these issues has been significantly improved by the introduction of the notion of *string attractor* (Kempa and Prezza 2018) and by the results showing the relationship between attractors and other measures of compressibility. When the input data are structured in a non-linear way, as in two-dimensional strings, inherent redundancy often offers an even richer source for compression. However, systematic studies on repetitiveness measures for two-dimensional strings are still scarce. In this paper, we extend to two or more dimensions the main measures of complexity introduced for one-dimensional strings. We distinguish between the measures δ and γ, defined in terms of the substrings of the input, and the measures g, g_{rl}, and b, which are based on copy-paste mechanisms. We study the properties and mutual relationships between these two classes and we show that the two classes become incomparable for d-dimensional inputs as soon as $d \geqslant 2$. Moreover, we show that our grammar-based representation of a d-dimensional string of size N enables direct access to any symbol in $O(\log N)$ time. We also compare our measures for two-dimensional strings with the 2D Block Tree data structure (Brisaboa et al., Comput. J. **67**(1), 391–406, 2024) and provide some insights for the design of future effective two-dimensional compressors. A preliminary version of this paper appeared in the proceedings of the conference SPIRE 2024.

Keywords Two–dimensional strings · Repetitiveness measures · Grammar compression · Text compression

1 Introduction

In the latest decades, the amount of data generated in the world has become massive but it has been observed that, in many fields, most of this data is highly repetitive. For

Extended author information available on the last page of the article

the study of highly repetitive one-dimensional data, an important role is played by the notions of *substring complexity*, *string attractor*, *bidirectional macro scheme*, and *grammar compression*, which lead to the definition of the repetitiveness measures δ, γ, b, g, and g_{rl} (see [1, 2] for the definitions and their remarkable properties). Such measures provide the theoretical basis for the design and analysis of data compressors and compressed indexing data structures for highly repetitive data [3, 4].

Two-dimensional data, ranging from images to matrices, often contains inherent redundancy, wherein identical or similar substructures recur throughout the dataset. This great source of redundancy can be exploited for compression. Recently, Brisaboa et al. introduced the conceptually simple 2D Block Trees data structure to compress two-dimensional strings supporting the efficient access to the individual symbols [5]. Experiments have shown the practicality of 2D Block Trees for storing raster images and the adjacency matrix of Web graphs [6]. On the theoretical side, in [7] the authors proposed two generalizations of the measures δ and γ for square 2D input strings. Such generalized measures are based on properties of the *square* submatrices of the input string. The choice of considering only square submatrices was dictated by purely practical considerations: 2D square submatrices can be efficiently handled using the 2D Suffix Tree data structure [8, 9], and the 2D Block Tree is based on repeated occurrences of square submatrices in the input string. Indeed, [7, 10] provide the first theoretical analysis of the space usage of the 2D Block Tree and an optimal linear time construction algorithm.

In this paper, we generalize the 1D measures mentioned above (δ, γ, b, g, and g_{rl}) to 2D strings, considering submatrices of any rectangular shape and not only square submatrices as in [7, 10]. Our main results can be summarized as follows:

- we show that using rectangular submatrices all the above 1D measures can be naturally generalized to 2D strings, and we compare their properties with those of the square-based 2D measures introduced in [7, 10];
- we establish some relationships between the new 2D measures and we prove that some properties which are valid in 1D are no longer valid in 2D; other properties are still valid but the gap between some measures can be asymptotically much larger than in 1D;
- we show that the measures δ and γ, which have a simple definition in terms of the submatrices of the input, are not as expressive as in 1D, while the measures g, g_{rl} and b, which are based on a copy-paste mechanism, appear to retain their role of capturing the repetitiveness of the input even in 2D;
- we show that a 2D grammar representing an $m \times n$ string can be enriched with additional information supporting the random access to individual symbols of the original string in $O(\log mn)$ time.
- we show that the 2D Block Tree data structure, being based on square submatrices, fails to capture the regularities of some inputs which are instead captured by the measures g, g_{rl} and b;
- we use our generalized measures to analyze a frequently used heuristics for 2D compression, namely *linearization*, i.e. the transformation of the input matrix into

a 1D string which is then compressed. We study the effectiveness of this technique for the simple row-by-row linearization and the more complex linearization based on the Peano-Hilbert space-filling curve;
- we show that the measures for 2D strings introduced in this paper can be generalized to d-dimensional strings for any $d > 0$ with similar properties.

Overall, our results shed some light on the difficulties of detecting and exploiting repetitiveness in the 2D setting, and show that some concepts/tools introduced in 1D are less effective in 2D. Our results also strengthen and expand the one in [11] showing that to fully capture the repetitiveness in 2D strings it is necessary to consider the repetition also of non-square substrings. These results suggest that to make the 2D Block Tree more effective we should also consider non-square partitions.

Representations of 2D strings based on grammar compression (approaching the measures g and g_{rl}) and macro schemes (approaching the measure b) appear to be the most compact for some families of 2D strings, and those based on grammar compression also support efficient access to the symbols of the uncompressed string. Such results suggest that it may be worthwhile to study how to generalize the heuristics for building 1D grammars to 2D strings, and whether such heuristics maintain their ability to produce grammars of size provably close to the optimal [12, 13]. Some algorithms generalizing the LZ text parsings to 2D strings are reviewed in [14], however they are heuristic in nature and, to the best of our knowledge, there are no bounds proving that their performances are close to those of optimal (2D) grammars or macro schemes.

2 Notation and Background

Let $\Sigma = \{a_1, a_2, \ldots, a_\sigma\}$ be a finite ordered set of *symbols*, which we call an *alphabet*. A *2D string* $\mathcal{M}_{m\times n}$ is a $(m \times n)$-matrix with m *rows* and n *columns* such that each element $\mathcal{M}[i][j]$ belongs to Σ. The *size* of $\mathcal{M}_{m\times n}$, denoted by $|\mathcal{M}|$, is $N = mn$. Note that a position in $\mathcal{M}_{m\times n}$ consists of a pair (i, j), with $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant n$. Throughout the paper, we assume that for each 2D string $\mathcal{M}_{m\times n}$ it holds that $m, n \geqslant 1$. Note that traditional one-dimensional strings are a special case of 2D strings with $m = 1$. We denote by $\Sigma^{m\times n}$ the set of all matrices with m rows and n columns over Σ. A 2D string in $\Sigma^{m\times n}$ is called *square* if $m = n$.

The concatenation between two matrices is a partial operation that can be performed horizontally ($\oslash$) or vertically ($\ominus$), with the constraint that the two operands must have the same number of rows (for $\oslash$) or columns (for $\ominus$).

Example 1 Consider the 2D strings

$$A = \begin{bmatrix} \mathtt{a}\ \mathtt{b}\ \mathtt{a} \\ \mathtt{a}\ \mathtt{b}\ \mathtt{b} \end{bmatrix},\ B = \begin{bmatrix} \mathtt{b}\ \mathtt{b}\ \mathtt{a}\ \mathtt{b} \\ \mathtt{b}\ \mathtt{b}\ \mathtt{b}\ \mathtt{b} \end{bmatrix},\ \text{and } C = \begin{bmatrix} \mathtt{a}\ \mathtt{a}\ \mathtt{a} \\ \mathtt{a}\ \mathtt{a}\ \mathtt{b} \\ \mathtt{a}\ \mathtt{b}\ \mathtt{b} \end{bmatrix}.$$

We can obtain new 2D strings by using $\oslash$ and $\ominus$, respectively:

$$A \oslash B = \begin{bmatrix} \mathtt{a}\ \mathtt{b}\ \mathtt{a}\ \mathtt{b}\ \mathtt{b}\ \mathtt{a}\ \mathtt{b} \\ \mathtt{a}\ \mathtt{b}\ \mathtt{b}\ \mathtt{b}\ \mathtt{b}\ \mathtt{b}\ \mathtt{b} \end{bmatrix}, \text{ and } A \ominus C = \begin{bmatrix} \mathtt{a}\ \mathtt{b}\ \mathtt{a} \\ \mathtt{a}\ \mathtt{b}\ \mathtt{b} \\ \mathtt{a}\ \mathtt{a}\ \mathtt{a} \\ \mathtt{a}\ \mathtt{a}\ \mathtt{b} \\ \mathtt{a}\ \mathtt{b}\ \mathtt{b} \end{bmatrix}.$$

Note that $A \ominus B$ and $A \oslash C$ are undefined.

The concatenation operations have been described in [15] where concepts and techniques of formal languages have been generalized to two dimensions. Such operations have been used in [16] to define Straight-Line Programs for 2D strings that we will recall and generalize in Section 4.

We denote by $\mathcal{M}_{m\times n}[i_1\,..\,i_2][j_1\,..\,j_2]$ the submatrix starting at position (i_1, j_1) and ending at position (i_2, j_2). We say that a matrix F is a *factor* or *substring* of $\mathcal{M}_{m\times n}$ if there exist two positions (i_1, j_1) and (i_2, j_2) such that $F = \mathcal{M}_{m\times n}[i_1\,..\,i_2][j_1\,..\,j_2]$. Given a 2D string $\mathcal{M}_{m\times n}$, the *2D substring complexity* function $P_{\mathcal{M}}$ counts for each pair of positive integers (k_1, k_2) the number of distinct $(k_1 \times k_2)$-factors in $\mathcal{M}_{m\times n}$.

Example 2 Consider the 2D strings

$$M = \begin{bmatrix} \mathtt{a}\ \mathtt{a}\ \mathtt{b}\ \mathtt{b} \\ \mathtt{a}\ \mathtt{a}\ \mathtt{b}\ \mathtt{b} \\ \mathtt{a}\ \mathtt{a}\ \mathtt{b}\ \mathtt{b} \\ \mathtt{a}\ \mathtt{a}\ \mathtt{b}\ \mathtt{b} \\ \mathtt{a}\ \mathtt{a}\ \mathtt{b}\ \mathtt{b} \end{bmatrix}, F_1 = \begin{bmatrix} \mathtt{a}\ \mathtt{b} \\ \mathtt{a}\ \mathtt{b} \\ \mathtt{a}\ \mathtt{b} \end{bmatrix}, F_2 = \begin{bmatrix} \mathtt{a}\ \mathtt{a} \\ \mathtt{a}\ \mathtt{a} \end{bmatrix}, F_3 = \begin{bmatrix} \mathtt{b}\ \mathtt{b} \\ \mathtt{b}\ \mathtt{b} \end{bmatrix} \text{ and } F_4 = \begin{bmatrix} \mathtt{a}\ \mathtt{b} \\ \mathtt{a}\ \mathtt{b} \end{bmatrix}.$$

The 2D string F_1 is a (3×2)-factor of M, as $F_1 = M[2\,..\,4][2\,..\,3]$. Moreover, it can be verified that F_2, F_3 and F_4 are the only (2×2)-factors of M. Hence, $P_M(2, 2) = 3$.

The main purpose of this paper is to generalize the repetitiveness measures δ, γ, b, g, and g_{rl} introduced in the last decade for 1D strings [2] to 2D strings and higher dimensional strings. Some initial results in this area have been recently obtained in [10] with the definition of generalizations of the measures δ, γ, and b to 2D strings looking at their *square* factors. Such measures will be recalled and analyzed in this paper using the symbols $\delta_\square$, $\gamma_\square$, and $b_\square$. Working with only square factors ensures that $\delta_\square$ can be computed in linear time, and makes this measure useful for the analysis of the 2D block tree by Brisaboa et al [6]. In this paper we consider repetitiveness measures defined in terms also of non-square factors since we are interested in their expressive power regardless of algorithmic considerations, and we want to compare them with grammar-induced measures whose definitions involve non-square factors.

Throughout this paper, we assume that the model of computation is the word RAM model: if n is the size of the input, the word size is $\Theta(\log n)$.

3 Measures δ and γ for 2D Strings

In this section, we extend to 2D strings the notions of δ and γ measures [1, 2, 17].

Definition 1 Let $\mathcal{M}_{m\times n}$ be a 2D string and $P_{\mathcal{M}}$ be the 2D substring complexity of $\mathcal{M}_{m\times n}$. Then, $\delta(\mathcal{M}_{m\times n}) = \max\{P_{\mathcal{M}}(k_1, k_2)/k_1k_2, 1 \leqslant k_1 \leqslant m, 1 \leqslant k_2 \leqslant n\}$.

Note that for 1D strings (i.e. $m = 1$) the above definition coincides with the one used in the literature [2, 18]. Recently, in [7] Carfagna and Manzini introduced an alternative extension of δ, here denoted by $\delta_{\square}$, limited to square 2D input strings and using only *square* factors for computing the substring complexity. Below, we report the definition of such a measure, applied to a generic two-dimensional string.

Definition 2 Let $\mathcal{M}_{m\times n}$ be a 2D string and $P_{\mathcal{M}}$ be the 2D substring complexity of $\mathcal{M}_{m\times n}$. Then, $\delta_{\square}(\mathcal{M}_{m\times n}) = \max\{P_{\mathcal{M}}(k, k)/k^2, 1 \leqslant k \leqslant \min\{m, n\}\}$.

From the definitions of $\delta_{\square}$ and δ, the following lemma easily follows.

Lemma 3 *For every 2D string $\mathcal{M}_{m\times n}$ it holds that $\delta(\mathcal{M}_{m\times n}) \geqslant \delta_{\square}(\mathcal{M}_{m\times n})$.*

Although the two measures δ and $\delta_{\square}$ may seem similar, considering square factors instead of rectangular ones may result in very different values. Example 4 shows how different the two measures can be when applied to one-dimensional strings, while Example 5 shows that there exist families of square 2D strings for which $\delta_{\square} = o(\delta)$.

Example 4 Given a 1D string $S \in \Sigma^n$, let $\mathcal{M}_{1\times n} \in \Sigma^{1\times n}$ be the matrix such that $\mathcal{M}_{1\times n}[1][1\,..\,n] = S[1\,..\,n]$. Since the only squares that occur in $\mathcal{M}_{1\times n}$ are the factors of size 1×1, it is $\delta_{\square}(\mathcal{M}_{1\times n}) = P_{\mathcal{M}}(1, 1)/1^2 \leqslant |\Sigma|$. On the other hand, $\delta(\mathcal{M}_{1\times n}) = \delta(S)$.

Example 5 Let $\mathcal{M}_{n\times n}$ be the square 2D string in [7, Lemma 4]. Assuming n is an even perfect square, the first row of $\mathcal{M}_{n\times n}$ is the string $S = B_1B_2\,..\,B_{\sqrt{n}/2}$ composed by $\sqrt{n}/2$ blocks, each one of size $2\sqrt{n}$, with $B_i = 1^i0^{(2\sqrt{n}-i)}$. The remaining rows of $\mathcal{M}_{n\times n}$ are all $\#^n$. In [7, Lemma 4] it is shown that $\delta_{\square}(\mathcal{M}_{n\times n}) = O(1)$. On the other hand, notice that for $i \in [2\,..\,\sqrt{n}/2]$ and $j \in [0\,..\,\sqrt{n} - i]$, the strings $0^j1^i0^{\sqrt{n}-j-i}$ are all different substrings of length $\sqrt{n}$ of S. Since these substrings are in total $\Omega(n)$, it is $\delta(\mathcal{M}_{n\times n}) = \Omega(\sqrt{n})$.

The following definition, first introduced in [19, Sect. 4], generalizes to 2D strings the notion of string attractor [1].

Definition 3 An *attractor* for a 2D string $\mathcal{M}_{m\times n}$ is a set $\Gamma \subseteq [1\,..\,m] \times [1\,..\,n]$ with the property that any substring $\mathcal{M}[i\,..\,j][k\,..\,l]$ of $\mathcal{M}_{m\times n}$ has an occurrence $\mathcal{M}[i'\,..\,j'][k'\,..\,l']$ such that $\exists(x, y) \in \Gamma$ with $i' \leqslant x \leqslant j'$ and $k' \leqslant y \leqslant l'$. The size of the smallest attractor for $\mathcal{M}_{m\times n}$ is denoted by $\gamma(\mathcal{M}_{m\times n})$.

When $m = 1$, the above definition coincides with the one for 1D strings, hence the measure γ inherits the properties for the one-dimensional case [1, 20]. In particular: γ is not monotone and computing $\gamma(\mathcal{M}_{m\times n})$ is NP-hard. In addition, the following property holds.

Proposition 6 *For every 2D string $\mathcal{M}_{m\times n}$, it is $\delta(\mathcal{M}_{m\times n}) \leqslant \gamma(\mathcal{M}_{m\times n})$.*

Proof Reasoning as in the 1D case, observing that any attractor position belongs to at most k_1k_2 factors of shape $k_1 \times k_2$, we get that for any 2D string $\mathcal{M}$ it is $P_{\mathcal{M}}(k_1, k_2) \leqslant k_1k_2\gamma(\mathcal{M})$. □

The next proposition shows that in the 2D context, the gap between δ and γ can be larger than the one-dimensional case, where it is logarithmic [18].

Proposition 7 *For all $m, n \geqslant 1$ there exists a 2D string $\mathcal{M}_{m\times n}$ such that $\delta(\mathcal{M}_{m\times n}) = O(1)$ and $\gamma(\mathcal{M}_{m\times n}) = \Omega(\min(m, n))$.*

Proof Let I_k be the $k \times k$ identity matrix. For all $m, n \geqslant 1$, let us consider the 2D string $\mathcal{M}_{m\times n}$ such that $\mathcal{M}_{m\times n}[1\,..\,\min(m, n)][1\,..\,\min(m, n)] = I_{\min(m,n)}$, and all the remaining symbols are 0's (see Fig. 1 for an example). When either $m = 1$ or $n = 1$, the proof is trivial from known results on 1D strings, so let us assume $m, n \geqslant 2$. Let us further assume $m < n$, next we show that $\Gamma = \{(2, 2)\,..\,(m - 1, m - 1)\} \cup \{(1, m), (m, 1), (m, m + 1)\}$ is an attractor for $\mathcal{M}_{m\times n}$. All the substrings of $\mathcal{M}_{m\times n}$ that contain at least two occurrences of 1's have an occurrence crossing the position (i, i), for some $1 < i < m$, while all the substrings that consist of only 0's have an occurrence aligned with one of the 0's at position $(1, m)$, $(m, 1)$, or $(m, m + 1)$.

