Compact Representation for Matrices of Bounded Twin-Width

Michał Pilipczuk
Institute of Informatics, University of Warsaw, Poland

Marek Sokołowski
Institute of Informatics, University of Warsaw, Poland

Anna Zych-Pawlewicz
Institute of Informatics, University of Warsaw, Poland

Abstract

For every fixed $d \in \mathbb{N}$, we design a data structure that represents a binary $n \times n$ matrix that is d-twin-ordered. The data structure occupies $\mathcal{O}_d(n)$ bits, which is the least one could hope for, and can be queried for entries of the matrix in time $\mathcal{O}_d(\log \log n)$ per query.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases twin-width, compact representation, adjacency oracle

Digital Object Identifier 10.4230/LIPIcs.STACS.2022.52

Related Version *Full Version*: `https://arxiv.org/abs/2110.08106` [14]

Funding This work is a part of projects that have received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreements No. 948057 (Mi. Pilipczuk, M. Sokołowski) and 714704 (A. Zych-Pawlewicz).

1 Introduction

Consider a *binary* matrix M, that is, one with entries in $\{0, 1\}$. Given two consecutive columns c_1, c_2 of M, the operation of *contracting* those columns consists of replacing them with a single column c with entries reconciled as follows. If in a row r columns c_1, c_2 agree, that is, both contain symbol 0 or both contain symbol 1, then in column c in row r we put the same symbol. Otherwise, we put a special mismatch symbol $\perp$. Contracting consecutive rows is defined analogously. Contractions of rows and columns can be then applied further with the rule that reconciling any entry with the mismatch symbol $\perp$ again results in $\perp$. For a nonnegative integer $d \in \mathbb{N}$, we say that M is *d-twin-ordered* if M can be contracted to a 1×1 matrix in such a way that at every point during the process, every row and every column contains at most d symbols $\perp$. Finally, the *twin-width* of M is the least d for which one can permute rows and columns of M so that the resulting matrix is d-twin-ordered.

The notion of twin-width was introduced very recently by Bonnet et al. in [5], and has immediately gathered immense interest. As shown in [5] and in multiple subsequent works [1, 2, 3, 4, 6, 9, 10], twin-width is a versatile measure of complexity not only for matrices, but also for permutations and for graphs by considering a suitable matrix representation, which in the latter case is just the adjacency matrix. In particular, for every fixed $t \in \mathbb{N}$, graphs excluding K_t as a minor and graphs having cliquewidth at most t have bounded twin-width, which means that the concept of boundedness of twin-width is a vast generalization of boundedness of cliquewidth that does not assume tree-likeness of the structure of the graph. As shown in the aforementioned works, this generalization is combinatorially rich [1, 5, 9], algorithmically useful [2, 4, 5], and exposes deep connections with notions studied in finite

model theory [3, 5, 6, 10]. In particular, assuming a suitable contraction sequence is provided on input, model-checking First-Order logic on graphs of bounded twin-width can be done in linear fixed-parameter time [5].

One of the fundamental directions in the work on twin-width is that of estimating the asymptotic *growth* of considered classes of objects. It has been proved in [1] that the number of distinct graphs on vertex set $\{1, \ldots, n\}$ that have twin-width at most d is bounded by $2^{\mathcal{O}_d(n)} \cdot n!$, which renders the class of graphs of twin-width at most d *small*. Similarly, the number of distinct $n \times n$ binary matrices that are d-twin-ordered is upper bounded by $2^{\mathcal{O}_d(n)}$ [3] (see also the proof of Lemma 16).

The latter result raises a natural data structure question, which we address in this work. In principle, a binary $n \times n$ matrix of twin-width at most d can be encoded using $\mathcal{O}_d(n)$ bits, just because the number of such matrices is bounded by $2^{\mathcal{O}_d(n)}$. However, would it be possible to design such a representation so that it is algorithmically useful in the following sense: the representation may serve as a data structure that supports efficient queries for entries of the matrix. This question originates from the area of *compact representations*, see for instance the work on such representations for graphs of bounded cliquewidth [11].

Let us review some solutions to the above problem that to smaller or larger extent follow from existing literature. The quality of a representation is measured by the number of bits it occupies and the worst-case time complexity of a query for an entry. Here, we assume the standard word RAM model with word length $\mathcal{O}(\log n)$.

- Storing the matrix explicitly is a representation with bitsize $\mathcal{O}(n^2)$ and query time $\mathcal{O}(1)$.
- In [1], Bonnet et al. presented an *adjacency labelling scheme* for graphs of bounded twin-width, which can be readily translated to the matrix setting. This scheme assigns to each row and each column of the matrix a *label* – a bitstring of length $\mathcal{O}_d(\log n)$ – so that the entry in the intersection of a row and a column can be uniquely decoded from the pair of their labels. In [1] the time complexity of this decoding is not analyzed, but a straightforward implementation runs in time linear in the length of labels. This gives a representation with bitsize $\mathcal{O}_d(n \log n)$ and query time $\mathcal{O}_d(\log n)$.
- It follows from the results of [2] that if matrix M is d-twin-ordered, then the entries 1 in M can be partitioned into $\ell = \mathcal{O}_d(n)$ rectangles, say $R_1, \ldots, R_\ell$ (see Lemma 10 for a proof). This reduces our question to *2D orthogonal point location*: designing a data structure that for a given point in $(i, j) \in \{1, \ldots, n\}^2$, may answer whether (i, j) belongs to any of the rectangles $R_1, \ldots, R_\ell$. For this problem, Chan [7] designed a data structure with bitsize $\mathcal{O}(n \log n)$ and query time $\mathcal{O}(\log \log n)$ assuming $\ell = \mathcal{O}(n)$. So we get a representation of M with bitsize $\mathcal{O}_d(n \log n)$ and query time $\mathcal{O}_d(\log \log n)$.
- For 2D orthogonal point location one can also design a simple data structure by persistently recording a sweep of the square $\{1, \ldots, n\}^2$ using a B-ary tree for $B = n^\varepsilon$, for any fixed $\varepsilon > 0$. This gives a representation with bitsize $\mathcal{O}_d(n^{1+\varepsilon})$ and query time $\mathcal{O}(1/\varepsilon)$. See Appendix A for details.