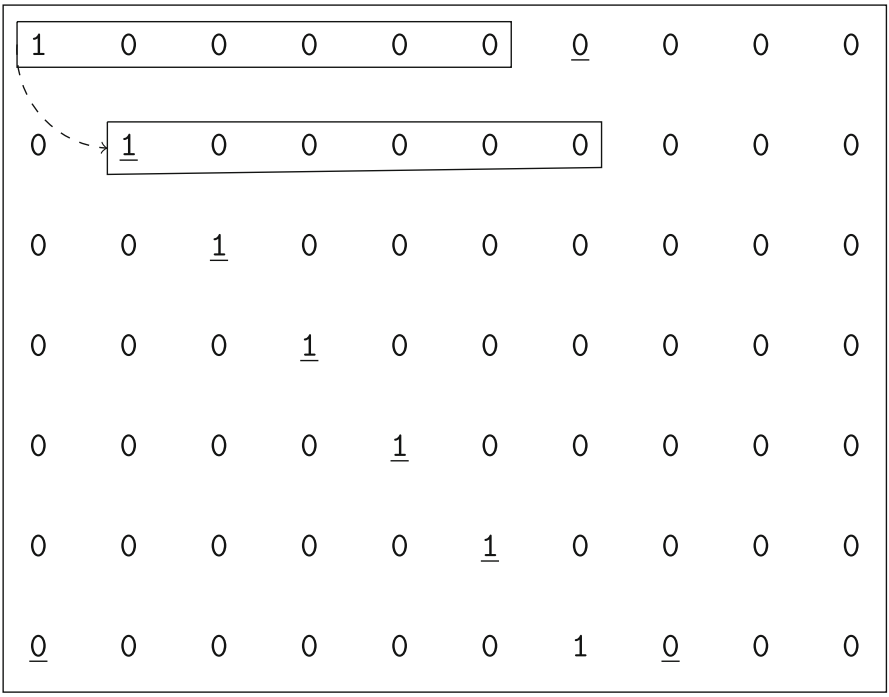

Fig. 1 2D string attractor for the matrix $\mathcal{M}_{m\times n}$ of Proposition 7. The cells whose positions belong to the string attractor are underlined. We show how $M[1\,..\,1][1\,..\,6]$ has an occurrence $M[2\,..\,2][2\,..\,7]$ crossing the string attractor position $(2, 2)$

The factors left contain only one occurrence of 1's that do not cross any position in Γ. These factors of size $k_1 \times k_2$ have to cross either the 1 in position $(1, 1)$ or in position (m, m), and therefore it is either $k_1 = 1$ and $k_2 < m$, or vice versa. Observe that all these factors have another occurrence either starting in $(2, 2)$ or ending in $(m-1, m-1)$, and therefore Γ is an attractor of $\mathcal{M}_{m\times n}$. Since $\mathcal{M}_{m\times n}$ has $m+1$ distinct columns the above attractor has minimum size, i.e. $\gamma(\mathcal{M}_{m\times n}) = m+1$. On the other hand, there exist at most $k_1 + k_2$ distinct substrings of size $k_1 \times k_2$ in $\mathcal{M}_{m\times n}$: $k_1 + k_2 - 1$ correspond to substrings where the diagonal of $\mathcal{M}_{m\times n}$ touches ones of the positions in the left or upper borders of the factor; the last one is the string of only 0's. Hence $\delta(\mathcal{M}_{m\times n}) \leqslant 2$. The case $m > n$ is treated symmetrically by considering the attractor $\Gamma' = \{(2, 2)\,..\,(n-1, n-1)\} \cup \{(1, n), (n, 1), (n+1, n)\}$. For the case $n = m$ it is $\mathcal{M}_{m\times n} = I_n$ and reasoning as above it is easy to see that $\Gamma'' = \{(2, 2)\,..\,(n-1, n-1)\} \cup \{(1, n), (n, 1)\}$ of size n is a minimal attractor for I_n and that $\delta(I_n) \leqslant 2$. □

In [7] the authors introduced an alternative definition of string attractors for square 2D input strings in which they consider only square factors. We can define such a measure, denoted by $\gamma_\square$, also for generic 2D strings, by simply considering only square substrings of $\mathcal{M}_{m\times n}$ in Definition 3. From the definitions of γ and $\gamma_\square$ we immediately get the following relationship:

Lemma 8 *For every 2D string $\mathcal{M}_{m\times n}$ it holds that $\gamma(\mathcal{M}_{m\times n}) \geqslant \gamma_\square(\mathcal{M}_{m\times n})$.*

The following example shows that γ and $\gamma_\square$ can be asymptotically different.

Example 9 Consider again the $m \times m$ identity matrix I_m. For each $k \leqslant m$, a $k \times k$ square factor of I_m either consists of i) all 0's, or ii) all 0's except only one diagonal composed by 1's. Hence, all square factors of type i) have an occurrence that includes position $(m, 1)$ (i.e. the bottom left corner), while all those of type ii) have an occurrence that includes the position $(\lfloor m/2 \rfloor, \lfloor m/2 \rfloor)$ (i.e. the 1 at the center). It follows that $\gamma_\square(I_m) = 2 \in O(1)$, while from the proof of Proposition 7 it can be deduced that $\gamma(I_m) = \Theta(m)$.

An important feature that many practical compressibility measures usually have is *reachability*. A measure μ is reachable if we can represent any string w in $O(\mu(w))$ words of space. We can generalize this notion to 2D strings.

Definition 4 A 2D compressibility measure μ is *reachable* if every string w can be represented in $O(\mu(w))$ words of space.

The measures δ and γ inherit from the 1D case the property that δ is unreachable and γ is unknown to be reachable [2]; but we will introduce some reachable 2D measures in the following sections.

One of the original motivations for considering the measure δ is that in the 1D case it can be computed in linear time and small extra space (see [21] and references therein). Since it is based only on square factors the measure $\delta_\square$ can be still computed in linear time [10, Theorem 2], while no linear time algorithm is known for computing δ for 2D strings. However, by simply enumerating all factors $\delta(\mathcal{M}_{m\times n})$ can be computed in time polynomial in $\max(m, n)$.

4 (Run-Length) Straight-Line Programs for 2D Strings

In this section, we consider a generalization of SLPs for the two-dimensional space introduced in [16] and use it to generalize the measures g and g_{rl} to 2D strings.

Definition 5 Let $\mathcal{M}_{m\times n}$ be a 2D string. A *2-dimensional Straight-Line Program* (2D SLP) for $\mathcal{M}_{m\times n}$ is a context-free grammar (V, Σ, R, S) that uniquely generates $\mathcal{M}_{m\times n}$ where V is the set of non-terminal symbols or variables, Σ is the set of terminal symbols, $S \in V$ is the axiom/starting symbol of the grammar and R is the set of rules. A rule $A \to \alpha$ in R with $A \in V$ can have one of the following three forms depending on its right-hand side α:

$$A \to a, \quad A \to B \oplus C, \quad \text{or } A \to B \ominus C$$

where $a \in \Sigma$, $B, C \in V$. We call these definitions *terminal rules*, *horizontal rules*, and *vertical rules*, respectively and their corresponding expansion is defined as

$$\exp(A) = a, \quad \exp(A) = \exp(B) \oplus \exp(C), \quad \text{or } \exp(A) = \exp(B) \ominus \exp(C)$$

The size $|G|$ of a 2D SLP G is the sum of the sizes of all the rules of G, where we assume that the terminal rules have size 1, and the horizontal and vertical rules have size 2. The measure $g(\mathcal{M}_{m\times n})$ is defined as the size of the smallest 2D SLP generating $\mathcal{M}_{m\times n}$.

Note that, by our definition of the $\oplus$ operator, a horizontal rule $A \to B \oplus C$ requires that the number of rows of $\exp(B)$ and $\exp(C)$ coincides, and the same must be true for the number of columns for a vertical rule $A \to B \ominus C$. We further assume that two distinct variables $A, B \in V$ do not have the same right-hand side.

For 2D SLP it is convenient to introduce the concepts of *parse tree* and *grammar tree*, see Figs. 2 and 3 respectively.

Definition 6 Let $G = (V, \Sigma, R, S)$ be a 2D SLP. The *parse tree* T_G of G is a directed labeled graph $T_G = T(S)$ obtained recursively as follows. For each variable $A \in V$:

- If $A \to a \in R$, then $T(A)$ is a tree with root A having a single child which is the leaf a.
- If $A \to B \oplus C \in R$, then $T(A)$ is a tree with root A, $T(B)$ as its left subtree, and $T(C)$ as its right subtree.
- If $A \to B \ominus C \in R$, then $T(A)$ is a tree with root A, $T(B)$ as its up subtree, and $T(C)$ as its down subtree.

Note that all the nodes with the same label are considered different. By *traversal of the parse tree* T_G, we mean the usual visit in preorder of the parse tree of the 2D SLP G, with the following peculiarity: for each node A corresponding to a horizontal rule $A \to B \oplus C$, we first visit the node A, then the left subtree $T(B)$, and then the right subtree $T(C)$; for each node A corresponding to a vertical rule $A \to B \ominus C$, we first visit the node A, then the up subtree $T(B)$, and then the down subtree $T(C)$.

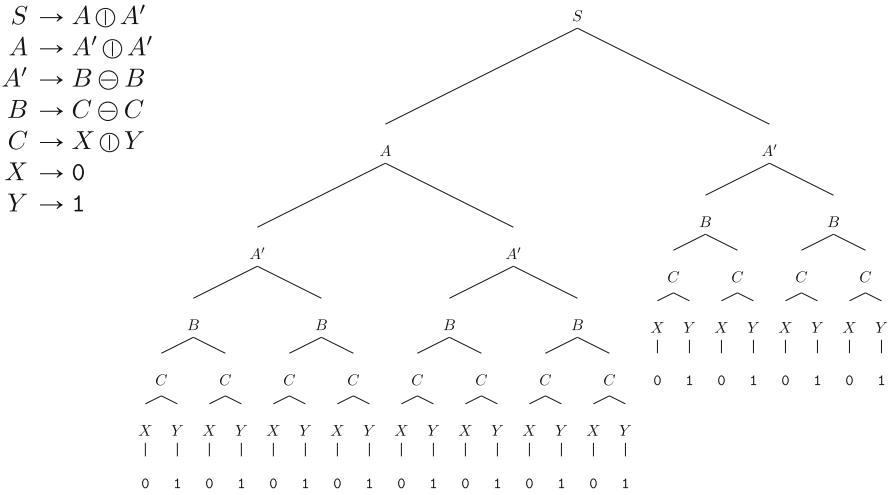

Fig. 2 Example of a 2D SLP $G = (V, \Sigma, R, S)$, where $V = \{S, A, A', B, C, X, Y\}$, $\Sigma = \{0, 1\}$, and the set of rules R is displayed on the left. The parse tree is displayed on the right. The 2D string $\mathcal{M} = \exp(S)$ is displayed in Fig. 3 (right). By exhaustive search, the grammar G has minimum size for $\mathcal{M}$, hence $g(\mathcal{M}) = 12$

The tree T_G has $\exp(\mathcal{M}_{m\times n}) = N$ leaves. For simplicity, given a tree node labeled A, we use $\exp(A)$ to denote the 2D string obtained by expanding the corresponding rule $A \to \alpha$ of G. A node in T_G that has no braching and corresponds to a production rule of the form $A \to a$ (i.e. $\exp(A) \in \Sigma$) is called *leaf-generating node*.

Definition 7 Let $G = (V, \Sigma, R, S)$ be a 2D SLP. We call *primary occurrence* of a variable $A \in V$ the first node labeled with A encountered in the traversal of the parse tree T_G. The remaining nodes labeled with the same A are then called *secondary*

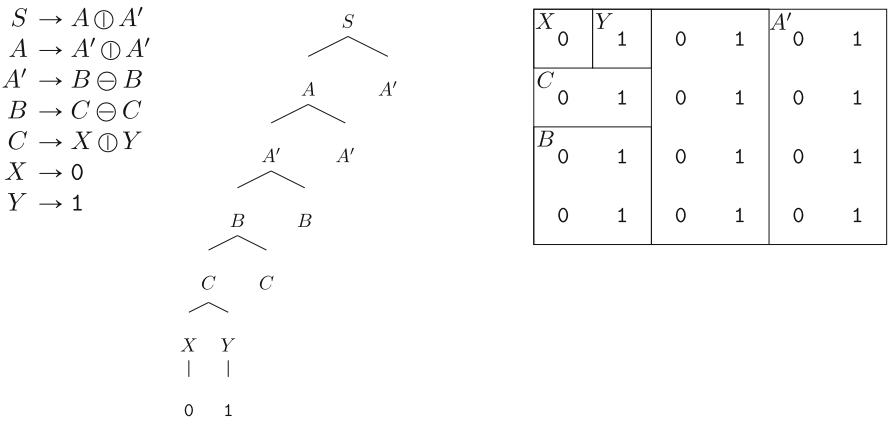

Fig. 3 The same 2D SLP as in Fig. 2, but instead of the parse tree we show the *grammar tree* where each variable is expanded exactly once. On the right, we show the generated binary 2D string with highlighted the expansion of some of the variables

occurrences. A *grammar tree* for G is constructed from the parse tree T_G by keeping all the nodes that are either primary occurrences or children of a primary occurrence, while the children of every secondary occurrence are pruned.

Proposition 10 *It always holds that* $g(\mathcal{M}_{m\times n}) = \Omega(\log(mn))$.

Proof The proof proceeds as the one of [22, Lemma 1] which proves the analogous result for strings and 1D context-free grammars. Given a matrix M of size $N = nm$ and any 2D SLP G for M, we build a sequence of non-terminals $\{A_i\}_{i=1,\dots,k}$ defined as follows: A_1 is the starting symbol of G and A_{i+1} is a non-terminal symbol in the right-hand side α of the rule $A_i \to \alpha$ with maximal expansion. Note that since G is acyclic all the A_i must be distinct and therefore by continuing to extend the sequence we eventually reach a non-terminal A_k with $|\exp(A_k)| = 1$; moreover, k is at most the number of rules, hence $k \leqslant g$. Notice that for every rule $A_i \to \alpha$ the size of the expansion $|\exp(A_i)|$ is at most $2|\exp(A_{i+1})|$, that is, each rule application can at most double the size of the produced matrix. Therefore, it holds that $N = |\exp(A_1)| \leqslant 2|\exp(A_2)| \leqslant \cdots \leqslant 2^{k-1}|\exp(A_k)| < 2^g$ and thus $g = \Omega(\log N)$. □

Proposition 11 *The problem of determining if there exists a 2D SLP of size at most k generating a 2D string* $\mathcal{M}_{m\times n}$ *is NP-complete.*

Proof The problem belongs to NP because its 1D version, which is known to be NP-complete [22], can be reduced to the 2D version by interpreting 1D strings as matrices of size $1 \times n$. □

As in the 1D case, we can extend 2D SLPs with *run-length rules* obtaining more powerful grammars.

Definition 8 A *2-dimensional Run-Length Straight-Line Program* (2D RLSLP) is a 2D SLP that in addition allows special rules, which are assumed to be of size 2, of the form

$$A \to ①^k B \text{ and } A \to ⊖^k B$$

for $k > 1$, with their expansions defined respectively as

$$\exp(A) = \underbrace{\exp(B) ① \exp(B) ① \cdots ① \exp(B)}_{k \text{ times}}$$
$$\exp(A) = \underbrace{\exp(B) ⊖ \exp(B) ⊖ \cdots ⊖ \exp(B)}_{k \text{ times}}$$

The measure $g_{rl}(\mathcal{M}_{m\times n})$ is defined as the size of the smallest 2D RLSLP generating $\mathcal{M}_{m\times n}$.

We also introduce the parse tree for 2D RLSLPs (Fig. 4).

Definition 9 Let a 2D RLSLP $G_{rl} = (V, \Sigma, R, S)$. The *parse tree* of G_{rl} is a directed labeled graph $T_{G_{rl}} = T(S)$ obtained recursively as follows. For each variable A:

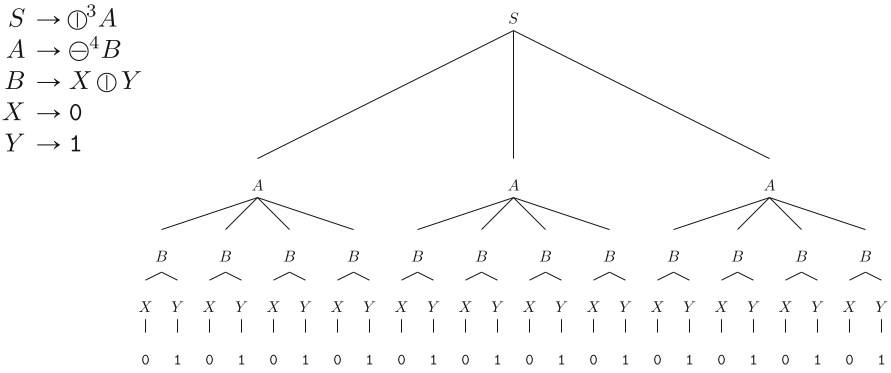

Fig. 4 Example of a 2D RLSLP $G_{rl} = (V, \Sigma, R, S)$, where $V = \{S, A, B, X, Y\}$, $\Sigma = \{0, 1\}$, and the set of rules R is displayed on the left. The parse tree is displayed on the right. The 2D string $\mathcal{M} = \exp(S)$ is the depicted in Fig. 3 (right). By exhaustive search, the grammar G_{rl} has minimum size for $\mathcal{M}$, hence $g_{rl}(\mathcal{M}) = 8$

- If $A \to a \in R$, then $T(A)$ is
  a tree with root A having a single child which is the leaf a.
- If $A \to B ⓘ C \in R$, then $T(A)$ is a tree with root A, $T(B)$ as its left subtree, and $T(C)$ as its right subtree.
- If $A \to B ⊖ C \in R$, then $T(A)$ is a tree with root A, $T(B)$ as its up subtree, and $T(C)$ as its down subtree.
- If $A \to ⓘ^k B \in R$, then $T(A)$ is a tree with root A, with k horizontal copies of $T(B)$ as its subtrees.
- If $A \to ⊖^k B \in R$, then $T(A)$ is a tree with root A, with k vertical copies of $T(B)$ as its subtrees.

As in the case of the parse tree of a 2D SLP, all the nodes with the same label are considered different. By *traversal of the parse tree* $T_{G_{rl}}$, we extend the definition of the traversal of a parse tree of a 2D SLP as follows: for each node A corresponding to a rule $A \to ⓘ^k B$, we first visit the node A, then each copy of the subtree $T(B)$ in order from the leftmost to the rightmost; for each node A to a rule $A \to ⊖^k B$, we first visit the node A, then each copy of the subtree $T(B)$ from the topmost to the bottommost.

From this notion of traversal of parse trees of 2D RLSLP's, the definition of grammar tree of a 2D RLSLP can be naturally derived from Definition 7, with the exception that each node of the grammar tree associated to a run-length rule $A \to ⓘ^k B$ (resp. $A \to ⊖^k B$) has two children as well: the left (resp. up) child is a node labeled by B, while the right (resp. down) child is a leaf labeled by $ⓘ^{k-1} B$ (resp. $⊖^{k-1} B$), that is the remaining $k - 1$ occurrences of B collapse in a single node.

Proposition 12 *For every 2D string $\mathcal{M}_{m\times n}$ it holds that $g_{rl}(\mathcal{M}_{n\times n}) \leqslant g(\mathcal{M}_{m\times n})$. Moreover, there are infinite string families where $g_{rl} = o(g)$.*

Proof The first claim is trivial by definition. The second claim is proven by considering the family of $n \times n$ 2D strings $\mathcal{M}_{n\times n} = 0^{n\times n}$: it is easy to see that the RLSLP $G_{rl} = (V, \Sigma, R, S)$, where $V = \{X, Y, Z\}$, $\Sigma = \{0\}$, $R = \{X \to ⊖^n Y, Y \to$

$①^n Z, Z \to 0\}$, and $S = X$, produces the string $\mathcal{M}_{n\times n}$, and therefore $g_{rl} = O(1)$. On the other hand, by Proposition 10, we have $g = \Omega(\log n)$, and the thesis follows. □

5 Efficient Direct Access on 2D (RL)SLPs

In this section, we show that a 2D RLSLP G_{rl} representing a matrix of size $m \times n$ can be enriched with additional information to support random access to each element of the matrix in $O(\log mn)$ time and $O(|G_{rl}| \log mn)$ bits of space. We first explain the result only for 2D SLPs, as it is easier to understand and generalize later to 2D RLSLPs.

Our approach builds upon the *heavy-path decomposition* of Bille et al. [23], defined for one-dimensional SLPs. In our setting, we consider the parse tree T_G and we assign to each node labeled by non-terminal $A \in V$ the weight $|\texttt{exp}(A)|$. The *heavy-path* associated to the node labeled by A is the path $A_0, A_1, \ldots, A_k$ in T_G, where $A_i \in V$ for each $i = 0, \ldots, k$, $A_0 = A$ and $\texttt{exp}(A_k) \in \Sigma$ (i.e. A_k is the label of a leaf-generating node). Starting from A, at each step the heavy-path extends through the *heaviest* child, i.e. the child with the greatest weight. In case of tie, the left child (in horizontal rules) or the up child (in vertical rules) is chosen, ensuring that each node is assigned a unique heavy path.

The key idea behind our algorithm is as follows: for each non-terminal $A \in V$, we store the terminal symbol a and its coordinates (y_A, x_A) within $\texttt{exp}(A)$, which is reached by traversing the heavy-path in the parse tree. If the queried position (y, x) coincides with (y_A, x_A), we return a. Otherwise, by using efficient (opportune) data structures, we identify the lowest common ancestor (LCA) $A_i \in V$ of the leaves corresponding to the positions (y_A, x_A) and (y, x). We then recursively continue the search in the child B of A_i that was not included in the heavy-path of A, adjusting the query coordinates to $(y', x') \leftarrow (y - r_B + 1, x - c_B + 1)$ where r_B and c_B denote the top-left coordinates of the occurrence of B within A. By starting the algorithm from the axiom S, it is easy to see that we will eventually access the element at the desired position.

This section is organized in the following way: we first describe how the nodes in a heavy-path are chosen from the parse tree of a 2D SLP; next, we describe the data structures that we need to efficiently perform lowest common ancestor queries; then, we show the space and time analysis of the algorithm; moreover, we describe the algorithm, and explain how to adapt the technique described to work with 2D RLSLP's; finally, we show how to optimize our data structures.

5.1 2D Heavy-Path Decomposition

Let $G = (V, \Sigma, R, S)$ a 2D SLP generating $\mathcal{M}_{m\times n}$ and let T_G be its parse tree. We consider the labels of the edges of T_G connecting nodes labeled by non-terminal symbols. More formally, we denote by $E \subseteq V \times V \times \{\texttt{L}, \texttt{R}, \texttt{U}, \texttt{D}\}$ the set of the *edge labels* in the parse tree with leaves pruned, where each edge label is uniquely identified

by the label of the parent, the label of the child and a spatial relationship as described below:

- $(A, B, \mathtt{L}), (A, C, \mathtt{R}) \in E$ if $A \to B \oplus C \in R$;
- $(A, B, \mathtt{U}), (A, C, \mathtt{D}) \in E$ if $A \to B \ominus C \in R$.

Definition 10 Let $G = (V, \Sigma, R, S)$ be a 2D SLP generating $\mathcal{M}_{m\times n}$ and T_G its parse tree. We denote by $E_{\mathrm{h}} \subseteq E$ the set of *heavy edge labels* formed by:

1. $(A, B, \mathtt{L})$ if $A \to B \oplus C \in R$ and $|\mathtt{exp}(B)| \geqslant |\mathtt{exp}(C)|$;
2. $(A, C, \mathtt{R})$ if $A \to B \oplus C \in R$ and $|\mathtt{exp}(B)| < |\mathtt{exp}(C)|$;
3. $(A, B, \mathtt{U})$ if $A \to B \ominus C \in R$ and $|\mathtt{exp}(B)| \geqslant |\mathtt{exp}(C)|$;
4. $(A, C, \mathtt{D})$ if $A \to B \ominus C \in R$ and $|\mathtt{exp}(B)| < |\mathtt{exp}(C)|$.

Definition 11 The *2D heavy-path decomposition* of T_G is a directed labeled forest H obtained by considering only the edges with label $e \in E_{\mathrm{h}}$. If a node u labeled by A has two children v and z, labeled by B and C respectively, and $(A, B, p) \in E_{\mathrm{h}}$, for some $p \in \{\mathtt{L}, \mathtt{R}, \mathtt{U}, \mathtt{D}\}$, then the node v is called *heavy child* of u, and z is called *light child* of u. We call *heavy-path* of a node u any directed path in H from u to some leaf-generating node v.

We can associate each heavy path with the sequence of labels of its nodes. Note that the heavy paths of two nodes labeled with the same label are associated to the same sequence of labels.

Lemma 13 *It holds* $|E_h| = |V| - |\Sigma| = O(|G|)$.

A key property of the heavy edges is that not following them rapidly decreases the distance to a leaf of the parse tree.

Lemma 14 *Any path from the root to a leaf-generating node in* T_G *traverses at most* $\log_2 N$ *edges not in* H.

Proof The result follows since traversing a light edge (i.e. an edge not in H) at least halves the size of the 2D string corresponding to the reached subtree, that is for a light edge $A \to B$ it is $|\mathtt{exp}(A)| \geqslant 2|\mathtt{exp}(B)|$. □

5.2 The Algorithm

Let $A_0, A_1, \ldots, A_k$ be the sequence of labels associated with a heavy-path. The *heavy symbol* of A_i, for all $i = 0, \ldots, k$, is the symbol $\mathtt{exp}(A_k) \in \Sigma$. The *heavy occurrence* of the symbol $\mathtt{exp}(A_k) \in \Sigma$ within $\mathtt{exp}(A_i)$ is defined recursively as the position (y_i, x_i), where:

$$y_i = \begin{cases} 1 & \text{if } i = k \\ y_{i+1} & \text{if } A_i \text{ corresponds to a horizontal rule or } A_i \to A_{i+1} \ominus B \\ y_{i+1} + m & \text{if } A_i \to B \ominus A_{i+1} \text{ with } \mathtt{exp}(B) \in \Sigma^{m\times n} \end{cases}$$

and

$$x_i = \begin{cases} 1 & \text{if } i = k \\ x_{i+1} & \text{if } A_i \text{ corresponds to a vertical rule or } A_i \to A_{i+1} \oslash B \\ x_{i+1} + n & \text{if } A_i \to B \oslash A_{i+1} \text{ with } \exp(B) \in \Sigma^{m \times n} \end{cases}$$

In other words, the heavy occurrence is the position in $\exp(A_i)$ reached by following the heavy-path starting from A_i.

Suppose we are trying to access the position (y, x) in A_0, i.e. $\exp(A_0)[y][x]$. As described at the beginning of this section, the algorithm checks whether the heavy occurrence (y_0, x_0) of $\exp(A_k)$ within $\exp(A_0)$ is exactly the position we are looking for: if the positions coincide we simply return the letter explicitly stored; otherwise, we look in the heavy path for the lowest common ancestor A_i between the leaves corresponding to the positions (y, x) and (y_0, x_0); then, we repeat recursively the procedure on the light child B of A_i, this time though looking for the position (y', x') that corresponds to the position (y, x) moved by an offset derived from the top-left corner of this occurrence of B in A_0, thus ensuring $\exp(A_0)[y][x] = \exp(B)[y'][x']$.

To retrieve the lowest common ancestor A_i and to compute the updated position (y', x'), we need the following information for each heavy-path:

- the *up size sequence* $u_0, u_1, \ldots, u_k$, where u_i is the sum of the number of rows of the light up children of each of the first i nodes in the heavy-path;
- the *down size sequence* $d_0, d_1, \ldots, d_k$, where d_i is the sum of the number of rows of the light down children of the first i nodes in the heavy-path;
- the *left size sequence* $l_0, l_1, \ldots, l_k$, where l_i is the sum of the number of columns of the light left children of the first i nodes in the heavy-path;
- the *right size sequence* $r_0, r_1, \ldots, r_k$, where r_i is the sum of the number of columns of the light right children of the first i nodes in the heavy-path.

Each sequence above counts for each i the number of rows above/below and columns to the left/right of the 2D substring $\exp(A_i)$ within $\exp(A_0) \in \Sigma^{m \times n}$ reached by following the heavy paths, that is,

$$\exp(A_0)[u_i + 1 \,..\, m - d_i][l_i + 1 \,..\, n - r_i] = \exp(A_i).$$

The following lemma is a direct consequence of the definitions of up/down/left/right size sequences.

Lemma 15 *Let $A_0, A_1, \ldots, A_k$ be the sequence of labels associated with a heavy-path in a parse tree. The following relationships hold:*

- $u_0 + 1 \leqslant u_1 + 1 \leqslant \ldots \leqslant u_k + 1 = m - d_k \leqslant \ldots \leqslant m - d_0$;
- $l_0 + 1 \leqslant l_1 + 1 \leqslant \ldots \leqslant l_k + 1 = n - r_k \leqslant \ldots \leqslant n - r_0$.

To locate the lowest common ancestor between the leaves corresponding to the positions (y_0, x_0) and (y, x) in $\exp(A_0)$, we look for the largest index i such that

$$u_i + 1 \leqslant y \leqslant m - d_i \tag{1}$$

and

$$l_i + 1 \leqslant x \leqslant n - r_i. \tag{2}$$

Based on the values of y with respect to y_0 we behave as follows:

1.a if $y > y_0$, we find the largest i verifying $y \leqslant m - d_i$ by performing a predecessor query of $m - y + 1$ in the down size sequence;
1.b if $y < y_0$, we find the largest i verifying $y \geqslant u_i + 1$ by performing a predecessor query of y in the up size sequence;
1.c if $y = y_0$, return $i = k$.

We proceed analogously according to the values of x with respect to x_0:

2.a if $x > x_0$, we find the largest j verifying $x \leqslant n - r_j$ by performing a predecessor query of $n - x + 1$ in the right size sequence;
2.b if $x < x_0$, we find the largest j verifying $x \geqslant l_j + 1$ by performing a predecessor query of x in the left size sequence;
2.c if $x = x_0$, return $j = k$.

Observe that if the conditions in 1.c and 2.c are both true, then we are done and we just return $\texttt{exp}(A_0)[y_0][x_0]$. Otherwise, observe that by Lemma 15 only $i' = \min(i, j)$ verifies Equations 1 and 2 at the same time.

Finally, we can recall the same procedure to the light child B of $A_{i'}$, after updating the position (y', x') to search within $\texttt{exp}(B)$. If we denote by $m_{i'+1}$ and $n_{i'+1}$ the number of rows and columns respectively in $A_{i'+1}$, we compute (y', x') as follows:

$$y' = \begin{cases} y - u_{i'} & \text{if } A_{i'} \text{ corresponds to a horizontal rule or } A_{i'} \to B \ominus A_{i'+1} \\ y - u_{i'} - m_{i'+1} & \text{if } A_{i'} \to A_{i'+1} \ominus B \end{cases}$$

and

$$x' = \begin{cases} x - l_{i'} & \text{if } A_{i'} \text{ corresponds to a vertical rule or } A_{i'} \to B \oslash A_{i'+1} \\ x - l_{i'} - n_{i'+1} & \text{if } A_{i'} \to A_{i'+1} \oslash B \end{cases}$$

Figure 5 illustrates heavy-paths and direct access on 2D SLPs.

5.3 Data Structures

Recall that the heavy-paths of two distinct nodes with the same non-terminal symbol share the same sequence of labels. This means that the sequence of labels $A_{0_j}, A_{1_j}, \ldots A_{k_j}$ associated with a heavy path of a node labeled by $V_j = A_{0_j} \in V$ is uniquely determined by V_j. Then, for each variable $V_j \in V$, with $j = 1, \ldots, |V|$ we need to store the following information:

- The number of rows m_{0_j} and columns n_{0_j} of the 2D string $\texttt{exp}(A_{0_j})$.
- The indexes y_{0_j} and x_{0_j} of the *heavy occurrence* of $\texttt{exp}(A_{k_j})$ within $\texttt{exp}(A_{0_j})$, and the symbol $\texttt{exp}(A_{k_j})[1][1] = \texttt{exp}(A_{0_j})[y_{0_j}][x_{0_j}]$.
- The *up size sequence* $u_{0_j}, u_{1_j}, \ldots, u_{k_j}$;
- The *down size sequence* $d_{0_j}, d_{1_j}, \ldots, d_{k_j}$;

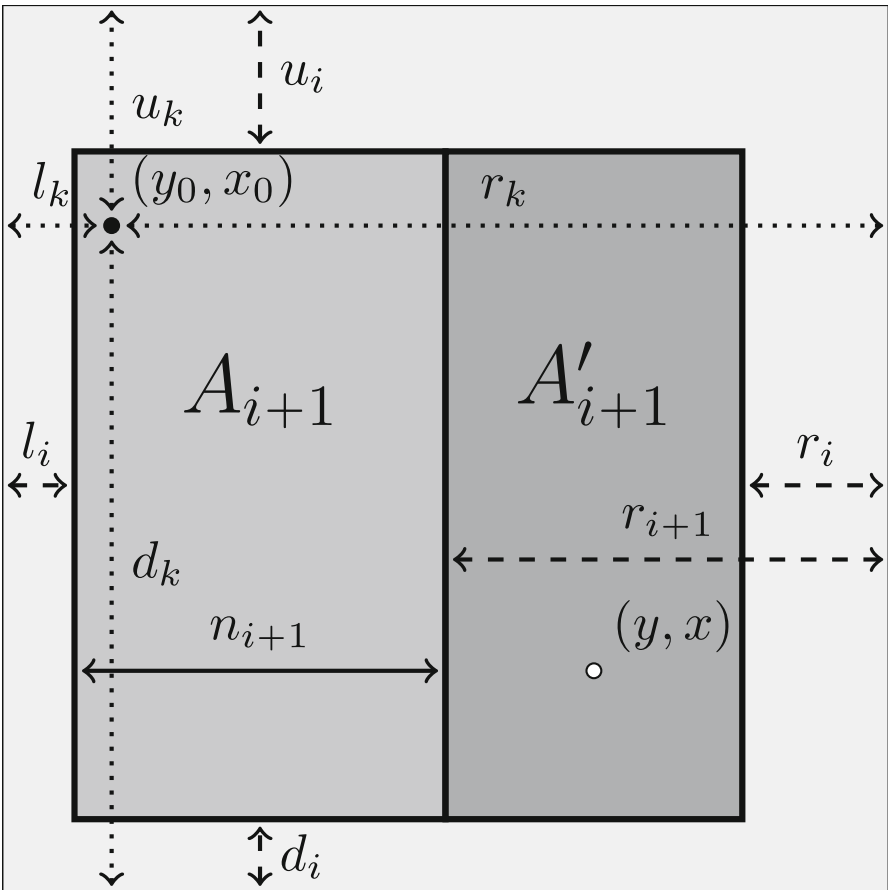

Fig. 5 Components of a heavy-path labeled $A_0, \ldots, A_k$. The 2D string $\texttt{exp}(A_0)$ is represented by the biggest rectangle. The black dot at coordinate (y_0, x_0) is the heavy-occurrence of $\texttt{exp}(A_k)$ in $\texttt{exp}(A_0)$. The rectangle in light gray is $\texttt{exp}(A_{i+1})$, and in dark gray is $\texttt{exp}(A'_{i+1})$. They are generated by the rule $A_i \to A_{i+1} \textcircled{1} A'_{i+1}$. A_{i+1} and A'_{i+1} are the heavy child and light child of A_i, respectively. We further show the values of the up/down/left/right sequence corresponding to the variables A_i, A_{i+1}, and A_k. For A_{i+1} we only represent r_{i+1} since $r_{i+1} \neq r_i$, while $l_{i+1} = l_i$, $u_{i+1} = u_i$ and $d_{i+1} = d_i$. The white dot at coordinates (y, x) is $\texttt{exp}(A_0)[y][x]$, the cell we want to access. Note that A_i is the lowest common ancestor between the heavy-path endpoint $\texttt{exp}(A_0)[y_0][x_0]$ and the leaf $\texttt{exp}(A_0)[y][x]$. After finding A_i, the search must continue within $\texttt{exp}(A'_{i+1})$. More precisely, we look for the position $(y - u_i, x - (l_i + n_{i+1}))$ within $\texttt{exp}(A'_{i+1})$

- The *left size sequence* $l_{0_j}, l_{1_j}, \ldots, l_{k_j}$;
- The *right size sequence* $r_{0_j}, r_{1_j}, \ldots, r_{k_j}$.