Note that in all solutions above, the bitsize of the representation is $\Omega(n \log n)$, and thus does not reach the information-theoretic limit of $\Theta_d(n)$.

Our result. We design a compact representation for d-twin-ordered matrices that simultaneously occupies $\mathcal{O}_d(n)$ bits and offers query time $\mathcal{O}_d(\log \log n)$. The result is summarized in the statement below.

▶ **Theorem 1.** *Let $d \in \mathbb{N}$ be a fixed constant. Then for a given binary $n \times n$ matrix M that is d-twin-ordered one can construct a data structure that occupies $\mathcal{O}_d(n)$ bits and can be queried for entries of M in worst-case time $\mathcal{O}(\log \log n)$ per query. The construction time is $\mathcal{O}_d(n \log n \log \log n)$ in the word RAM model, assuming M is given by specifying $\ell = \mathcal{O}_d(n)$ rectangles $R_1, \ldots, R_\ell$ that form a partition of symbols 1 in M.*

The proof of Theorem 1 proceeds roughly as follows. Consider a parameter m that divides n and a partition of the given matrix M into $(n/m)^2$ *zones* – square submatrices – each of which is induced by m consecutive rows and m consecutive columns. Such a partition is called the *regular* (n/m)*-division*. Even though the total number of zones in the regular (n/m)-division is $(n/m)^2$, one can use the connections between the notions of being twin-ordered and that of *mixed minors*, developed in [5], to show that actually there will be only $\mathcal{O}_d(n/m)$ *different* zones, in the sense that zones are considered equal if they have exactly the same values in corresponding entries.

Our data structure describes the zones in the regular (n/m)-divisions of M for m ranging over a sequence of parameters $m_0 > m_1 > \ldots > m_\ell$ for $\ell = \mathcal{O}(\log \log n)$, where m_j divides m_i whenever $i \leq j$. Roughly speaking, we set $m_0 = n$ and $m_i = m_{i-1}^{2/3}$ for $i \geq 1$, though for technical reasons we resort to the recursion $m_i = m_{i-1}/2$ once m_i reaches the magnitude of $\log^3 n$. Each different zone present in the regular (n/m_i)-division is represented by a square matrix consisting of $(m_i/m_{i+1})^2$ pointers to representations of its subzones in the regular (n/m_{i+1})-division. When we reach $m_i < c_d \cdot \log n$ for some small constant c_d depending on d, we stop the construction and set $\ell = i$. At this point the number of different zones present in the regular (n/m_ℓ)-division of M is strongly sublinear in n, because we have such an upper bound on the total number of different $(c_d \log n) \times (c_d \log n)$ binary matrices that are d-twin-ordered, and $n/m_\ell \leq c_d \log n$. Therefore, all those matrices can be stored in the representation explicitly within bitsize $\mathcal{O}_d(n)$.

The query algorithm is very simple: just follow appropriate pointers through the $\mathcal{O}(\log \log n)$ levels of the data structure and read the relevant entry in a matrix stored explicitly in the last level. The analysis of bitsize is somewhat more complicated, but crucially relies on the fact that in the ith level, it suffices to represent only $\mathcal{O}_d(n/m_i)$ different matrices that are zones in the (n/m_i)-division.

We remark that the idea of dividing the given matrix into a number of polynomially smaller zones, and describing them recursively, is also the cornerstone of the approach used by Chan for the orthogonal point location problem in [7]. However, when it comes to details, his construction is quite different and technically more complicated. For instance, in [7] the recursion can be applied not only on single zones, but also on wide or tall strips consisting of several zones, or even submatrices induced by non-contiguous subsets of rows and columns. The conceptual simplification achieved here comes from the strong properties implied by the assumption that the matrix is d-twin-ordered, which is stronger than the assumption used by Chan that the symbols 1 in the matrix can be partitioned into $\mathcal{O}(n)$ rectangles.

Organization of the paper. In Section 2 we define the twin-width of matrices formally and recall a number of notions related to d-twin-ordered matrices. In Section 3, we prove several new structural properties of those matrices. These properties are exploited in Section 4 to derive an efficient and compact representation of d-twin-ordered matrices, completing the non-constructive part of Theorem 1. The efficient algorithm for construction of the data structure is deferred to the full version of the paper [14].

2 Preliminaries

For a positive integer p we write $[p] = \{1, \ldots, p\}$. We use the $\mathcal{O}_d(\cdot)$ notation to hide multiplicative factors depending on d.

Matrices, divisions, and zones. A *binary matrix* is a matrix with entries in $\{0,1\}$; all matrices in this paper are binary unless explicitly stated.

Let M be a matrix. Note that rows of M are totally ordered, and similarly for columns of M. A *row block* in M is a non-empty set of rows of M that are consecutive in this total order; *column blocks* are defined analogously. If R is a row block and C is a column block, then the *zone* induced by R and C is the rectangular submatrix of M consisting of entries at the intersections of rows from R and columns from C. In general, by a *submatrix* of M we mean the zone induced by some row block and some column block.

A submatrix is *constant* if all its entries are the same. It is horizontal if all its columns are the same (equivalently, all rows are constant), and *vertical* if all its rows are the same (equivalently, all columns are constant). Note that thus, a constant submatrix is both horizontal and vertical. A submatrix that is neither horizontal nor vertical is called *mixed*.