Note that the values $m_{0_j}, n_{0_j}, x_{0_j}, y_{0_j}$, and $\texttt{exp}(A_{k_j})$ depend only on V_j itself, hence we can store them for all the variables using $5|V|$ words of space.

To compactly represent all heavy paths and efficiently support predecessor queries, we construct a data structure that encodes the up, down, left and right sequences. This structure is an adaptation to the two-dimensional context of the heavy-path suffix forest

defined in [23], which allows us to reduce a predecessor query to a *weighted ancestor query* [24]. We consider the forest F of tries obtained by reversing the direction of the edges of the heavy-path decomposition H of the parse tree T_G. In this new representation, nodes that were previously leaf-generating become the roots of the new forest F. This means that each trie in F corresponds to a distinct terminal symbol from the alphabet Σ (produced by a leaf-generating node). Then, the number of tries in F is at most Σ. Moreover, every non-terminal symbol appears exactly once in the entire forest F. The core of our data structure is the weighted forest F that stores, for each edge labeled $(A_{j_{i+1}}, A_{j_i}, p)$, where $p \in \{\mathtt{L}, \mathtt{R}, \mathtt{U}, \mathtt{D}\}$, four spatial weights to encode the contribution of light children: a *left weight* w_l, a *right weight* w_r, an *up weight* w_u, and a *down weight* w_d, defined as follows:

- if $p = \mathtt{L}$ (i.e. $A_i \to A_{i+1} \oslash B$), then $w_l = w_u = w_d = 0$ and $w_r = n_B$;
- if $p = \mathtt{R}$ (i.e. $A_i \to B \oslash A_{i+1}$), then $w_r = w_r = w_u = 0$ and $w_l = n_B$;
- if $p = \mathtt{U}$ (i.e. $A_i \to A_{i+1} \ominus B$), then $w_l = w_r = w_u = 0$ and $w_d = m_B$;
- if $p = \mathtt{D}$ (i.e. $A_i \to B \ominus A_{i+1}$), then $w_l = w_r = w_d = 0$ and $w_u = m_B$,

where n_B and m_B denote the number of columns and the number of rows in the matrix $\mathtt{exp}(B)$, respectively. Thus, if a node labeled B is a light child of a node labeled A_i in the parse tree T_G, these values represent the contribution of the light child B to the spatial weights. This data structure integrates both the heavy-path decomposition and the weighted forest, enabling efficient retrieval of lowest common ancestors and fast computation of updated positions, as described in the next subsection. By using the formulation of Bille et al. [23], performing a weighted ancestor query on the up/down/left/right weights is equivalent to performing a predecessor query on the up/down/left/right size sequence, respectively. Each query is performed in $O(\log\log\max(m,n))$ time [24], if G generates the 2D string $\mathcal{M}_{m\times n}$.

5.4 Space and Time Complexities

By Lemma 14, there are at most $\log mn$ light edges in any path from root to a leaf-generating node in the parse tree of a 2D SLP G generating $\mathcal{M}_{m\times n}$. That is, we can change from one heavy-path to another at most $\log mn$ times. The data structures described in the previous paragraph take $O(|V|\log mn) = O(|G|\log mn)$ bits of space, can be built in $O(|G|)$ time, and support predecessor queries in $O(\log\log\max(m,n))$ time [24]. Therefore, we obtain the following result.

Theorem 16 *Let $\mathcal{M}_{m\times n}$ be a 2D string and let G be a 2D SLP generating $\mathcal{M}_{m\times n}$. We can build in $O(|G|)$ preprocessing time a data structure of $O(|G|\log mn)$ bits of space which supports direct access queries to any cell $\mathcal{M}[i][j]$ in $O(\log mn \log\log\max(m,n))$ time.*

5.5 Generalization to 2D RLSLPs

The strategy based on heavy-paths can be generalized to work on 2D RLSLPs, by adding specific additional considerations related to the run-length rules. In particular, given the heavy-path $A_0, A_1, \ldots, A_k$:

1. when a node u labeled by A_i corresponds to the rule $A_i \to \oplus^{\ell} A_{i+1}$ (resp. $A_i \to \ominus^{\ell} A_{i+1}$), the heavy child of u is the leftmost (resp. upmost) child labeled by A_{i+1}; the remaining children are considered light;
2. in the definitions of heavy occurrence we add the conditions that if $A_i \to \oplus^{\ell} A_{i+1}$ or $A_i \to \ominus^{\ell} A_{i+1}$, then $y_i = y_{i+1}$ and $x_i = x_{i+1}$
3. in the definitions of left and up size sequences, if $A_i \to \oplus^{\ell} A_{i+1}$ or $A_i \to \ominus^{\ell} A_{i+1}$, then $l_{i+1} = l_i$ and $u_{i+1} = u_i$; in the definitions of right (resp. down) size sequences, if $A_i \to \oplus^{\ell} A_{i+1}$ (resp. $A_i \to \ominus^{\ell} A_{i+1}$) and $\texttt{exp}(A_{i+1}) \in \Sigma^{m_{i+1} \times n_{i+1}}$, then $r_{i+1} = r_i + (\ell - 1)n_{i+1}$ (resp. $r_{i+1} = r_i$) and $u_{i+1} = u_i$ (resp. $u_{i+1} = u_i + (\ell - 1)m_{i+1}$);
4. let us denote by i' the value returned when steps (1) and (2) described in Subsection 5.2 are applied to find the lowest common ancestor $A_{i'}$ between the leaves corresponding to the heavy symbol of A_0 and $\texttt{exp}(A_0)[y][x]$. If $A_{i'} \to \oplus^{\ell} A_{i'+1}$ (resp. $A_{i'} \to \ominus^{\ell} A_{i'+1}$) then $y' = 1 + (y' - u_{i'}) \mod m_{i'+1}$ (resp. $y' = y' - u_{i'}$) and $x' = 1 + (x' - l_{i'}) \mod n_{i'+1}$ (resp. $x' = x' - l_{i'}$).

Note that the considerations described above allow us to use the same algorithmic strategy presented in Subsection 5.2 and the same data structures defined in Subsection 5.3, while preserving the same time and space complexities. So, we derive the following result.

Theorem 17 *Let $\mathcal{M}_{m \times n}$ be a 2D string and let G_{rl} be a 2D RLSLP generating $\mathcal{M}_{m \times n}$. We can build in $O(|G_{rl}|)$ preprocessing time a data structure of $O(|G_{rl}| \log mn)$ bits of space which supports direct access queries to any cell $\mathcal{M}[i][j]$ in $O(\log mn \log\log \max(m, n))$ time.*

5.6 Speeding up the Algorithm to $O(\log mn)$

The time complexity described in Subsection 5.4 is determined by the $O(\log\log \max(m, n))$ time required for each of the $O(\log mm)$ weighted ancestor queries, where mn is the size of the 2D string $\mathcal{M}_{m \times n}$.

Instead of performing both horizontal and vertical queries at each step of the algorithm, we can first execute only one of the two queries and, in $O(1)$ time, check whether the retrieved node is the lowest common ancestor v between the heavy-occurrence (y', x') of the heavy-path and the queried position (y, x). If the first query does not succeed, we proceed with the second query, which is then guaranteed to return the correct lowest common ancestor v. However, in the best-case scenario, this optimization only reduces the number of queries by half, thus reaching asymptotically the same upper bound.

To get rid of the $O(\log\log \max(m, n))$ factor, we adapt the predecessor data structure, called *interval-biased search tree*, proposed by Bille et al. [23] to our setting. A predecessor query p on this data structure takes $O(\log U/U')$ time, where U is the universe size and U' is the gap between the successor and the predecessor of p. We use this data structure to obtain the following theorem.

Theorem 18 *Let $\mathcal{M}_{m\times n}$ be a 2D string and let G_{rl} be a 2D RLSLP generating $\mathcal{M}_{m\times n}$. There exists a data structure that uses $O(|G_{rl}|\log mn)$ bits of space and supports direct access queries to any cell $\mathcal{M}[i][j]$ in $O(\log mn)$ time.*

Proof We follow the same strategy described in Section 5.2, with the key difference that we execute the queries on the left/right size sequence and on the up/down size sequence concurrently using the interval-biased search trees. As soon as we locate the node in the parse tree where the search should proceed (the light child of the lowest common ancestor), we terminate all other ongoing queries (if any). Note that each potential solution can be checked in $O(1)$ time, ensuring that the time complexity depends only on the query that successfully locates the lowest common ancestor.

We distinguish the query times to locate the lowest common ancestor (LCA) into two groups based on the corresponding type of rule:

1. LCAs with up/down children. If we perform l queries of this type, the complexity for ith query is $O(\log Y_i/Y_i')$, where Y_i is the number of rows of the 2D string generated by the first node of the i-th heavy path, and Y_i' is the number of rows of the 2D string generated by the light child of the LCA found in the query
2. LCAs with left/right children. If we perform t queries of this type, the complexity for jth query is $O(\log X_j/X_j')$, where X_j is the number of columns of the 2D string generated by the first node of the j-th heavy path, and X_j' is the number of columns of the 2D string generated by the light child of the LCA found in the query.

The total access time is then

$$O\left(\sum_{i=1}^{l}\log Y_i/Y_i' + \sum_{j=1}^{t}\log X_j/X_j'\right).$$

Since each LCA expansion represents a 2D substring of the expansion of the previous LCA found, the following inequalities hold:

$$n \geqslant X_1 \geqslant X_1' \geqslant X_2 \geqslant X_2' \geqslant \ldots \geqslant X_t \geqslant X_t' \geqslant 1,$$

and

$$m \geqslant Y_1 \geqslant Y_1' \geqslant Y_2 \geqslant Y_2' \geqslant \ldots \geqslant Y_l \geqslant Y_l' \geqslant 1,$$

where we assume m and n the number of rows and the number of columns of $\mathcal{M}_{m\times n}$, respectively. For simplicity, in the following equalities m and n are also denoted as Y_0' and X_0', respectively.

We obtain the final time

$$\begin{aligned} O\left(\sum_{i=1}^{l}\log Y_i/Y_i' + \sum_{j=1}^{t}\log X_j/X_j'\right) &= O\left(\sum_{i=1}^{l}\log Y_{i-1}'/Y_i' + \sum_{j=1}^{t}\log X_{j-1}'/X_j'\right) \\ &= O(\log m + \log n) \\ &= O(\log mn). \end{aligned}$$

□

Therefore, the following corollary can be inferred.

Corollary 19 *Let $\mathcal{M}_{m\times n}$ be a 2D string. There exists a data structure that uses $O(g_{rl}\log mn)$ bits of space that supports direct access queries to any cell $\mathcal{M}[i][j]$ in $O(\log mn)$ time.*

6 Macro Schemes for 2D Strings

Navarro [2] introduced the measure b as the size of the smallest bidirectional macro scheme [25] for a 1D string. The notion of macro scheme can be generalized to two dimensions [11]: this leads to a generalization of the measure b to 2D strings.

Definition 12 A *2D macro scheme* for a string $\mathcal{M}_{m\times n}$ is any factorization of $\mathcal{M}_{m\times n}$ into a set of disjoint phrases such that any phrase is either a square of dimension 1×1 called an *explicit symbol/phrase*, or is a copied phrase with source in $\mathcal{M}_{m\times n}$ starting at a different position. For a 2D macro scheme to be *valid* or *decodable*, the function $\texttt{map} : ([1\,..\,m]\times[1\,..\,n])\cup\{\perp\}\rightarrow([1\,..\,m]\times[1\,..\,n])\cup\{\perp\}$ induced by the factorization must verify that:

i) $\texttt{map}(\perp)=\perp$, and if $\mathcal{M}[i][j]$ is an explicit symbol, then $\texttt{map}(i,j)=\perp$;
ii) for each copied phrase $\mathcal{M}[i_1\,..\,j_1][i_2\,..\,j_2]$, it must hold that $\texttt{map}(i_1+t_1, i_2+t_2) = \texttt{map}(i_1,i_2)+(t_1,t_2)$ for $(t_1,t_2)\in[0\,..\,j_1-i_1]\times[0\,..\,j_2-i_2]$, where $\texttt{map}(i_1,i_2)$ is the upper left corner of the source for $\mathcal{M}[i_1\,..\,j_1][i_2\,..\,j_2]$;
iii) for each $(i,j)\in[1\,..\,m]\times[1\,..\,n]$ there exists $k>0$ such that $\texttt{map}^k(i,j)=\perp$.

We define $b(\mathcal{M}_{m\times n})$ as the size of a smallest valid 2D macro scheme for $\mathcal{M}_{m\times n}$.

Example 20 Let I_n be the $n\times n$ identity matrix. A macro scheme for I_n consists of the phrases $\{X_1, X_2, X_3, X_4, X_5, X_6\}$ where: i) $X_1 = I_n[1][1]$ is an explicit symbol (the `1` in the top-left corner); ii) $X_2 = I_n[1][2]$ is an explicit symbol; $X_3 = I_n[2][1]$ is an explicit symbol; $X_4 = I_n[1][3\,..\,n]$ is a phrase with source $(1,2)$; $X_5 = I_n[3\,..\,n][1]$ is a phrase with source $(2,1)$; and $X_6 = I_n[2\,..\,n][2\,..\,n]$ is a phrase with source $(1,1)$. The underlying function `map` is defined as $\texttt{map}(1,1)=\texttt{map}(1,2)=\texttt{map}(2,1)=\perp$, $\texttt{map}(1,j)=(1,j-1)$ for $j\in[3\,..\,n]$, $\texttt{map}(i,1)=(i-1,1)$ for $i\in[3\,..\,n]$, and $\texttt{map}(i,j)=(i-1,j-1)$ for $i,j\in[2\,..\,n]\times[2\,..\,n]$. One can see that $\texttt{map}^n(i,j)=\perp$ for each i and j. Hence, the macro scheme is valid and $b(I_n)\leqslant 6$. Figure 6 shows this macro scheme for I_7.

The next proposition trivially shows that computing b for 2D strings is NP-complete. A similar statement was also in [11, Theorem 2.1].

Proposition 21 *The problem of determining if there exists a valid 2D macro scheme of size at most k for a 2D string $\mathcal{M}_{m\times n}$ is NP-complete.*

Proof The 1D version of the problem, which is known to be NP-complete [26], reduces to the 2D version of the problem in constant time. □

In the literature, several heuristic solution based on copy operations have been proposed to compress two-dimensional data, in both the lossless and lossy setting

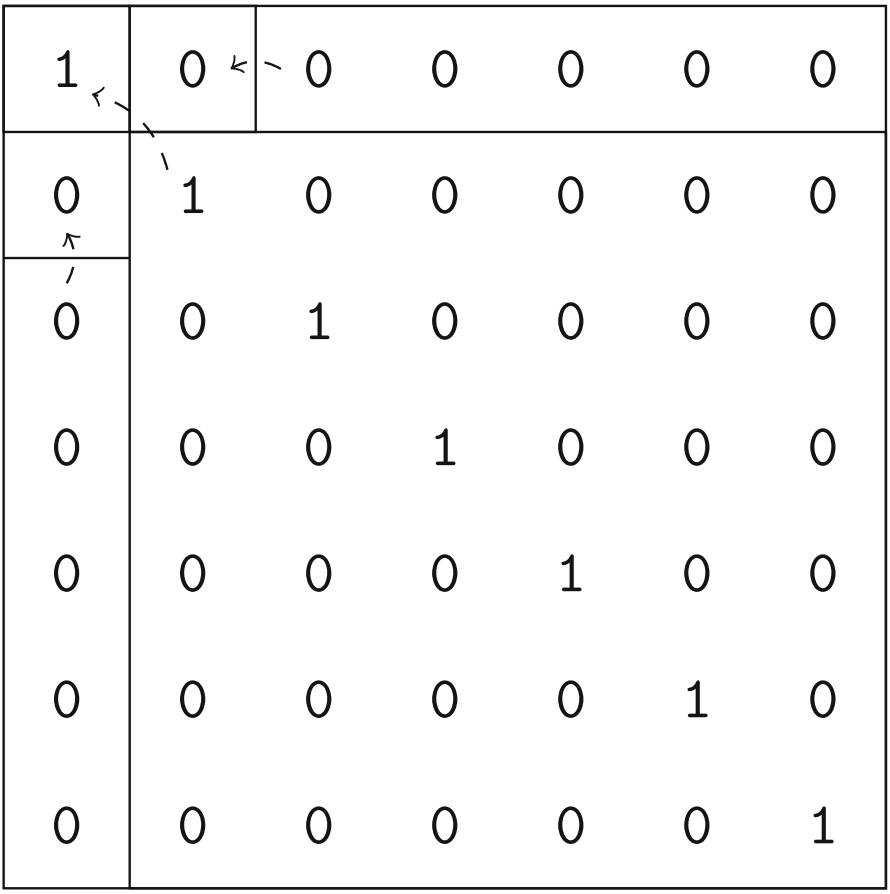

Fig. 6 Macro scheme with 6 phrases for I_7. The entries (1, 1), (1, 2), and (2, 1) are explicit symbols. The remaining phrases point to the source from where they are copied

(see [14] and the references therein). These methods can be seen as 2D macro schemes, but the number of phrases they emit can be asymptotically larger than b, see for example [11, Theorem 3.2].

Next we show that some relationships between the measures b, g_{rl} and g are preserved in the 2D context.

Proposition 22 *For every 2D string $\mathcal{M}_{m\times n}$ it holds that $b(\mathcal{M}_{m\times n}) \leqslant g_{rl}(\mathcal{M}_{m\times n})$.*

Proof We show how to construct a macro scheme from a 2D RLSLP, representing the same 2D string and having the same asymptotic size. Let G_{rl} be a 2D RLSLP generating $\mathcal{M}_{m\times n}$ and consider its grammar tree. Each leaf of the grammar tree corresponding to a variable that expands to a single symbol at cell $\mathcal{M}_{m\times n}[i][j]$, induces an explicit phrase of the parsing at that specific cell. A leaf of the grammar tree corresponding to an occurrence of the variable A expanding at cells $\mathcal{M}_{m\times n}[i_1 \,..\, i_2][j_1 \,..\, j_2]$ becomes a phrase of the parsing at these cells, and its source is aligned with the upper-left

corner of the expansion on $\mathcal{M}_{m\times n}$ of the primary occurrence of A in the grammar tree. For a leaf $①^{k-1}B$ induced by a horizontal run-length variable, which expands to $\mathcal{M}_{m\times n}[i_1..i_2][j_1..j_2]$ in $\mathcal{M}_{m\times n}$, we construct a phrase at these cells, pointing to the occurrence of $\exp(B)$ at its left in $\mathcal{M}_{m\times n}$. We do analogously, for 2D vertical run-length rules. This parsing is decodable and its size is bounded by the size g_{rl} of the grammar. □

A graphical representation of the proof is shown in Fig. 7.

In the case of 1D strings it is already known that for the Fibonacci words it holds $b = o(g_{rl})$ [13, 27]. The following result shows that the same result holds even for square 2D matrices.

Proposition 23 *There exists an infinite family of square 2D strings of size N where $g_{rl} = \Omega(b \log N)$.*

Proof Consider the identity matrix I_n of order n and a run-length 2D SLP G for I_n of size g. In the following we call run-length symbols the non-terminals of G corresponding to a run-length rule, and regular symbols the remaining non-terminals. We preliminarily prove that G can generate the 1s of I_n only by expanding regular symbols: consider without loss of generality a horizontal run-length rule $A \to ①^k B$ of G with $k > 1$, if $\exp(A)$ contains at least a 1 then the same must hold for $\exp(B)$ and therefore $\exp(A)$, and thus I_n, would contain at least $k > 1$ 1s in the same row which is impossible.

Given a non-terminal symbol A, we denote with #A the number of 1s in $\exp(A)$. Similarly to what we have done in the proof of Proposition 10 we consider the sequence

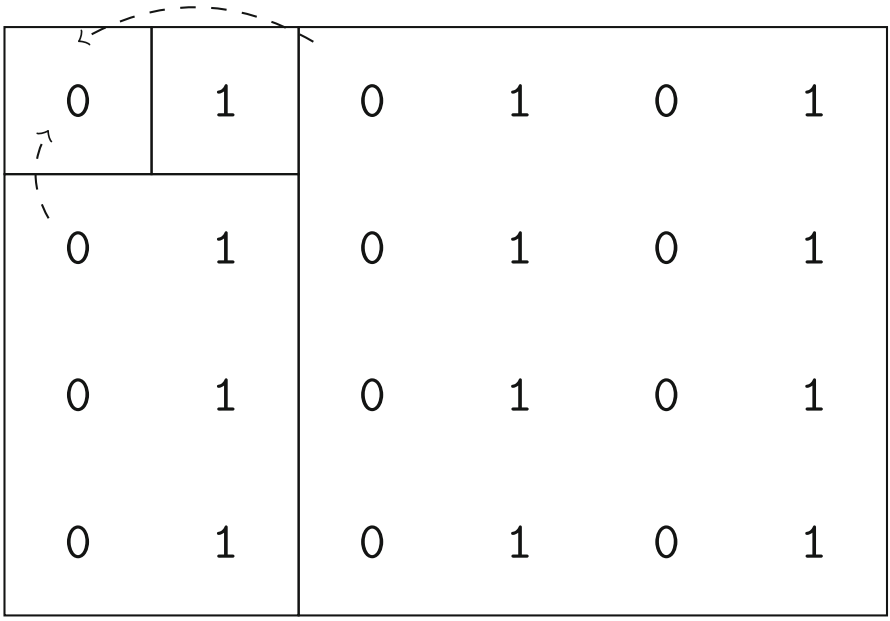

Fig. 7 Example of 2D macro scheme derived from the 2D RLSLP in Fig. 4

of non-terminals $\{A_i\}_{i=1\ldots k}$ defined as follows: A_1 is the starting symbol of G and A_{i+1} is a non-terminal symbol in the right-hand side α of the rule $A_i \to \alpha$ which maximizes the number of 1s in its expansion. Since $\exp(A_1) = I_n$ it is $\#A_1 = n$ and A_1 cannot be a run-length symbol, therefore it is $\#A_1 \leqslant 2\#A_2$ and A_2 cannot be a run-length symbol otherwise it would be $\#A2$ and $\#A_1 = 0$. By iterating this reasoning we inductively build a chain $n = \#A_1 \leqslant 2\#A_2 \leqslant \cdots \leqslant 2^{k-1}\#A_k$ of inequalities in which no run-length symbol appears. Similarly to what we observed in the proof of Proposition 10 it holds that A_k corresponds to a terminal rule $A_k \to 1$ with $k \leqslant g$. As a consequence, $n \leqslant 2^{k-1} < 2^g$ and therefore $g = \Omega(\log n)$. The results follows since $\log n = \Theta(\log N)$ and $b(I_n) = O(1)$ (see Example 20). □

In [10] the authors introduce a notion of 2D macro scheme for square strings that differs from the one in Definition 12 in that 1) phrases must all be square and 2) phrases are allowed to overlap (see [10, Section 4] for details). Given a 2D string $\mathcal{M}$, in the following we call $b_\square(\mathcal{M})$ the minimum number of phrases in a macro scheme with square, possibly overlapping, phrases. We note that the use of square phrases can significantly limit the power of a 2D macro scheme. In [11, Theorem 3.1], the authors exhibit a family of square matrices of size N for which $b_\square = \Omega(b\sqrt[4]{N})$ and therefore $b = o(b_\square)$. In the following, we show that the gap can be significantly larger.

For any $k > 0$, let D_k denote a binary de Bruijn sequence of length $n = 2^k + k - 1$ containing all the possible binary substrings of length k exactly once. We define the $n \times n$ matrix B_k over the alphabet $\Delta = \{\langle 0, 0\rangle, \langle 0, 1\rangle, \langle 1, 0\rangle, \langle 1, 1\rangle\}$ by the relationship

$$B_k[i][j] = \langle D_k[i], D_k[j]\rangle.$$

Notice that each row and each column of B_k is a de Bruijn sequence over a binary alphabet which is a subset of Δ. For example, if $D_k[i] = 1$, then row i of B_k is a de Bruijn sequence over the alphabet $\{\langle 1, 0\rangle, \langle 1, 1\rangle\}$. Similarly, if $D_k[j] = 0$, then column j of B_k is a de Bruijn sequence over the alphabet $\{\langle 0, 0\rangle, \langle 1, 0\rangle\}$. Notice that B_k contains only two distinct rows/columns since rows/columns i and j are different if and only if $D_k[i] \neq D_k[j]$. An example is shown in Fig. 8.

Lemma 24 *For any $k > 0$ it is $g(B_k) = O(n/\log n)$, where B_k has size $n \times n$, with $n = 2^k + k - 1$.*

Proof It is known that $g = O(n/\log_\sigma n)$ on any string [13, Lemma 12]. Hence, for binary de Bruijn sequences D_k, it holds $g(D_k) = O(n/\log n)$, that is, there exists an SLP G of size $O(n/\log n)$ generating D_k.

If in G we replace the terminal symbols $\{0, 1\}$ with $\{\langle 0, 0\rangle, \langle 0, 1\rangle\}$ we obtain an SLP G_0 that generates all rows i in B_k such that $D_k[i] = 0$. Similarly, if in G we replace the terminals $\{0, 1\}$ with $\{\langle 1, 0\rangle, \langle 1, 1\rangle\}$ we obtain an SLP G_1 that generates all rows i in B_k such that $D_k[i] = 1$. Finally, if in G we make all rules vertical and we replace the terminal 0 with the starting symbol of G_0 and the terminal 1 with the starting symbol of G_1, we obtain a 2D SLP G_2 that, combined with G_0 and G_1, generates the matrix B_k. The thesis follows since the size of $G_2 \cup G_1 \cup G_0$ is $O(n/\log n)$. □

Lemma 25 *Every square submatrix of size k or more appears in B_k at most once.*

	0	0	0	1	0	1	1	1	0	0
0	$\langle 0,0\rangle$	$\langle 0,0\rangle$	$\langle 0,0\rangle$	$\langle 0,1\rangle$	$\langle 0,0\rangle$	$\langle 0,1\rangle$	$\langle 0,1\rangle$	$\langle 0,1\rangle$	$\langle 0,0\rangle$	$\langle 0,0\rangle$
0	$\langle 0,0\rangle$	$\langle 0,0\rangle$	$\langle 0,0\rangle$	$\langle 0,1\rangle$	$\langle 0,0\rangle$	$\langle 0,1\rangle$	$\langle 0,1\rangle$	$\langle 0,1\rangle$	$\langle 0,0\rangle$	$\langle 0,0\rangle$
0	$\langle 0,0\rangle$	$\langle 0,0\rangle$	$\langle 0,0\rangle$	$\langle 0,1\rangle$	$\langle 0,0\rangle$	$\langle 0,1\rangle$	$\langle 0,1\rangle$	$\langle 0,1\rangle$	$\langle 0,0\rangle$	$\langle 0,0\rangle$
1	$\langle 1,0\rangle$	$\langle 1,0\rangle$	$\langle 1,0\rangle$	$\langle 1,1\rangle$	$\langle 1,0\rangle$	$\langle 1,1\rangle$	$\langle 1,1\rangle$	$\langle 1,1\rangle$	$\langle 1,0\rangle$	$\langle 1,0\rangle$
0	$\langle 0,0\rangle$	$\langle 0,0\rangle$	$\langle 0,0\rangle$	$\langle 0,1\rangle$	$\langle 0,0\rangle$	$\langle 0,1\rangle$	$\langle 0,1\rangle$	$\langle 0,1\rangle$	$\langle 0,0\rangle$	$\langle 0,0\rangle$
1	$\langle 1,0\rangle$	$\langle 1,0\rangle$	$\langle 1,0\rangle$	$\langle 1,1\rangle$	$\langle 1,0\rangle$	$\langle 1,1\rangle$	$\langle 1,1\rangle$	$\langle 1,1\rangle$	$\langle 1,0\rangle$	$\langle 1,0\rangle$
1	$\langle 1,0\rangle$	$\langle 1,0\rangle$	$\langle 1,0\rangle$	$\langle 1,1\rangle$	$\langle 1,0\rangle$	$\langle 1,1\rangle$	$\langle 1,1\rangle$	$\langle 1,1\rangle$	$\langle 1,0\rangle$	$\langle 1,0\rangle$
1	$\langle 1,0\rangle$	$\langle 1,0\rangle$	$\langle 1,0\rangle$	$\langle 1,1\rangle$	$\langle 1,0\rangle$	$\langle 1,1\rangle$	$\langle 1,1\rangle$	$\langle 1,1\rangle$	$\langle 1,0\rangle$	$\langle 1,0\rangle$
0	$\langle 0,0\rangle$	$\langle 0,0\rangle$	$\langle 0,0\rangle$	$\langle 0,1\rangle$	$\langle 0,0\rangle$	$\langle 0,1\rangle$	$\langle 0,1\rangle$	$\langle 0,1\rangle$	$\langle 0,0\rangle$	$\langle 0,0\rangle$
0	$\langle 0,0\rangle$	$\langle 0,0\rangle$	$\langle 0,0\rangle$	$\langle 0,1\rangle$	$\langle 0,0\rangle$	$\langle 0,1\rangle$	$\langle 0,1\rangle$	$\langle 0,1\rangle$	$\langle 0,0\rangle$	$\langle 0,0\rangle$

Fig. 8 Example of 2D string B_3 on the alphabet $\{\langle 0,0\rangle, \langle 0,1\rangle, \langle 1,0\rangle, \langle 1,1\rangle\}$. It is based on the de Bruijn word $D_3 = \texttt{0001011100}$, displayed horizontally and vertically respectively above and on the left of B_3

Proof It suffices to prove the result for the square submatrices of size k. Assume that the $k \times k$ submatrix with top-left corner in (i, j) is identical to the submatrix with top-left corner in (u, v). The crucial observation is that $B_k[i][j] = B_k[u][v]$ implies $D_k[i] = D_k[u]$ and $D_k[j] = D_k[v]$. Considering also the other entries, we get that if the two submatrices are equal then we must have $D_k[i..i+k-1] = D_k[u..u+k-1]$ and $D_k[j..j+k-1] = D_k[v..v+k-1]$. Since D_k is a de Bruijn sequence, this implies $i = u$ and $j = v$ as claimed. □

Proposition 26 *For the 2D string B_k of size $N = n \times n$, with $n = 2^k + k - 1$, it holds that $b_\square = \Omega(b\sqrt{N}/\log N) = \Omega(g\sqrt{N}/\log N)$.*

Proof Since $b \leqslant g$, by Lemma 24 $b = O(\sqrt{N}/\log N)$. Lemma 25 implies that there cannot be square phrases of size $k \times k$ or larger. Hence, the number of phrases is at least n^2/k^2, so $b_\square = \Omega(N/\log^2 N)$ and the lemma follows. □

7 On the Relative Power of 2D Measures

A remarkable property of the measures considered in this paper is that, for 1D strings, they can be totally ordered in terms of their relative power at capturing regularities in the input; indeed, for any 1D string S it is $\delta(S) \leqslant \gamma(S) \leqslant b(S) \leqslant g_{rl}(S) \leqslant g(S)$ (see [1, 2]). In the previous sections, we have shown that when we consider also 2D strings the relationships $\delta \leqslant \gamma$ and $b \leqslant g_{rl} \leqslant g$ still hold. In this section, however, we prove that the two classes of measures, i.e. δ and γ from one side, based on the counting and distribution of distinct factors, and b, g_{rl}, and g from the other side, based on

copy-paste mechanisms, become incomparable when also 2D strings are considered. In particular g can be asymptotically smaller than δ:

Proposition 27 *There exists an infinite family of 2D strings of size N with $\delta = \Omega(gN/\log^3 N)$.*

Proof For $k \geqslant 1$, consider the 2D binary string E_k of size $N = k \times 2^k$ such that, for all $1 \leqslant i \leqslant 2^k$, the ith column of E_k is the binary representation of $i-1$ in k digits (with the top row containing the least significant bits). Since all columns of E_k are distinct, E_k contains 2^k distinct factors of size $k \times 1$ and therefore it is $\delta(E_k) \geqslant 2^k/k = kN/k^3$. We prove that $g(E_k) = O(k)$ by exhibiting a 2D SLP for E_k of size $O(k)$ and the result immediately follows because $k < \log N$.

Consider the 2D SLP $G_k = (V_k, \{\mathtt{0}, \mathtt{1}\}, R'_k, S_k)$ having the following rules R'_k:

- $X_0 \to \mathtt{0}$ and $X_h \to X_{h-1} \oslash X_{h-1}$ for all $1 \leqslant h \leqslant k-1$;
- $Y_0 \to \mathtt{1}$ and $Y_h \to Y_{h-1} \oslash Y_{h-1}$ for all $1 \leqslant h \leqslant k-1$;
- $C_h \to X_{h-1} \oslash Y_{h-1}$ for all $2 \leqslant h \leqslant k$;
- $S_1 = X_0 \oslash Y_0$ and $S_h \to R_h \ominus C_h$ for all $2 \leqslant h \leqslant k$.
- $R_h \to S_{h-1} \oslash S_{h-1}$ for all $2 \leqslant h \leqslant k$;

G_k has size $\Theta(k)$ and it is easy to see that $\exp(X_h) = \mathtt{0}^{(2^h)}$, $\exp(Y_h) = \mathtt{1}^{(2^h)}$ and therefore $\exp(C_h) = \mathtt{0}^{(2^{h-1})}\mathtt{1}^{(2^{h-1})}$. In the following, we show by induction on k that G_k is a 2D SLP for E_k, that is $\exp(S_k) = E_k$. For the base case $k = 1$ it is $\exp(S_1) = \exp(X_0) \oslash \exp(Y_0) = \mathtt{0} \oslash \mathtt{1} = E_1$. For the inductive step we assume that $\exp(S_k) = E_k$ and we note that for $k \geqslant 1$ the bottom row $E_{k+1}[k+1][1..2^{k+1}]$ of E_{k+1} is the string $\mathtt{0}^{(2^k)}\mathtt{1}^{(2^k)} = \exp(C_{k+1})$ and the remaining rectangular submatrix $E_{k+1}[1..k][1..2^{k+1}]$ is $E_k \oslash E_k$.

By taking two expansion steps starting from S_{k+1}, we obtain $\exp(S_{k+1}) = (\exp(S_k) \oslash \exp(S_k)) \ominus \exp(C_{k+1})$ which by inductive hypotheses and the definition of C_{k+1} expands to $(E_k \oslash E_k) \ominus E_{k+1}[k+1][1..2^{k+1}] = E_{k+1}$. □

From the above proposition, we get that the measure g (and therefore also g_{rl} and b) can be much smaller than both δ and γ. Intuitively, the reason is that the matrix E_k is hard to compress by columns (they are all distinct) but easily compressible by rows. The measure g is defined in terms of the best grammar compressor, so it fully exploits row compressibility. In contrast, the measure δ is based on occurrences of factors of any shape and is therefore "hindered" by the difficulty of compressing columns. For 1D strings it is always $\delta \leqslant g$, but in the 2D setting, because of the greater freedom in choosing the shape of the copied patterns, the measures b, g_{rl} and g become mutually incomparable with both γ and δ.