A *division* of matrix M is a pair $(\mathcal{R},\mathcal{C})$, where $\mathcal{R}$ is a partition of rows into row blocks and $\mathcal{C}$ is a partition of columns into column blocks. Note that such a division partitions M into $|\mathcal{R}|\cdot|\mathcal{C}|$ zones, each induced by a pair of blocks $(R,C)\in\mathcal{R}\times\mathcal{C}$. We call them the *zones* of the division $(\mathcal{R},\mathcal{C})$. A *$t$-division* is a division where $|\mathcal{R}|=|\mathcal{C}|=t$.

Twin-width. Let M be a matrix. If $(\mathcal{R},\mathcal{C})$ is a division of M, then a *contraction* of $(\mathcal{R},\mathcal{C})$ is any division $(\mathcal{R}',\mathcal{C}')$ obtained from $(\mathcal{R},\mathcal{C})$ by either merging two consecutive row blocks $R_1,R_2\in\mathcal{R}$ into a single row block $R_1\cup R_2$, or merging two consecutive column blocks $C_1,C_2\in\mathcal{C}$ into a single column block $C_1\cup C_2$. A *contraction sequence* for M is a sequence of divisions

$$(\mathcal{R}_0,\mathcal{C}_0),(\mathcal{R}_1,\mathcal{C}_1),\dots,(\mathcal{R}_p,\mathcal{C}_p),$$

such that

- $(\mathcal{R}_0,\mathcal{C}_0)$ is the finest division where every row and every column is in a separate block;
- $(\mathcal{R}_p,\mathcal{C}_p)$ is the coarsest partition where all rows are in a single row block and all columns are in a single column block; and
- for each $i\in\{1,\dots,p\}$, $(\mathcal{R}_i,\mathcal{C}_i)$ is a contraction of $\mathcal{R}_{i-1},\mathcal{C}_{i-1}$.

Note that thus, p has to be equal to the sum of the dimension of M minus 2.

Finally, for a division $(\mathcal{R},\mathcal{C})$ of M, the *error value* of $(\mathcal{R},\mathcal{C})$ is the least d such that in $(\mathcal{R},\mathcal{C})$, every row block and every column block contains at most d non-constant zones.

With all these ingredients in place, we can formally define the twin-width of matrices. There are a few alternative definitions spanning through [1, 2, 3, 5]; here we follow the terminology from [5].

▶ **Definition 2.** *A binary matrix M is* d-twin-ordered *if it admits a contraction sequence in which every division has error value at most d. The* twin-width *of a binary matrix M is the least d such that one can permute the rows and columns of M so that the obtained matrix is d-twin-ordered.*

It is straightforward to see that the definition above is equivalent to the one given in the first paragraph of Section 1.

Observe that in the above definition, certifying that a matrix is d-twin-ordered requires showing a suitable contraction sequence where all divisions have error value at most d. In our algorithmic results we do not require that such a contraction sequence is given on input, as we will exploit the assumption that the matrix is d-twin-ordered only through combinatorial properties provided by the connections with mixed minors, which we discuss next. In fact, as discussed [5], it is currently unknown how to efficiently compute a contraction sequence witnessing that a matrix is d-twin-ordered.

Matrix minors and Marcus-Tardos Theorem. We need the following definitions of matrix minors, which intuitively are "complicated substructures" in matrices.

▶ **Definition 3.** *Let M be a binary matrix. A* t-grid minor *in M is a t-division of M where every zone contains at least one entry* 1*. A* t-mixed minor *in M is a t-division of M where every zone is mixed. We say that M is* t-mixed-free *if M does not contain a t-mixed minor.*

The celebrated result of Marcus and Tardos asserts that if a matrix has a large density of entries 1, then it contains a large grid minor.

▶ **Theorem 4** ([12])**.** *For every $t \in \mathbb{N}$ there exists $c_t \in \mathbb{N}$ such that the following holds. Suppose M is an $n \times m$ binary matrix with at least $c_t \cdot \max(n, m)$ entries* 1*. Then M has a t-grid minor.*

The currently best upper bound on c_t is $\frac{8}{3}(t+1)^2 2^{4t}$, due to Cibulka and Kynčl [8]. From now on we adopt the constant c_t in the notation.

In [5], Bonnet et al. used the result of Marcus and Tardos to show that, intuitively, large mixed minors are canonical obstacles for having bounded twin-width.

▶ **Theorem 5** ([5])**.** *Let M be a binary matrix. Then the following implications hold:*

- *If M is d-twin-ordered, then M is $(2d+2)$-mixed-free.*
- *If M is t-mixed-free, then M has twin-width at most k_t, where k_t is a constant depending only on t.*

Note that the conclusion of the second implication of Theorem 5 is only a bound on the twin-width: the matrix might still need to be permuted to be k_t-twin-ordered. In this work we will only rely on the first implication of Theorem 5: being d-twin-ordered implies $(2d+2)$-mixed-freeness.

We now derive some simple properties of mixed-free matrices that will be used throughout the paper. First, in a t-mixed-free matrix every ℓ-division has only $\mathcal{O}_t(\ell)$ mixed zones.

▶ **Lemma 6.** *Let M be a t-mixed free matrix, and let $(\mathcal{R}, \mathcal{C})$ be an ℓ-division of M, for some integer ℓ. Then $(\mathcal{R}, \mathcal{C})$ has at most $c_t \cdot \ell$ mixed zones.*

Proof. Construct an $\ell \times \ell$ matrix A by taking the division $(\mathcal{R}, \mathcal{C})$ and substituting each mixed zone with a single entry 1, and each non-mixed zone with a single entry 0. Observe that A may have at most $c_t \cdot \ell$ entries 1, for otherwise, by Theorem 4, A would contain a t-grid minor, which would correspond to a t-mixed minor in M. Hence $(\mathcal{R}, \mathcal{C})$ may have at most $c_t \cdot \ell$ mixed zones. ◀

For the next observations we need the following notion. A *corner* in a matrix M is simply a mixed 2×2 submatrix which is an intersection of two consecutive rows with two consecutive columns. The following observation was pivotally used in the proof of Theorem 5 in [5].

▶ **Lemma 7** ([5])**.** *A matrix is mixed if and only if it contains a corner.*

The next lemma is essentially proven in [5] but never stated explicitly. So we include a proof for completeness.

▶ **Lemma 8** (implicit in [5])**.** *A t-mixed-free $n \times n$ matrix contains at most $2c_t(n+2)$ corners.*

Proof. Let M be a t-mixed-free $n \times n$ matrix. Consider the $\lceil n/2 \rceil$-division $(\mathcal{R}, \mathcal{C})$ of M, in which every row block consists of rows with indices $2i-1$ and $2i$ for some $i \in \{1, \ldots, \lfloor n/2 \rfloor\}$, possibly except the last block that consists only of row n in case n is odd, and similarly

for column blocks. By Lemma 6, $(\mathcal{R}, \mathcal{C})$ has at most $c_t \lceil n/2 \rceil \leq c_t(n/2+1)$ mixed zones, which implies that M has at most $c_t(n/2+1)$ corners in which the bottom-right entry is in the intersection of an even-indexed row and an even-indexed column. Call such corners of *type* 00; corners of types 01, 10, and 11 are defined analogously. By suitably modifying the pairing of rows and columns in $(\mathcal{R}, \mathcal{C})$, we can analogously prove that the number of corners of each of the remaining three types is also bounded by $c_t(n/2+1)$. Hence, in total there are at most $4c_t(n/2+1) = 2c_t(n+2)$ corners in M. ◀

Next, we will need a variant of Lemma 6 that focuses on mixed borders between neighboring zones. Here, two different zones in a division $(\mathcal{R}, \mathcal{C})$ are called *adjacent* if they are either in the same row block and consecutive column blocks, or in the same column block and consecutive row blocks. A *mixed cut* in $(\mathcal{R}, \mathcal{C})$ is a pair of adjacent zones such that there is a corner that crosses the boundary between them, i.e., has two entries in each of them. A *split corner* in $(\mathcal{R}, \mathcal{C})$ is a corner intersecting four different zones, i.e., it has an entry in four different zones.

The proof of the following observation is again essentially present in [5].

▶ **Lemma 9.** *Let M be a t-mixed free matrix, and let $(\mathcal{R}, \mathcal{C})$ be an ℓ-division of M, for some integer ℓ. Then $(\mathcal{R}, \mathcal{C})$ has at most $c_t \cdot (\ell+2)$ mixed cuts and at most $2c_t \cdot (\ell+1)$ split corners.*

Proof. Let $(\mathcal{R}^{00}, \mathcal{C}^{00})$ be the division obtained from $(\mathcal{R}, \mathcal{C})$ by merging the row blocks indexed $2i-1$ and $2i$ into a single row block, and merging the column blocks indexed $2i-1$ and $2i$ into a single column block, for each $i \in \{1, \ldots, \lfloor \ell/2 \rfloor\}$. Obtain divisions $(\mathcal{R}^{10}, \mathcal{C}^{10})$, $(\mathcal{R}^{01}, \mathcal{C}^{01})$, and $(\mathcal{R}^{11}, \mathcal{C}^{11})$ in a similar manner, where if the first number in the superscript is 1 then we merge row blocks $2i$ and $2i+1$ for each $i \in \{1, \ldots, \lceil \ell/2 \rceil - 1\}$ instead, and if the second number in the superscript is 1 then we merge column blocks $2i$ and $2i+1$ for each $i \in \{1, \ldots, \lceil \ell/2 \rceil - 1\}$ instead.

Observe that for every mixed cut of $(\mathcal{R}, \mathcal{C})$, the two zones in the mixed cut end up in the same zone in either $(\mathcal{R}^{00}, \mathcal{C}^{00})$ or in $(\mathcal{R}^{11}, \mathcal{C}^{11})$, rendering this zone mixed. However, by Lemma 6, $(\mathcal{R}^{00}, \mathcal{C}^{00})$ and $(\mathcal{R}^{11}, \mathcal{C}^{11})$ have at most $c_t \cdot (\ell/2+1)$ mixed zones. It follows that $(\mathcal{R}, \mathcal{C})$ has at most $2c_t \cdot (\ell/2+1) = c_t \cdot (\ell+2)$ mixed cuts. The bound on the number of split corners follows from the same argument combined with the observation that every split corner in $(\mathcal{R}, \mathcal{C})$ is entirely contained in a single zone of exactly one of divisions $(\mathcal{R}^{00}, \mathcal{C}^{00})$, $(\mathcal{R}^{10}, \mathcal{C}^{10})$, $(\mathcal{R}^{01}, \mathcal{C}^{01})$, and $(\mathcal{R}^{11}, \mathcal{C}^{11})$. ◀

Partitioning into rectangles. We conclude with another observation about twin-ordered matrices: they can be decomposed into a small number of rectangles. Formally, for a binary matrix M, a *rectangle decomposition* of M is a set $\mathcal{K}$ of pairwise disjoint rectangular submatrices (i.e., zones induced by some row block and some column block) such that every submatrix in $\mathcal{K}$ is entirely filled with 1s and there is no entry 1 outside the submatrices in $\mathcal{K}$. The following lemma is stated and proved in the graph setting in [2]; we adapt the proof here to the matrix setting.