Given the above observation, it is worthwhile to compare g with the measures $\delta_\square$ and $\gamma_\square$ whose definitions are based on square factors. In some sense, using square factors can be seen as a method to capture both horizontal and vertical compressibility. Unfortunately, the following proposition shows that g can be asymptotically smaller than $\delta_\square$.

Proposition 28 *There exists a family of rectangular 2D strings of size N with $\delta_\square = \Omega(gN/\log^4 N)$.*

Proof Consider again the binary matrix E_k of size $N = k \times 2^k$ of Proposition 27 having $g(E_k) = O(k)$. We note that any $k \times k$ submatrix of E_k is a distinct factor of size k^2 and therefore it is $\delta_{\square}(E_k) \geqslant (2^k - k + 1)/k^2 = \Omega(kN/k^4)$. The result immediately follows since $g(E_k) = O(k)$ and $k < \log N$. □

The previous two propositions are based on rectangular matrices with a number of columns that is exponentially larger than the number of rows. Since measure $\delta_{\square}$ was originally proposed only for square input matrices, one may wonder whether the gap is due to the highly skewed shape of the input matrix. The next result shows that this is not the case in the sense that for input square matrices we have a smaller, but still significant, gap between g and $\delta_{\square}$. Note that the same result also holds for $\gamma_{\square}$, δ, and γ, since $\delta_{\square} \leqslant \gamma_{\square}$ and $\delta_{\square} \leqslant \delta \leqslant \gamma$.

Proposition 29 *There exists an infinite family of square 2D strings of size N with $\delta_{\square} = \Omega(g\sqrt{N}/\log N)$.*

Proof Consider the matrix B_k defined in Section 6.

By Lemma 24 it is $g(B_k) = O(n/\log n) = O(\sqrt{N}/\log N)$. By Lemma 25 B_k contains $\Theta(n^2) = \Theta(N)$ distinct $k \times k$ factors, hence $\delta_{\square} = \Omega(N/\log^2 N)$ and the bound follows. □

Note that Propositions 26, 27 and 29 show that in the 2D setting there can be a significant gap between b, g_{rl} and g from one side, and $b_{\square}$, $\delta_{\square}$, $\gamma_{\square}$, δ, γ from the other. Furthermore, since the square matrix $0_{n\times n}$ consisting of only zeros has constant measures $b_{\square}$, $\delta_{\square}$, $\gamma_{\square}$ but it is $g(0_{n\times n}) = \Omega(\log n)$ (see Proposition 10), Propositions 26 and 29 imply that g is also mutually incomparable with each of the above square measures, even if we consider only square input matrices. Moreover g_{rl} is incomparable with both $\delta_{\square}$ and $\gamma_{\square}$ since for the identity matrix I_n it is $\delta_{\square} \leqslant \gamma_{\square} = O(1)$ by Example 9 and $g_{rl} = \Omega(\log n)$ by Proposition 23.

We now show that even the recently introduced 2D Block Tree data structure [6], which is also based on square factors, can fail to capture the regularity of certain two-dimensional strings. The 2D Block Tree is a tree-like compressed representation of a square matrix supporting random access to individual entries in logarithmic time. Given an $n \times n$ input matrix $\mathcal{M}$, an integer parameter $c > 1$, and assuming that n is a power of c, the root of the 2D Block tree at level $\ell = 0$ represents the whole matrix $\mathcal{M}$. To build the level $\ell \geqslant 1$ of the tree we recursively partition (some of) the submatrices represented at level $\ell - 1$ into c^2 smaller non-overlapping submatrices of size $n/c^\ell \times n/c^\ell$ called *blocks*; for each of these blocks, the tree stores a corresponding descending node at level ℓ. The 2D Block Tree attempts to compress the input matrix by avoiding the storage of redundant submatrices: if a block has a prior occurrence in row-major order (RMO), its corresponding subtree is candidate to be pruned and replaced with $O(1)$ pointers to the nodes overlapping its first occurrence in RMO. The pruned blocks are not partitioned into smaller matrices, and their corresponding nodes are leaves in the 2D block tree. See [6, 10] for details.

Unfortunately, the following example exhibits a family of 2D strings that are significantly more compressible when represented as an SLP compared to their 2D Block Tree representation: for these matrices, the 2D Block Tree fails to achieve a compression close to the measure g (and therefore to g_{rl} and b).

Proposition 30 *There exists an infinite family of square 2D strings of size N such that the number of nodes of their 2D Block Tree is $\Omega(g\sqrt{N}/\log N)$. The same result holds also for the attractor-based 2D Block Tree defined in [10, Theorem 5].*

Proof We prove the result by showing that all the above variants of the 2D Block Tree, built on the matrix B_k defined in Section 6, contains $\Omega(N/\log^2 N)$ nodes. The proposition follows since by Lemma 24 it is $g(B_k) = O(\sqrt{N}/\log N)$. By Lemma 25 until we reach the tree level in which the blocks are smaller than $k \times k$, all blocks are first occurrences, and therefore the corresponding tree nodes cannot be pruned. Hence, the 2D Block Tree nodes are $\Omega(n^2/k^2) = \Omega(N/\log^2 N)$. Note that the results hold even considering the variant described in [10, Theorem 4] in which the number of nodes at the first level is $\Theta(\delta_\square(B_k))$, since, as shown in the proof of Proposition 29, it is $\delta_\square(B_k) = \Omega(N/\log^2 N)$.

For the attractor-based 2D Block Tree we note that in each level we mark the at least $\gamma_\square$ nodes corresponding to blocks including an attractor position, and the result follows since $\gamma_\square \geqslant \delta_\square$. Similarly, the result holds also for the variant described in [10, Theorem 5], since in that variant there are $\Theta(\gamma_\square(B_k)) = \Omega(\delta_\square(B_k))$ nodes at the first level. □

8 Effectiveness of Linearization Techniques

A classical heuristic for compressing 2D strings is to transform a matrix $\mathcal{M}_{m\times n}$ into a 1D string S and use a one-dimensional compressor on S. Having generalized 1D measures to 2D strings, it is natural to measure the effectiveness of linearization techniques by comparing, for a given measure μ, the values $\mu(\mathcal{M}_{m\times n})$ and $\mu(S)$. Clearly, for each matrix, there exists a linearization that makes the 2D string highly compressible: we can visit in order from left to right and from top to bottom all the occurrences of $a_1 \in \Sigma$, followed by all the occurrences of $a_2 \in \Sigma$, and so on, obtaining a string consisting in $|\Sigma|$ equal-letter runs. However, this method requires an ad-hoc linearization for each matrix which may require substantial additional information to retrieve the original input. It is therefore customary in the literature to consider linearization techniques that can be inverted efficiently in terms of both time and space.

The simplest linearization technique consists of mapping a matrix to the string obtained by concatenating its rows, as formally defined below.

Definition 13 The *row-linearization* is the map

$$\texttt{rlin} : \bigcup_{m,n>0} \Sigma^{m\times n} \mapsto \Sigma^*$$

such that $\texttt{rlin}(\mathcal{M}_{m\times n}) = \bigodot_{i=1}^{i=m} \mathcal{M}[i][1..n] = \mathcal{M}[1][1..n]\cdots\mathcal{M}[m][1..n]$.

The (lack of) effectiveness of `rlin` with respect to grammar compression has been already shown in [16, Theorem 2.2] with an example of a matrix T_n of size

$(2^{n+1}-1)\times(2^n+1)2^n$ such that $g(T_n)=O(n)$, while $g(\mathtt{rlin}(T_n))=\Omega(2^n)$. The following example shows a similar result for the measure δ.

Example 31 Let $\mathcal{M}_{n\times n}$ be obtained by appending to the identity matrix I_{n-1} a row of 0's at the bottom, and then a column of 1's at the right, as shown in Fig. 9. For each k_1, k_2, $P_{\mathcal{M}}(k_1,k_2)$ is at most $3(k_1+k_2)$. We can see this by considering three cases: the submatrices that do not intersect the last row or column, the submatrices intersecting the last row, and the submatrices intersecting the last column. In each case, the distinct submatrices are associated to where the diagonal of 1's intersects a submatrix (if it does so). This can happen in at most k_1+k_2 different ways. As $3(k_1+k_2)/k_1k_2 \leqslant 6$, we obtain $\delta(\mathcal{M}_{n\times n}) = O(1)$. On the other hand, for each $k\in[0\,..\,n-2]$ and $i\in[0\,..\,n-k-2]$, each factor $0^i10^k10^{n-k-i-2}$ appears in $\mathtt{rlin}(\mathcal{M}_{n\times n})$. There are $n-k-1$ of these factors for each k. Summing over all k, we obtain $P_{\mathcal{M}}(n)/n \geqslant (n-1)/2=\Omega(n)$. Thus, $\delta(\mathtt{rlin}(\mathcal{M}_{n\times n}))=\Omega(n)$.

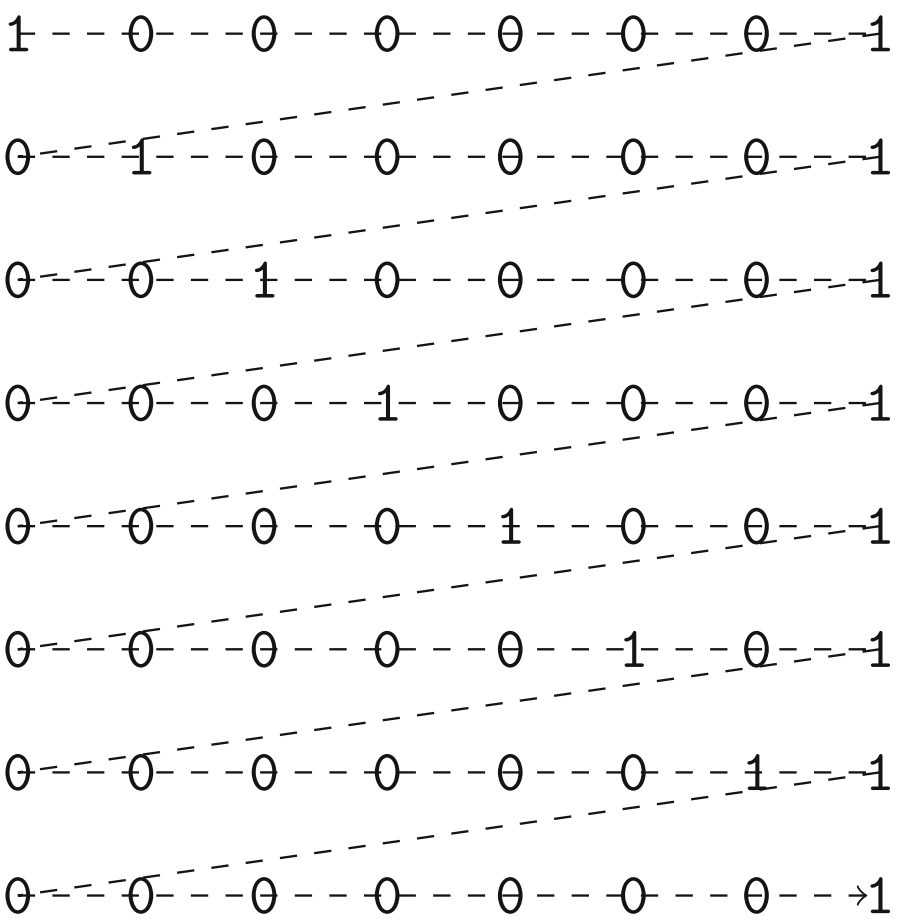

Fig. 9 Example of the 2D string $\mathcal{M}_{8\times 8}$ from Example 31. The row-linearization is spelled by following the dashed arrow starting from the top-left corner

Somewhat surprisingly, in some settings, the linearized matrix has a smaller measure. The following example shows a family of 2D strings E_k for which $\gamma(\texttt{rlin}(E_k))$ is asymptotically smaller than $\gamma(E_k)$.

Example 32 Consider the matrix E_k having size $N = k \times 2^k$ of Proposition 27. We note that the i-th row of E_k is the periodic string $(\texttt{0}^{(2^{i-1})}\texttt{1}^{(2^{i-1})})^{(2^{k-i})}$ and therefore $\texttt{rlin}(E_k) = \bigodot_{i=1}^{i=k}(\texttt{0}^{(2^{i-1})}\texttt{1}^{(2^{i-1})})^{(2^{k-i})}$. We define the set $A = \bigcup_{i=1}^{i=k}\{(i-1)2^k + 1, (i-1)2^k + 1 + 2^{i-1}, (i-1)2^k + 2^i\}$, that is the set of positions of $\texttt{rlin}(E_k)$ where respectively the first `0` and the first/last `1` of the leftmost occurrence of $\texttt{0}^{(2^{i-1})}\texttt{1}^{(2^{i-1})}$ in row i are mapped during the linearization. We claim that A is an attractor for $\texttt{rlin}(E_k)$. If a substring S of $\texttt{rlin}(E_k)$ spans more than one row of E_k, it includes a `0` from the first column of E_k, and therefore it crosses a position of A. Otherwise S is a substring of the ith row and since the rows of E_k are periodic, the leftmost occurrence S' of S starts inside the first group of $\texttt{0}^{(2^{i-1})}\texttt{1}^{(2^{i-1})}$ i.e. in $E_k[i][1..2^i]$. Suppose that S' does not include any attractor position. Then, S' has to be shorter than the maximum distance between two adjacent attractor positions in the same row i.e. it must be $l = |S'| \leqslant 2^{i-1} - 1$, and therefore $S' = \texttt{0}^a\texttt{1}^{l-a}$ or $S' = \texttt{1}^a\texttt{0}^{l-a}$ for some $0 \leqslant a < l$ because S' cannot overlap two distinct groups of `1`'s or `0`'s. If $a = 0$ then S' must include respectively the first `1` or the first `0` in the run, otherwise it must include respectively the first or the last `1`, therefore we conclude that A is an attractor for $\texttt{rlin}(E_k)$ (Fig. 10).

Since A has size $3k - 1$, it is $\gamma(\texttt{rlin}(E_k)) = O(k)$, on the other hand since each column of E_k is a distinct non-overlapping $k \times 1$ factor it is $\gamma(E_k) \geqslant 2^k$.

Another well-known linearization technique uses a plane-filling curve, such as the Peano-Hilbert curve. Unlike the linearization discussed above, this is defined on square matrices whose dimensions are a power of 2. For such a definition, the notion of upper left (UL), upper right (UR), lower left (LL), and lower right (LR) quadrants is used [28]. More formally, given the matrix $\mathcal{M} \in \Sigma^{2^i \times 2^i}$, $UL(\mathcal{M}) = \mathcal{M}[1..2^{i-1}][1..2^{i-1}]$, $UR(\mathcal{M}) = \mathcal{M}[1..2^{i-1}][2^{i-1}+1..2^i]$, $LL(\mathcal{M}) = \mathcal{M}[2^{i-1}+1..2^i][1..2^{i-1}]$, $LR(\mathcal{M}) = \mathcal{M}[2^{i-1}+1..2^i][2^{i-1}+1..2^i]$.

Definition 14 The *left, right, up and down scans*, denoted by `ls`, `rs`, `us`, and `ds`, respectively, are the maps
$\bigcup_{i \geqslant 0} \Sigma^{2^i \times 2^i} \to \Sigma^*$ defined recursively as follows:

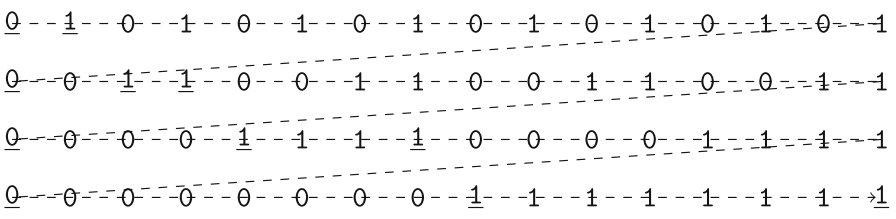

Fig. 10 Example of the 2D string E_4 from Example 32. The row-linearization is spelled by following the dashed arrow starting from the top-left corner. The underlined symbols correspond to the positions in the string attractor A of $\texttt{rlin}(E_4)$

- $\mathtt{ls}(\mathcal{M}) = \mathtt{rs}(\mathcal{M}) = \mathtt{us}(\mathcal{M}) = \mathtt{ds}(\mathcal{M}) = \mathcal{M}$, if $|\mathcal{M}| = 1$;
- $\mathtt{rs}(\mathcal{M}) = \mathtt{ds}(UL(\mathcal{M}))\cdot\mathtt{rs}(UR(\mathcal{M}))\cdot\mathtt{rs}(LR(\mathcal{M}))\cdot\mathtt{us}(LL(\mathcal{M}))$, if $|\mathcal{M}| > 1$;
- $\mathtt{ds}(\mathcal{M}) = \mathtt{rs}(UL(\mathcal{M}))\cdot\mathtt{ds}(LL(\mathcal{M}))\cdot\mathtt{ds}(LR(\mathcal{M}))\cdot\mathtt{ls}(UR(\mathcal{M}))$, if $|\mathcal{M}| > 1$;
- $\mathtt{us}(\mathcal{M}) = \mathtt{ls}(LR(\mathcal{M}))\cdot\mathtt{us}(UR(\mathcal{M}))\cdot\mathtt{us}(UL(\mathcal{M}))\cdot\mathtt{rs}(LL(\mathcal{M}))$, if $|\mathcal{M}| > 1$;
- $\mathtt{ls}(\mathcal{M}) = \mathtt{us}(LR(\mathcal{M}))\cdot\mathtt{ls}(LL(\mathcal{M}))\cdot\mathtt{ls}(UL(\mathcal{M}))\cdot\mathtt{ds}(UR(\mathcal{M}))$, if $|\mathcal{M}| > 1$.

The *Peano-Hilbert-linearization* is the map

$$\mathtt{phlin} : \bigcup_{i\geqslant 0} \Sigma^{2^i\times 2^i} \to \Sigma^*$$

such that, for each matrix $\mathcal{M}$ of size $2^i \times 2^i$,

- $\mathtt{phlin}(\mathcal{M}) = \mathtt{rs}(\mathcal{M})$ if i is odd;
- $\mathtt{phlin}(\mathcal{M}) = \mathtt{ds}(\mathcal{M})$ if i is even.

From the definition of phlin, it easily follows that each quadrant of a matrix $\mathcal{M}_{2^i\times 2^i}$ is completely visited before moving to the following. An example of Peano-Hilbert linearization of the identity matrix I_{2^k}, with $k = 3$, is shown in Fig. 11.

The following two lemmas show that the right and down scans produce the same string when applied to the identity matrix (Lemma 33) and provide a characterization of the string $\mathtt{phlin}(I_{2^k})$ (Lemma 34).

Lemma 33 *Let us denote by I_{2^k} the identity matrix of size $2^k \times 2^k$. Then, $\mathtt{ds}(I_{2^k}) = \mathtt{rs}(I_{2^k})$, for all $k \geqslant 0$.*

Proof We provide a proof by induction. For $k = 0$, clearly $\mathtt{ds}(I_{2^0}) = \mathtt{rs}(I_{2^0}) = 1$. For the inductive step, assume that $\mathtt{ds}(I_{2^j}) = \mathtt{rs}(I_{2^j})$ for all $j \leqslant k$. For $t \geqslant 0$, let 0_{2^t} denote the matrix of size $2^t \times 2^t$ containing only 0's. Observe that $\mathtt{ds}(0_{2^t}) = \mathtt{rs}(0_{2^t}) = \mathtt{ls}(0_{2^t}) = \mathtt{us}(0_{2^t}) = 0^{4^t}$, for all $t \geqslant 0$. Then,

$$\begin{aligned}
\mathtt{ds}(I_{2^{k+1}}) &= \mathtt{rs}(UL(I_{2^{k+1}}))\cdot\mathtt{ds}(LL(I_{2^{k+1}}))\cdot\mathtt{ds}(LR(I_{2^{k+1}}))\cdot\mathtt{ls}(UR(I_{2^{k+1}}))\\
&= \mathtt{rs}(I_{2^k})\cdot\mathtt{ds}(0_{2^k})\cdot\mathtt{ds}(I_{2^k})\cdot\mathtt{ls}(0_{2^k})\\
&= \mathtt{ds}(I_{2^k})\cdot\mathtt{rs}(0_{2^k})\cdot\mathtt{rs}(I_{2^k})\cdot\mathtt{us}(0_{2^k})\\
&= \mathtt{ds}(UL(I_{2^{k+1}}))\cdot\mathtt{rs}(UR(I_{2^{k+1}}))\cdot\mathtt{rs}(LR(I_{2^{k+1}}))\cdot\mathtt{us}(LL(I_{2^{k+1}}))\\
&= \mathtt{rs}(I_{2^{k+1}}),
\end{aligned}$$

and the thesis follows. □

Lemma 34 *For the identity matrix I_{2^k} of size $2^k \times 2^k$, with $k \geqslant 1$ it holds that,*

$$\mathtt{phlin}(I_{2^k}) \;=\; \mathtt{phlin}(I_{2^{k-1}})\,0^{4^{k-1}}\,\mathtt{phlin}(I_{2^{k-1}})\,0^{4^{k-1}}.$$

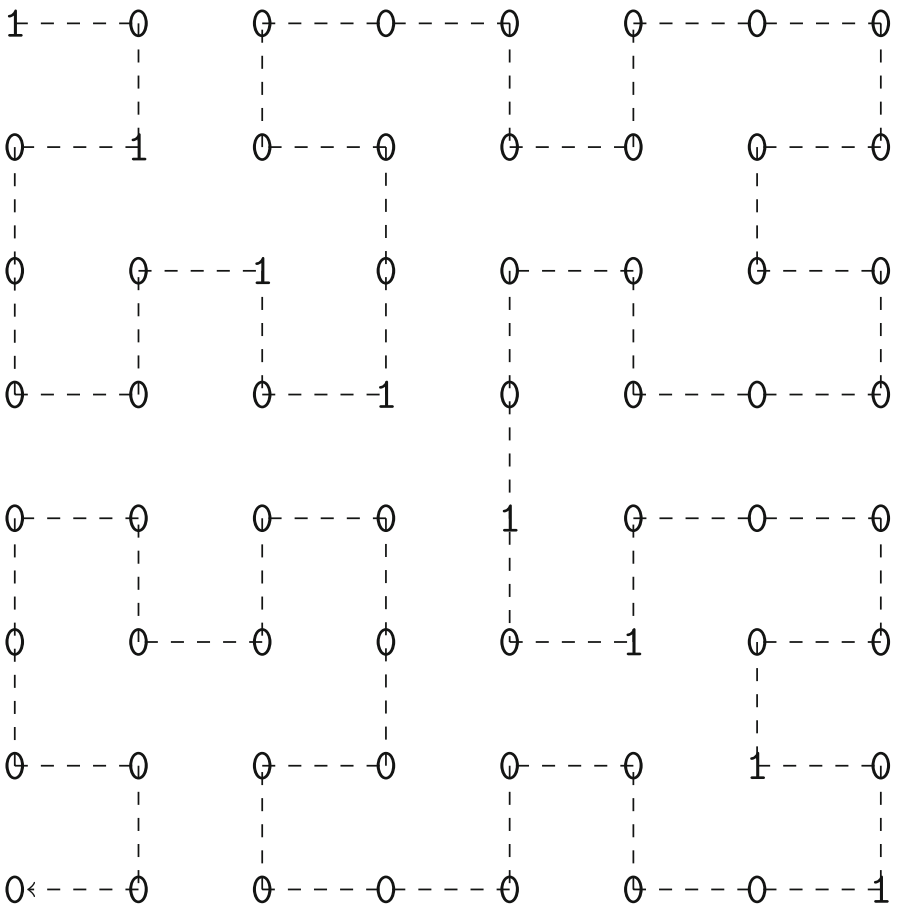

Fig. 11 Example of the 2D string I_8. The Peano-Hilbert linearization is spelled by following the dashed arrow starting from the top-left corner. The string $\texttt{phlin}(I_8)$ is obtained as $\texttt{rs}(I_8) = \texttt{ds}(UL(I_8)) \cdot \texttt{rs}(UR(I_8)) \cdot \texttt{rs}(LR(I_8)) \cdot \texttt{us}(LL(I_8)) = \texttt{phlin}(I_4) \cdot 0^{16} \cdot \texttt{phlin}(I_4) \cdot 0^{16}$

Proof By Lemma 33, $\texttt{phlin}(I_{2^k}) = \texttt{rs}(I_{2^k}) = \texttt{ds}(I_{2^k})$ independently of the value of k, therefore for every $k \geqslant 1$ it holds that

$$\begin{aligned}
\texttt{phlin}(I_{2^k}) &= \texttt{rs}(I_{2^k}) \\
&= \texttt{ds}(UL(I_{2^k})) \cdot \texttt{rs}(UR(I_{2^k})) \cdot \texttt{rs}(LR(I_{2^k})) \cdot \texttt{us}(LL(I_{2^k})) \\
&= \texttt{ds}(I_{2^{k-1}}) \cdot \texttt{rs}(0_{2^{k-1}}) \cdot \texttt{rs}(I_{2^{k-1}}) \cdot \texttt{us}(0_{2^{k-1}}) \\
&= \texttt{phlin}(I_{2^{k-1}}) \cdot 0^{4^{k-1}} \cdot \texttt{phlin}(I_{2^{k-1}}) \cdot 0^{4^{k-1}}
\end{aligned}$$

□

Lempel and Ziv [28, Lemma 1] showed that phlin is effective for compressing 2D strings using a finite-state encoder. In the next proposition, we show that there

exist matrices with a 2D bidirectional macro scheme that is smaller by a logarithmic factor compared to the optimal scheme of their $\texttt{phlin}$ linearized form.

Proposition 35 *It holds that* $b(I_{2^k}) = O(1)$ *and* $b(\texttt{phlin}(I_{2^k})) = \Omega(k)$.

Proof We have already observed in Example 20 that $b(I_n) = O(1)$ for any $n > 0$. To prove the second bound, for $\ell \geqslant 1$ define $t_\ell = \sum_{i=0}^{\ell-1} 4^i$. We preliminarily prove by induction on k that: *a)* $\texttt{phlin}(I_{2^k})$ starts with $\texttt{1}$, *b)* $\texttt{phlin}(I_{2^k})$ ends with $\texttt{10}^{t_k}$, *c)* $\texttt{phlin}(I_{2^k})$ contains the substrings $\texttt{10}^{t_\ell}\texttt{1}$ for $\ell = 1, \ldots, k$.

For $k = 1$, $\texttt{phlin}(I_2) = \texttt{1010}$ satisfies all conditions. For the inductive step *a)* is trivial. By Lemma 34, $\texttt{phlin}(I_{2^{k+1}})$ ends with $\texttt{phlin}(I_{2^k})\texttt{0}^{4^k}$, then *b)* is immediate. To prove *c)* observe that, by Lemma 34, $\texttt{phlin}(I_{2^{k+1}})$ contains $\texttt{phlin}(I_{2^k})$. Then, by induction it contains all substrings $\texttt{10}^{t_\ell}\texttt{1}$ for $\ell = 1, \ldots, k$. To prove that it also contains $\texttt{10}^{t_{k+1}}\texttt{1}$ observe that by *a)* $\texttt{phlin}(I_{2^{k+1}})$ starts with $\texttt{phlin}(I_{2^k})\texttt{0}^{4^k}\texttt{1}$ and by *b)* this string ends with $\texttt{10}^{t_k}\texttt{0}^{4^k}\texttt{1} = \texttt{10}^{t_{k+1}}\texttt{1}$, and therefore is a substring of $\texttt{phlin}(I_{2^{k+1}})$.

Having established that $\texttt{phlin}(I_{2^k})$ contains the *distinct* substrings $\texttt{10}^{t_\ell}\texttt{1}$ for $\ell = 1, \ldots, k$, since a single string position can be contained in at most two such substrings, we conclude that $b(\texttt{phlin}(I_{2^k})) \geqslant \gamma(\texttt{phlin}(I_{2^k})) = \Omega(k)$. □

9 Generalization to Multidimensional Strings

In the previous sections, we focused on extending repetitiveness measures of strings to the two-dimensional context. Here, we outline how the definitions of the measures δ, γ, g_{rl}, g, and b, and some of their properties, can be naturally generalized to d-dimensional strings, for every $d > 0$. Note that the measures $\gamma_\square$, $\delta_\square$ have been generalized to 3D strings in [10], where it is shown that in 3D it is still $\delta_\square \leqslant \gamma_\square$ and that the gap between them can be larger than in 2D.

Given a tuple of d integers $\mathbf{n} = (n_1, \ldots, n_d)$, with $n_i > 0$ for all $i \in [1\,..\,d]$, a dD *string* $\mathcal{M}_\mathbf{n}$ is a multidimensional array of elements from Σ of size $N = n_1 \times \overset{d}{\cdots} \times n_d$, where n_i is the size over the ith dimension. Every element of $\mathcal{M}_\mathbf{n}$ can be accessed by $\mathcal{M}_\mathbf{n}[\mathbf{j}] \in \Sigma$, where $\mathbf{j}$ is a d-tuple $\mathbf{j} = (j_1, \ldots, j_d)$, with $1 \leqslant j_i \leqslant n_i$ for all $i \in [1\,..\,d]$, or alternatively using the notation $\mathcal{M}_\mathbf{n}[j_1]..[j_d]$.

Given two positions $\mathbf{i}, \mathbf{j}$ of $\mathcal{M}_\mathbf{n}$, we denote by $\mathcal{M}_\mathbf{n}[\mathbf{i}\,..\,\mathbf{j}]$ the dD substring starting at position $\mathbf{i}$ and ending at position $\mathbf{j}$, that is $\mathcal{M}[i_1\,..\,j_1][i_2\,..\,j_2]\cdots[i_d\,..\,j_d]$. Given two dD strings $\mathcal{M}_\mathbf{n}$ and $\mathcal{M}'_\mathbf{m}$, for some $\mathbf{n} = (n_1, \ldots, n_d)$ and $\mathbf{m} = (m_1, \ldots, m_d)$, the concatenation over the kth dimension of $\mathcal{M}_\mathbf{n}$ and $\mathcal{M}'_\mathbf{m}$, denoted by $\mathcal{M}_\mathbf{n} \oslash_{(k)} \mathcal{M}'_\mathbf{m}$, is a partial operation that can be performed only if $n_j = m_j$ for all $j \neq k$. This operation produces a dD string with the same size of $\mathcal{M}_\mathbf{n}$ and $\mathcal{M}'_\mathbf{m}$ along every direction except along the kth one where the resulting size becomes $n_k + m_k$.

Formally, the resulting matrix is defined as follows: given a d-tuple $\mathbf{i} = (i_1, \ldots, i_d)$ such that $i_j \in [1\,..\,n_j]$ when $j \neq k$ and $i_k \in [1\,..\,n_k + m_k]$, it is:

- $(\mathcal{M}_\mathbf{n} \oslash_{(k)} \mathcal{M}'_\mathbf{m})[\mathbf{i}] = \mathcal{M}_\mathbf{n}[\mathbf{i}]$ if $i_k \leqslant n_k$;
- $(\mathcal{M}_\mathbf{n} \oslash_{(k)} \mathcal{M}'_\mathbf{m})[\mathbf{i}] = \mathcal{M}'_\mathbf{m}[\mathbf{i}']$ otherwise

where $\mathbf{i}'$ is a d-tuple with ${i_j}' = i_j$ for all $j \neq k$ and $i'_k = i_k - n_k$.

The dD substring complexity $P_{\mathcal{M}}$ counts for each d-tuple of positive integers $(k_1, \ldots, k_d)$ the number of distinct $(k_1 \times \cdots \times k_d)$-factors in $\mathcal{M}_{\mathbf{n}}$.

In the following, we provide the above-mentioned definitions of the measures δ, γ, g, g_{rl}, and b for the d-dimensional case. Observe that as $d = 2$, these match with the definitions given in the previous sections.

Definition 15 Let $\mathcal{M}_{\mathbf{n}}$ be a dD string, for some $d > 0$ and $\mathbf{n} = (n_1, \ldots, n_d)$, and let $P_{\mathcal{M}}$ be the dD substring complexity of $\mathcal{M}_{\mathbf{n}}$. Then,

$$\delta(\mathcal{M}_{\mathbf{n}}) = \max\left\{\frac{P_{\mathcal{M}}(k_1, \ldots, k_d)}{k_1 \cdots k_d}, 1 \leqslant k_i \leqslant n_i \text{ for all } 1 \leqslant i \leqslant d\right\}.$$

Definition 16 An *attractor* for a dD string $\mathcal{M}_{\mathbf{n}}$, for some $d > 0$ and $\mathbf{n} = (n_1, \ldots, n_d)$, is a set $\Gamma \subseteq [1\,..\,n_1] \times [1\,..\,n_2] \times \cdots \times [1\,..\,n_d]$ with the property that any dD substring $\mathcal{M}[i_1\,..\,j_1][i_2\,..\,j_2]\cdots[i_d\,..\,j_d]$ of $\mathcal{M}_{\mathbf{n}}$ has an occurrence $\mathcal{M}[i'_1\,..\,j'_1][i'_2\,..\,j'_2]\cdots[i'_d\,..\,j'_d]$ such that $\exists \mathbf{x} = (x_1, \ldots, x_d) \in \Gamma$ with $i'_i \leqslant x_i \leqslant j'_i$ for all $i \in [1\,..\,d]$. The size of the smallest attractor for $\mathcal{M}_{\mathbf{n}}$ is denoted by $\gamma(\mathcal{M}_{\mathbf{n}})$.

Definition 17 Let $\mathcal{M}_{\mathbf{n}}$ be a dD string, for some $d > 0$ and $\mathbf{n} = (n_1, \ldots, n_d)$. A *$d$-dimensional Straight-Line Program* (dD SLP) for $\mathcal{M}_{\mathbf{n}}$ is a context-free grammar (V, Σ, R, S) that uniquely generates $\mathcal{M}_{\mathbf{n}}$ and where the definition of the right-hand side of a variable can have the form

$$A \to a,\ A \to B \oslash_{(i)} C,\ \text{with } i \in [1\,..\,d],$$

where $a \in \Sigma$, $B, C \in V$. We call these definitions *terminal rules* and *i-rules* respectively. The expansion of a variable is defined as

$$\mathtt{exp}(A) = a,\ \mathtt{exp}(A) = \mathtt{exp}(B) \oslash_{(i)} \mathtt{exp}(C),\ \text{with } i \in [1\,..\,d],$$

respectively.

The size $|G|$ of a dD SLP G is the sum of the sizes of all the rules of G, where we assume that the terminal rules have size 1 and the i-rules have size 2. The measure $g(\mathcal{M}_{\mathbf{n}})$ is defined as the size of the smallest dD SLP generating $\mathcal{M}_{\mathbf{n}}$.

Definition 18 A *d-dimensional Run-Length Straight-Line program* (dD RLSLP) is a dD SLP that in addition allows special rules, which are assumed to be of size 2, of the form

$$A \to \oslash^k_{(i)} B,\ \text{with } i \in [1\,..\,d],\ \text{and } k > 1$$

with their expansions defined as

$$\mathtt{exp}(A) = \underbrace{\mathtt{exp}(B) \oslash_{(i)} \mathtt{exp}(B) \oslash_{(i)} \cdots \oslash_{(i)} \mathtt{exp}(B)}_{k \text{ times}}$$

The measure $g_{rl}(\mathcal{M}_{\mathbf{n}})$ is defined as the sum of the size of the rules of a smallest dD RLSLP generating $\mathcal{M}_{\mathbf{n}}$.