▶ **Lemma 10.** *Let M be an $n \times n$ binary matrix that is d-twin-ordered. Then M admits a rectangle decomposition $\mathcal{K}$ with $|\mathcal{K}| \leq d(2n-2)+1$.*

Proof. Let $(\mathcal{R}_0, \mathcal{C}_0), \ldots, (\mathcal{R}_{2n-2}, \mathcal{C}_{2n-2})$ be a contraction sequence for M with error value at most d. Let $\mathcal{S}_i$ be the set of zones of $(\mathcal{R}_i, \mathcal{C}_i)$, and let

$$\mathcal{S} = \bigcup_{i=0}^{2n-2} \mathcal{S}_i.$$

Note that $\mathcal{S}$ is a *laminar* family, that is, every two submatrices in $\mathcal{S}$ are either disjoint or one is contained in the other.

Let $\mathcal{K}$ be the subfamily of $\mathcal{S}$ consisting of those submatrices that are entirely filled with 1s, and are inclusion-wise maximal in $\mathcal{S}$ subject to this property. Note that every entry 1 in M is contained in some member of $\mathcal{K}$, for the zone of $(\mathcal{R}_0, \mathcal{C}_0)$ in which this entry is contained is a 1×1 submatrix entirely filled with 1s. Since $\mathcal{S}$ is laminar, it follows that $\mathcal{K}$ is a rectangle decomposition of M. So it remains to argue that $|\mathcal{K}| \leq d(2n-2)+1$.

Consider any $A \in \mathcal{K}$ and let i be the largest index such that $A \in \mathcal{S}_i$. We may assume that $i < 2n-2$, for otherwise the matrix M is entirely filled with 1s and the postulated claim is trivial. By maximality, A is contained in a non-constant zone $B \in \mathcal{S}_{i+1}$ that resulted from merging A with another adjacent zone $A' \in \mathcal{S}_i$, which is not entirely filled with 1s. In particular, B lies in the unique row block or column block of $(\mathcal{R}_{i+1}, \mathcal{C}_{i+1})$ that resulted from merging two row blocks or two column blocks of $(\mathcal{R}_i, \mathcal{C}_i)$. There can be at most d non-constant zones in this row/column block of $(\mathcal{R}_{i+1}, \mathcal{C}_{i+1})$, and B is one of them. We infer that i can be the largest index satisfying $A \in \mathcal{S}_i$ for at most d different submatrices $A \in \mathcal{K}$. Since this applies to every index $i \in \{0, 1, \ldots, 2n-3\}$, we conclude that $|\mathcal{K}| \leq d(2n-2)$. ◀

Observe that Lemma 10 provides a way to encode an $n \times n$ d-twin-ordered matrix in $\mathcal{O}_d(n \log n)$ bits: one only needs to specify the vertices of the submatrices of a rectangle decomposition of size $\mathcal{O}_d(n)$. The proof is also effective, in the sense that given a suitably represented contraction sequence one can compute the obtained decomposition $\mathcal{K}$. To abstract away the nuances of representing contraction sequences, throughout this paper we assume that d-twin-ordered matrices are provided on input through suitable rectangle decompositions.

3 Structural properties of divisions

Before we proceed to constructing the promised compact representation, we need to describe some new combinatorial properties of twin-ordered matrices. For the remainder of this section, we fix $d \in \mathbb{N}$ and consider a matrix M that is d-twin-ordered. In particular, by Theorem 5, M is $(2d+2)$-mixed-free.

Strips. We begin by considering non-constant vertical and horizontal zones of a given division of M. We will show that these zones can be grouped into $\mathcal{O}_d(t)$ *strips* that again are vertical or horizontal, respectively. This partitioning is formalized as follows.

▶ **Definition 11.** *Let $(\mathcal{R}, \mathcal{C})$ be a division of a matrix M. A* vertical strip *in $(\mathcal{R}, \mathcal{C})$ is an inclusion-wise maximal set of non-constant vertical zones of $\mathcal{D}$ that are contained in the same column block of $(\mathcal{R}, \mathcal{C})$, span a contiguous interval of row blocks, and whose union is again a vertical submatrix.* Horizontal strips *are defined analogously.*

1	1	1	1	1	1	1	0	0	1	1
0	0	0	0	0	0	0	0	0	1	1
1	1	1	1	1	1	1	0	0	1	1
1	1	1	1	1	1	1	0	0	1	1
0	0	0	1	1	1	1	0	0	0	1
0	0	0	0	0	1	0	0	0	0	1
1	1	1	1	1	1	1	0	0	0	1
0	0	0	0	0	1	0	0	0	0	1

Figure 1 Strips in an example 4-division of a matrix. Horizontal strips are painted in shades of yellow. Vertical strips are painted in shades of blue. Unpainted zones are constant or mixed.

Naturally, each non-constant vertical zone belongs to exactly one vertical strip; and similarly, each non-constant horizontal zone belongs to exactly one horizontal strip.

We will now show an upper bound on the number of vertical and horizontal strips present in any t-division of M.

▶ **Lemma 12.** *For every $t \in \mathbb{N}$, the total number of vertical and horizontal strips in any t-division of M is at most $\mathcal{O}_d(t)$.*

Proof. We focus on the bound for vertical strips only; the proof for horizontal strips is symmetric. Fix some t-division $(\mathcal{R}, \mathcal{C})$ of M. Observe that each vertical strip S of the division either intersects the top row of the matrix, or the top-most zone of S is adjacent from the top to another zone C such that adding C to S yields a submatrix that is not vertical. (We say that C is adjacent to S *from the top*.) Thus, we partition the family of vertical strips in the t-division of M into three types:

- **(I)** strips intersecting the top row of M;
- **(II)** strips adjacent to a mixed zone C from the top; and
- **(III)** strips adjacent to a non-mixed zone C from the top.