Definition 19 Let $\mathcal{M}_{\mathbf{n}}$ be a dD string, for some $d > 0$ and $\mathbf{n} = (n_1, \ldots, n_d)$. A *$d$D macro scheme* for $\mathcal{M}_{\mathbf{n}}$ is any factorization of $\mathcal{M}_{\mathbf{n}}$ into a set of disjoint phrases such that any phrase is either an element from Σ called an *explicit symbol/phrase*, or is a copied phrase with source in $\mathcal{M}_{\mathbf{n}}$ starting at a different position. For a dD macro scheme to be *valid* or *decodable*, the function

$$\texttt{map} : ([1\,..\,n_1] \times \cdots \times [1\,..\,n_d]) \cup \{\perp\} \to ([1\,..\,n_1] \times \cdots \times [1\,..\,n_d]) \cup \{\perp\}$$

induced by the factorization must verify that:

i) $\texttt{map}(\perp) = \perp$, and if $\mathcal{M}_{\mathbf{n}}[\mathbf{i}]$ is an explicit symbol, then $\texttt{map}(\mathbf{i}) = \perp$;
ii) for each copied phrase $\mathcal{M}_{\mathbf{n}}[\mathbf{i}\,..\,\mathbf{j}]$, it must hold that $\texttt{map}(\mathbf{i} + \mathbf{t}) = \texttt{map}(\mathbf{i}) + \mathbf{t}$ for $\mathbf{t} \in [0\,..\,j_1 - i_1] \times \cdots \times [0\,..\,j_d - i_d]$, where $\texttt{map}(\mathbf{i})$ is the top-left corner of the source for $\mathcal{M}_{\mathbf{n}}[\mathbf{i}\,..\,\mathbf{j}]$;
iii) for each $\mathbf{i} \in [1\,..\,n_1] \times \cdots \times [1\,..\,n_d]$ there exists $k > 0$ such that $\texttt{map}^k(\mathbf{i}) = \perp$.

We define $b(\mathcal{M}_{\mathbf{n}})$ as the smallest number of phrases in a valid dD macro scheme for $\mathcal{M}_{\mathbf{n}}$.

It is not hard to see that Propositions 6, 12, and 22, establishing that $\delta \leqslant \gamma$, and $b \leqslant g_{rl} \leqslant g$, can be generalized to the d-dimensional case, with analogous proofs, for all $d > 2$. We conclude this section by showing a class of multidimensional strings for which the gap between the measures δ and g grows with the number of dimensions d.

Consider again a binary de Bruijn sequence D_k of size $n = 2^k + k - 1$. We define the dD string $B_{d,k}$ of size $N = \overbrace{n \times \cdots \times n}^{d}$ over the alphabet $\Delta_d = \{\langle b_1, \ldots, b_d\rangle \mid b_i \in \{0, 1\}, i \in [1\,..\,d]\}$ of size 2^d by the following relation: given a position $\mathbf{i} = (i_1, \ldots, i_d) \in [1\,..\,n]^d$ it is

$$B_{d,k}[\mathbf{i}] = \langle D_k[i_1], \ldots, D_k[i_d]\rangle.$$

Observe that the definition of $B_{d,k}$ is a generalization of the 2D string B_k defined in Section 6 since $B_{2,k} = B_k$.

Lemma 36 *For every $d, k > 0$ it is $g(B_{d,k}) = O(2^d n / \log n)$.*

Proof For $d = 1$, Navarro et al. showed in [13] that the statement is true for all k, that is, there exists a one-dimensional SLP $G_{1,k}$ of size $O(n/\log n)$ generating the de Bruijn sequence D_k. In the following, we exhibit a d-dimensional SLP $G_{d,k}$ of the claimed size which expands to $B_{d,k}$ for all d and k. We note that for every i, the substring $B_{d,k}[1..n]..[1..n][i]$ can be of only two types depending on the value of $D_k[i]$, in particular, it is equal (comparing them element by element) to $B_{d-1,k}$ where we replaced every element $B_{d-1,k}[i_1]..[i_{d-1}] = \langle D_k[i_1], \ldots, D_k[i_{d-1}]\rangle$ with respectively $\langle D_k[i_1], \ldots, D_k[i_{d-1}], 0\rangle$ if $D_k[i] = 0$, or with $\langle D_k[i_1], \ldots, D_k[i_{d-1}], 1\rangle$ otherwise. In the following we call these two $(d-1)$D strings respectively $B^0_{d-1,k}$ and $B^1_{d-1,k}$. We obtain a grammar $G_{d,k}$ for $B_{d,k}$ inductively as follows. Given a grammar $G_{d-1,k}$ for $B_{d-1,k}$ we obtain the grammars $G^0_{d-1,k}$ and $G^1_{d-1,k}$ corresponding to

$B^0_{d-1,k}$ and $B^1_{d-1,k}$ by replacing in $G_{d-1,k}$ each terminal symbol $\langle b_1, \ldots, b_{d-1}\rangle$ with $\langle b_1, \ldots, b_{d-1}, 0\rangle$ or $\langle b_1, \ldots, b_{d-1}, 1\rangle$ respectively; in both cases without changing the size of the grammars. To obtain the final grammar we transform all the horizontal rules of the one-dimensional grammar $G_{1,k}$ in $①_{(d)}$ rules, and we replace the terminals 0 and 1 respectively with the starting symbols of the grammars $G^0_{d-1,k}$ and $G^1_{d-1,k}$ without changing the size of $G_{1,k}$. As a consequence the resulting grammar has size $|G_{d,k}| = O(n/\log n) + 2|G_{d-1,k}| = O(2^d n/\log n)$. □

The proof of Lemma 25 can be easily generalized to derive the following.

Lemma 37 *Every substring of size* $\underbrace{k \times \cdots \times k}_{d}$ *appears in* $B_{d,k}$ *at most once.*

Proof We observe that by definition of $B_{d,k}$, moving in $B_{d,k}$ along every fixed dimension j, we read a de Bruijn sequence over a binary alphabet. More formally, for all j, the substring $B_{d,k}[i_1]..[i_{j-1}][1..n][i_{j+1}]..[i_d]$ spells a de Bruijn sequence of order k over the binary alphabet having symbols $\langle D_k[i_1], .., D_k[i_{j-1}], 0, D_k[i_{j+1}], .., D_k[i_d]\rangle$ and $\langle D_k[i_1], .., D_k[i_{j-1}], 1, D_k[i_{j+1}], .., D_k[i_d]\rangle$. As a consequence, there cannot be two distinct occurrences of the same $\underbrace{k \times \cdots \times k}_{d}$ substring. □

Proposition 38 *For every* $d > 0$*, there exists an infinite family of* d*D strings with* $\delta = \Omega\left(gN^{\frac{d-1}{d}}\left(\frac{d}{2\log N}\right)^{d-1}\right)$*, where* N *is the size of the input string.*

Proof Consider the dD string $B_{d,k}$, for some $d, k \geqslant 1$.

By Lemma 36 it is $g(B_{d,k}) = O(2^d n/\log n) = O(d2^d N^{\frac{1}{d}}/\log N)$. By Lemma 37 $B_{d,k}$ contains $\Theta(n^d) = \Theta(N)$ distinct $\underbrace{k \times \cdots \times k}_{d}$ factors, hence $\delta = \Omega\left(\frac{Nd^d}{\log^d N}\right)$. □

9.1 Direct access to *d*D RLSLPs

The heavy-path approach for direct access on 2D RLSLPs can be generalized to dD RLSLPs. We need to take into account that in dD RLSLPs, there are d predecessor queries that need to be made to change the heavy-path. In terms of space, for each variable we need to store its dimension array (the size of its expansion), the coordinate of its heavy symbol (the heavy occurrence), and d pair of size sequences values.

After making the above considerations, the following result is straightforward.

Theorem 39 *Let* $\mathcal{M}_{\mathbf{n}}$ *be a* N*-size* d*D string and let* G_{rl} *be a* d*D RLSLP generating* $\mathcal{M}_{\mathbf{n}}$*, for some* $\mathbf{n} = (n_1, \ldots, n_d)$*. There exists a data structure that uses* $O(|G_{rl}|\log N)$ *bits of space and supports direct access queries to any cell* $\mathcal{M}_{\mathbf{n}}[\mathbf{i}]$ *in* $O(d\log N)$ *time. Moreover, if* d *processes can be executed in parallel, we can support direct access queries to any cell* $\mathcal{M}_{\mathbf{n}}[\mathbf{i}]$ *in* $O(\log N)$ *time.*

10 Conclusions and Future Works

In this paper, we have shown how to generalize the repetitiveness measures previously used in the one-dimensional context to generic two-dimensional strings. In particular, we have introduced extensions to the 2D case of the measures δ and γ based on distinct factors of arbitrary rectangular shape, as well as the extensions of the measures g, g_{rl}, and b, which are based on copy-paste mechanisms. We have studied the mutual relationships between these measures and we have shown that $\delta \leqslant \gamma$ and $b \leqslant g_{rl} \leqslant g$. We have proven that, unlike in the 1D context where $\delta \leqslant \gamma \leqslant b \leqslant g_{rl} \leqslant g$, the two classes of measures become incomparable when 2D strings are considered. Indeed, we have shown that, depending on the 2D input, the measures g, g_{rl}, and b can be asymptotically smaller than δ and γ.

The results presented in the paper highlight that in the 2D case, the measures δ and γ (as well as their square-based versions introduced in [7, 10]) are not completely satisfactory for capturing the regularities of a generic two-dimensional string, which are instead effectively detected by g, g_{rl}, and b measures.

We have also analyzed the recently introduced 2D Block-Tree data structure [6] which is able to compress a 2D string and provide efficient access to its individual symbols. Our results show that for some 2D strings the 2D Block-Tree fails to achieve a compression close to g (and therefor g_{rl} and b). We have also studied the use of linearization strategies as preprocessing to compress two-dimensional input, and shown that they are not always effective even when considering approaches based on the Peano-Hilbert space-filling curve.

Our results indicate that the problem of finding a time-efficient 2D compression scheme that approaches the theoretically well-grounded measures g, g_{rl}, or b is still open. A possible avenue to tackle this problem could be to explore possible approximation strategies for b and g, as well as 2D versions of greedy grammar construction algorithms like the ones described in [12, 13]. Our analysis suggests that another strategy worth pursuing is to modify the 2D-block tree so that it is not limited to searching for identical square substrings; handling rectangular shapes as well appears to be essential for maximizing compression.

Author Contributions All authors equally contributed to the whole writing and reviewing process of the manuscript.

Funding Open access funding provided by Università degli Studi di Palermo within the CRUI-CARE Agreement. LC and GM are partially funded by the PNRR ECS00000017 Tuscany Health Ecosystem, Spoke 6, CUP I53C22000780001, funded by the NextGeneration EU programme, by the spoke "FutureHPC & BigData" of the ICSC — Centro Nazionale di Ricerca in High-Performance Computing, Big Data and Quantum Computing, funded by the NextGeneration EU programme.
GR and MS are partially funded by the MUR PRIN Project "PINC, Pangenome INformatiCs: from Theory to Applications" (Grant No. 2022YRB97K).
MS is partially supported by Project "ACoMPA" (CUP B73C24001050001) funded by the NextGeneration EU programme PNRR ECS00000017 Tuscany Health Ecosystem (Spoke 6).
LC, GM, MS, and GR are partially funded by the INdAM-GNCS Project CUP E53C24001950001.
CU is partially funded by ANID-Subdirección de Capital Humano/Doctorado Nacional/2021-21210580, ANID, Chile; NIC Chile Doctoral Scholarship, NIC, Chile; Basal Funds FB0001 and AFB240001, ANID, Chile; and FONDECYT Project 1-230755, ANID, Chile.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Competing interests The authors declare no competing interests.

References

1. Kempa, D., Prezza, N.: At the roots of dictionary compression: string attractors. In: STOC, pp. 827–840. ACM, New York (2018)
2. Navarro, G.: Indexing highly repetitive string collections, part I: repetitiveness measures. ACM Comput. Surv. **54**(2), 29 (2021)
3. Belazzougui, D., Cáceres, M., Gagie, T., Gawrychowski, P., Kärkkäinen, J., Navarro, G., Ordóñez, A., Puglisi, S.J., Tabei, Y.: Block trees. J. Comput. Syst. Sci. **117**, 1–22 (2021)
4. Navarro, G.: Indexing highly repetitive string collections, part II: compressed indexes. ACM Comput. Surv. **54**(2), 26 (2021)
5. Brisaboa, N., Gagie, T., Gómez-Brandón, A., Navarro, G.: Two-dimensional block trees. In: Proceedings of the 28th Data Compression Conference (DCC), pp. 229–238 (2018)
6. Brisaboa, N.R., Gagie, T., Gómez-Brandón, A., Navarro, G.: Two-dimensional block trees. Comput. J. **67**(1), 391–406 (2024)
7. Carfagna, L., Manzini, G.: Compressibility measures for two-dimensional data. In: Proceedings of the 30th international symposium on String Processing and Information Retrieval, SPIRE 2023. LNCS, vol. 14240, pp. 102–113. Springer, Cham (2023)
8. Giancarlo, R.: A generalization of the suffix tree to square matrices, with applications. SIAM J. Comput. **24**(3), 520–562 (1995)
9. Kim, D.K., Na, J.C., Sim, J.S., Park, K.: Linear-time construction of two-dimensional suffix trees. Algorithmica **59**(2), 269–297 (2011). https://doi.org/10.1007/s00453-009-9350-z
10. Carfagna, L., Manzini, G.: The landscape of compressibility measures for two-dimensional data. IEEE Access **12**, 87268–87283 (2024)
11. Storer, J.A., Helfgott, H.: Lossless image compression by block matching. Comput. J. **40**(2–3), 137–145 (1997). https://doi.org/10.1093/comjnl/40.2_and_3.137
12. Bannai, H., Hirayama, M., Hucke, D., Inenaga, S., Jez, A., Lohrey, M., Reh, C.P.: The smallest grammar problem revisited. IEEE Trans. Inf. Theory **67**(1), 317–328 (2021)
13. Navarro, G., Ochoa, C., Prezza, N.: On the approximation ratio of ordered parsings. IEEE Trans. Inf. Theory **67**(2), 1008–1026 (2021)
14. Rizzo, F., Storer, J.A., Carpentieri, B.: Lz-based image compression. Inf. Sci. **135**(1–2), 107–122 (2001). https://doi.org/10.1016/s0020-0255(01)00104-9
15. Giammarresi, D., Restivo, A.: Two-dimensional languages. In: Handbook of formal languages (3), pp. 215–267. Springer, Berlin, Heidelberg, New York (1997)
16. Berman, P., Karpinski, M., Larmore, L.L., Plandowski, W., Rytter, W.: On the complexity of pattern matching for highly compressed two-dimensional texts. J. Comput. Syst. Sci. **65**(2), 332–350 (2002)
17. Christiansen, A.R., Ettienne, M.B., Kociumaka, T., Navarro, G., Prezza, N.: Optimal-time dictionary-compressed indexes. ACM Trans. Algorithms **17**(1), 8–1839 (2021)
18. Kociumaka, T., Navarro, G., Prezza, N.: Toward a definitive compressibility measure for repetitive sequences. IEEE Trans. Inf. Theory **69**(4), 2074–2092 (2023)

19. Prezza, N.: On string attractors. In: Proceedings of the 19th Italian Conference on Theoretical Computer Science. ICTCS (2018). https://ceur-ws.org/Vol-2243/award1.pdf
20. Mantaci, S., Restivo, A., Romana, G., Rosone, G., Sciortino, M.: A combinatorial view on string attractors. Theor. Comput. Sci. **850**, 236–248 (2021)
21. Bernardini, G., Fici, G., Gawrychowski, P., Pissis, S.P.: Substring Complexity in Sublinear Space. In: 34th Int. Symposium on Algorithms and Computation (ISAAC 2023). Leibniz International Proceedings in Informatics (LIPIcs), vol. 283, pp. 12–11219 (2023)
22. Charikar, M., Lehman, E., Liu, D., Panigrahy, R., Prabhakaran, M., Sahai, A., Shelat, A.: The smallest grammar problem. IEEE Trans. Inf. Theory **51**(7), 2554–2576 (2005)
23. Bille, P., Landau, G.M., Raman, R., Sadakane, K., Satti, S.R., Weimann, O.: Random access to grammar-compressed strings and trees. SIAM J. Comput. **44**(3), 513–539 (2015)
24. Farach, M., Muthukrishnan, S.: Perfect hashing for strings: Formalization and algorithms. In: Combinatorial pattern matching, pp. 130–140. Springer, Berlin, Heidelberg (1996)
25. Storer, J.A., Szymanski, T.G.: Data compression via textual substitution. J. ACM **29**(4), 928–951 (1982). https://doi.org/10.1145/322344.322346
26. Gallant, J.K.: String compression algorithms. PhD thesis, Princeton University (1982)
27. Gagie, T., Navarro, G., Prezza, N.: On the approximation ratio of Lempel-Ziv parsing. In: Proceedings LATIN 2018. LNCS, vol. 10807, pp. 490–503. Springer, Cham (2018)
28. Lempel, A., Ziv, J.: Compression of two-dimensional data. IEEE Trans. Inf. Theory **32**(1), 2–8 (1986)

Authors and Affiliations

Lorenzo Carfagna[1] · Giovanni Manzini[1] · Giuseppe Romana[2] · Marinella Sciortino[2] · Cristian Urbina[3,4]

✉ Giuseppe Romana
giuseppe.romana01@unipa.it

Lorenzo Carfagna
lorenzo.carfagna@phd.unipi.it

Giovanni Manzini
giovanni.manzini@unipi.it

Marinella Sciortino
marinella.sciortino@unipa.it

Cristian Urbina
crurbina@dcc.uchile.cl

1 Dipartimento di Informatica, University of Pisa, Pisa, Italy

2 Dipartimento di Matematica e Informatica, University of Palermo, Palermo, Italy

3 Department of Computer Science, University of Chile, Santiago, Chile

4 Centre for Biotechnology and Bioengineering (CeBiB), Santiago, Chile