Obviously, there are at most t vertical strips of type (I). Next, each vertical strip of type (II) can be assigned a private mixed zone C adjacent to it from the top. Hence, the number of vertical strips of this type is upper bounded by the number of mixed zones in $(\mathcal{R}, \mathcal{C})$, which by Lemma 6 is bounded by $\mathcal{O}_d(t)$.

Finally, let us consider vertical strips of type (III). Let S be a vertical strip of this type, D be its top-most zone, and C be the non-mixed zone adjacent to D from the top. Since D is vertical, all rows of D are repetitions of the same row vector v_D. Since D is non-constant, v_D is non-constant as well.

As C is non-mixed, it is either horizontal or vertical. If C is vertical, then all its rows are repetitions of the same row vector v_C. Observe that since strip S could not be extended by C, we have $v_C \neq v_D$. Now, as v_D is non-constant, it follows that the union of the bottom-most row of C and the top-most row of D contains a corner. On the other hand, if C is horizontal, then the bottom-most row of C is constant and again there is a corner in the union of the (constant) bottom-most row of C and the (non-constant) top-most row of D.

So in both cases we conclude that C and D form a mixed cut. By Lemma 9, the total number of mixed cuts in $(\mathcal{R}, \mathcal{C})$ is bounded by $\mathcal{O}_d(t)$, so also there are at most $\mathcal{O}_d(t)$ vertical strips of type (III). This concludes the proof. ◀

Regular divisions. We move our focus to a central notion of our data structure: *regular divisions* of a matrix:

▶ **Definition 13.** *Given M and an integer $s \in \mathbb{N}$, we define the* s-regular division *of M as the $\lceil \frac{n}{s} \rceil$-division of M in which each row block (respectively, column block), possibly except the last one, contains s rows (resp. columns). Precisely, if $s \nmid n$, then the last row block and the last column block contain exactly $n \bmod s$ rows or columns, respectively.*

In the data structure, given a square input matrix M, we will construct multiple regular divisions of M of varying granularity (the value of s). Crucially, in order to ensure the space efficiency of the data structure, we will require that the number of *distinct* zones in each such regular division of M should be small. This is facilitated by the following definition:

▶ **Definition 14.** *For $s \in \mathbb{N}$, the* s-zone family *of M, denoted $\mathcal{F}_s(M)$, is the set of all different zones participating in the s-regular division of M.*

Let us stress that we treat $\mathcal{F}_s(M)$ as a set of matrices and do not keep duplicates in it. That is, if the regular s-division of M contains two or more isomorphic zones – with same dimensions and equal corresponding entries – then these zones are represented in $\mathcal{F}_s(M)$ only once.

For the remainder of this section, we will prove good bounds on the cardinality of $\mathcal{F}_s(M)$. Trivially, the cardinality of $\mathcal{F}_s(M)$ is bounded by $\lceil \frac{n}{s} \rceil^2$ (i.e., the number of zones in the s-regular division). Also, the same cardinality is trivially bounded by $2^{\mathcal{O}(s^2)}$ (i.e., the total number of distinct matrices with at most s rows and columns). However, given that M is d-twin-ordered, both bounds can be improved dramatically. First, the dependence on $\frac{n}{s}$ in the former bound can be improved to linear:

▶ **Lemma 15.** *For every $s \in \{1, \ldots, n\}$, the cardinality of $\mathcal{F}_s(M)$ is bounded by $\mathcal{O}_d(\frac{n}{s})$.*

Proof. First assume that $s \mid n$; hence, each zone in the s-regular division of M has s rows and s columns. Then, the matrices in $\mathcal{F}_s(M)$ can be categorized into four types:

- Constant zones. There are at most 2 of them – constant 0 and constant 1.
- Mixed zones. Here, Lemma 6 applies directly: since the considered division is an $\frac{n}{s}$-division of M, there are at most $\mathcal{O}_d(\frac{n}{s})$ mixed zones in M in total.
- Vertical zones. By Lemma 12, all vertical zones of the considered division can be partitioned into $\mathcal{O}_d(\frac{n}{s})$ vertical strips. As all zones have the same dimensions, the zones belonging to a single vertical strip are pairwise isomorphic. From this we infer the $\mathcal{O}_d(\frac{n}{s})$ upper bound on the number of different vertical zones.
- Horizontal zones are handled symmetrically to vertical zones.

Finally, if $s \nmid n$, then let M' be equal to M, truncated to the first $n - (n \bmod s)$ rows and columns; equivalently, M' is equal to M with all zones with fewer than s rows or columns removed. The argument given above applies to M', yielding at most $\mathcal{O}_d(\frac{n}{s})$ different $s \times s$ zones in M' (and equivalently in M). The proof is concluded by the observation that M contains exactly $2\lceil \frac{n}{s} \rceil - 1 = \mathcal{O}(\frac{n}{s})$ zones in its s-regular division that have fewer than s rows or columns. ◀

Second, from the works of Bonnet et al. [1, 3] one can easily derive an upper bound that is exponential in s rather than in s^2:

▶ **Lemma 16.** *For every $s \in \{1, \ldots, n\}$, the cardinality of $\mathcal{F}_s(M)$ is bounded by $2^{\mathcal{O}_d(s)}$.*

Proof. Observe that a submatrix of a d-twin-ordered matrix is also d-twin-ordered. Thus, it is only necessary to upper bound the total number of different $s \times s$ matrices that are d-twin-ordered. To this end, we use the notion of twin-width of ordered binary relational structures introduced in the work of Bonnet et al. [3]. This notion is more general than twin-orderedness in the following sense: each $s \times s$ matrix that is d-twin-ordered corresponds to a different ordered binary structure over s elements of twin-width at most d. As proved in [3], the number of different such structures is upper bounded by $2^{\mathcal{O}_d(s)}$. The claim follows. ◀

While the bound postulated by Lemma 15 is more powerful for coarse regular divisions of M (i.e., s-regular divisions for large s), Lemma 16 yields a better bound for $s \leq p_d \cdot \log n$, where $p_d > 0$ is a sufficiently small constant depending on d.

4 Data structure

In this section we present the data structure promised in Theorem 1. Recall that it should represent a given binary $n \times n$ matrix M that is d-twin-ordered, and it should provide access to the following query: for given $(i, j) \in [n]^2$, return the entry $M[i, j]$. Here we focus only

on the description of the data structure, implementation of the query, and analysis of the bitsize. The construction algorithm promised in Theorem 1 is given in the full version of the work [14].

Without loss of generality, we assume that n is a power of 2. Otherwise we enlarge M, so that its order is the smallest power of 2 larger than n. We use dummy 0s to fill additional entries. It is straightforward to see that the resulting matrix is $(d+1)$-twin-ordered. Similarly, in the analysis we may assume that n is sufficiently large compared to any constants present in the context.

Description. Our data structure consists of $\ell+1$ layers: $\mathcal{L}_0, \ldots, \mathcal{L}_\ell$. Recall from Definition 14 that $\mathcal{F}_s(M)$ is the family of pairwise different zones participating in the s-regular division of M. Each layer $\mathcal{L}_i$ in our data structure corresponds to $\mathcal{F}_{m_i}(M)$ for a carefully chosen parameter m_i. Let $\mathsf{low}(x)$ be the largest power of 2 smaller or equal to x. We define parameters m_i inductively as follows: set $m_0 = n$ and for $i \geq 0$,

$$m_{i+1} = \begin{cases} \mathsf{low}({m_i}^{2/3}) & \text{if } m_i \geq \log^3 n \\ m_i/2 & \text{if } \log n/(2\beta_d) \leq m_i < \log^3 n \end{cases}$$

where β_d is the constant hidden in the $\mathcal{O}_d(\cdot)$ notation in Lemma 16, i.e., $|\mathcal{F}_s(M)| \leq 2^{\beta_d \cdot s}$. The construction stops when we reach m_i satisfying $m_i < \log n/(2\beta_d)$, in which case we set $\ell = i$. Note that all parameters m_i are powers of 2, so m_j divides m_i whenever $i \leq j$.

We also observe the following.

▷ Claim 17. $\ell \in \mathcal{O}(\log \log n)$.

Proof. Let k be the least index for which $m_k < \log^3 n$. Observe that for $i \in [1, k]$ we have $m_i \leq n^{(2/3)^i}$. So it must be that $k \leq \log_{3/2} \log n + 1 \in \mathcal{O}(\log \log n)$, for otherwise we would have $m_{k-1} \leq n^{(2/3)^{\log_{3/2} \log n}} = n^{1/\log n} = 2 < \log^3 n$. Next, observe that for $i \in [k+1, \ell]$ we have $m_i = m_k/2^{i-k}$. Therefore, we must have $\ell - k \leq \log(\log^3 n) + 1 \in \mathcal{O}(\log \log n)$, for otherwise we have $m_{\ell-1} \leq m_k/2^{\log \log^3 n} < \log^3 n/\log^3 n = 1$. The claim follows. ◁

Layer $\mathcal{L}_\ell$ is special and we describe it separately, so let us now describe the content of layer $\mathcal{L}_i$ for each $i < \ell$. Since n is divisible by m_i, every $Z \in \mathcal{F}_{m_i}(M)$ is an $m_i \times m_i$ matrix that appears at least once as a zone in the (n/m_i)-regular division of M. Such Z will be represented by an object $\mathsf{obj}(Z)$ in $\mathcal{L}_i$. Each object $\mathsf{obj}(Z)$ stores $(m_i/m_{i+1})^2$ pointers to objects in $\mathcal{L}_{i+1}$; recall here that m_{i+1} divides m_i. Consider the m_{i+1}-regular division of Z. This division consists of $(m_i/m_{i+1})^2$ zones; index them as $\mathsf{subzone}_Z(i,j)$ for $i,j \in [m_i/m_{i+1}]$ naturally. Observe that for all $i,j \in [m_i/m_{i+1}]$, it holds that $\mathsf{subzone}_Z(i,j) \in \mathcal{F}_{m_{i+1}}(M)$. In our data structure, each object $\mathsf{obj}(Z) \in \mathcal{L}_i$, corresponding to a matrix $Z \in \mathcal{F}_{m_i}(M)$, stores an array ptr of $(m_i/m_{i+1})^2$ pointers, where $\mathsf{ptr}[i,j]$ points to the address of $\mathsf{subzone}_Z(i,j)$ for all $i,j \in [m_i/m_{i+1}]$. This concludes the description of layer $\mathcal{L}_i$ for $i < \ell$.

We now describe layer $\mathcal{L}_\ell$. It is also a collection of objects, and for each matrix $Z \in \mathcal{F}_{m_\ell}(M)$ there is an object $\mathsf{obj}(Z) \in \mathcal{L}_\ell$; these objects are pointed to by objects from $\mathcal{L}_{\ell-1}$. However, instead of storing further pointers, each object $\mathsf{obj}(Z) \in \mathcal{L}_\ell$ stores the entire matrix $Z \in \mathcal{F}_{m_\ell}(M)$ as a binary matrix of order $m_\ell \times m_\ell$, using m_ℓ^2 bits. This concludes the description of $\mathcal{L}_\ell$.

Observe that in $\mathcal{L}_0$ there is only one object corresponding to the entire matrix M. We store a global pointer ptrGlo to this object. Our data structure is accessed via ptrGlo upon each query.

Implementation of the query. The description of the data structure is now complete and we move on to describing how the query is executed. The query is implemented as method $\mathsf{entry}(i,j)$ and returns $M[i,j]$; see Algorithm 1 for the pseudocode (where $\mathsf{ptrIt} \rightarrow$ stands for dereference of a pointer ptrIt, i.e., the object pointed to by ptrIt). Given two integers $i,j \in [n]$, the method starts with pointer ptrGlo, and uses i and j and iterator pointer ptrIt to navigate via pointers down the layers, ending with a pointer to an object in layer $\mathcal{L}_\ell$. Initially, the iterator ptrIt is set to ptrGlo and it points to $\mathsf{obj}(Z)$ for the only matrix $Z \in \mathcal{F}_{m_0}(M)$. Integers i,j are the positions of the desired entry with respect to zone Z. After a number of iterations, ptrIt points to an object $\mathsf{obj}(Z) \in \mathcal{L}_k$ for a matrix $Z \in \mathcal{F}_{m_k}(M)$, and maintains current coordinates i and j. The invariant is that the desired output is the entry $Z[i,j]$. In one step of the iteration, the algorithm finds the matrix Z' in $\mathcal{F}_{m_{k+1}}(M)$ containing the desired entry $Z[i,j]$, which is the zone $\mathsf{subzone}_Z(i \operatorname{\mathsf{div}} m_{k+1}, j \operatorname{\mathsf{div}} m_{k+1}) \in \mathcal{F}_{m_{k+1}}(M)$, and moves the pointer ptrIt to $\mathsf{obj}(Z') \in \mathcal{L}_{k+1}$. The new coordinates of the desired entry with respect to Z' are $(i \bmod m_{k+1})$ and $(j \bmod m_{k+1})$, so i and j are altered accordingly. Once the iteration reaches $\mathcal{L}_\ell$, the object pointed to by ptrIt contains the entire zone explicitly, so it suffices to return the desired entry. Obviously, the running time of the query is $\mathcal{O}(\log \log n)$, since the algorithm iterates through $\ell \in \mathcal{O}(\log \log n)$ layers.

Algorithm 1 Query algorithm.

Input : Integers $i,j \in [n]$
Output : $M[i,j]$

```
ptrIt ← ptrGlo
for k ← 0 to ℓ − 1 do
    ptrIt ← (ptrIt → ptr[i div m_{k+1}, j div m_{k+1}] ;
    i ← i mod m_{k+1} ;
    j ← j mod m_{k+1} ;
return ptrIt → Z[i, j]
```

Analysis of bitsize. We now analyze the number of bits occupied by the data structure. First note that the total number of objects stored is bounded by the total number of submatrices of M, which is polynomial in n. Hence, every pointer can be represented using $\mathcal{O}(\log n)$ bits. Keeping this in mind, the total bitsize occupied by the data structure is proportional to

$$\sum_{i=0}^{\ell-1} |\mathcal{F}_{m_i}(M)| \left(\frac{m_i}{m_{i+1}}\right)^2 \log n + |\mathcal{F}_{m_\ell}(M)| m_\ell^2, \tag{1}$$

This is because for all layers $\mathcal{L}_i$ for $i < \ell$ we store $|\mathcal{F}_{m_i}(M)|$ objects, each storing $\left(\frac{m_i}{m_{i+1}}\right)^2$ pointers, and in $\mathcal{L}_\ell$ we store $|\mathcal{F}_{m_\ell}(M)|$ objects, each storing a binary matrix of order $m_\ell \times m_\ell$.

We first bound the second term of Equation (1). By Lemma 16, we have

$$|\mathcal{F}_{m_\ell}(M)| m_\ell^2 \leq 2^{\beta_d \cdot m_\ell} \cdot m_\ell^2 \leq 2^{\beta_d \cdot \frac{\log n}{2\beta_d}} \cdot \left(\frac{\log n}{2\beta_d}\right)^2 = \sqrt{n} \cdot \left(\frac{\log n}{2\beta_d}\right)^2 \in o(n).$$

We move on to bounding the first term of Equation (1). Let k be the least index for which $m_k < \log^3 n$. We can split the first term of Equation (1) into two sums:

$$\sum_{i=0}^{\ell-1} |\mathcal{F}_{m_i}(M)| \left(\frac{m_i}{m_{i+1}}\right)^2 \log n =$$

$$= \sum_{i=0}^{k-1} |\mathcal{F}_{m_i}(M)| \left(\frac{m_i}{m_{i+1}}\right)^2 \log n + \tag{2}$$

$$+ \sum_{i=k}^{\ell-1} |\mathcal{F}_{m_i}(M)| \left(\frac{m_i}{m_{i+1}}\right)^2 \log n \tag{3}$$

We first apply Lemma 15 to bound the sum (2). More precisely, if α_d is the constant hidden in the $\mathcal{O}_d(\cdot)$ notation in Lemma 15, we have

$$(2) \leq \log n \cdot \sum_{i=0}^{k-1} \alpha_d \frac{n}{m_i} \cdot 4m_i^{2/3} = 4\alpha_d n \log n \cdot \sum_{i=0}^{k-1} \frac{1}{m_i^{1/3}} \tag{4}$$

Since for $i \in [k-1]$ we have $m_{i+1} = \mathsf{low}(m_i^{2/3})$ and $m_i \geq \log^3 n$, we have $m_i/m_{i+1} \geq 2$. Therefore $m_i \geq 2^{k-i-1} m_{k-1}$ for $i \in [0, k-1]$, so we can continue bounding the last expression in Equation (4):

$$(4) \leq \alpha_d n \log n \sum_{i=0}^{k-1} \cdot \frac{1}{(2^{k-i-1} m_{k-1})^{1/3}} \leq \alpha_d n \cdot \frac{\log n}{m_{k-1}^{1/3}} \cdot \sum_{i=0}^{k-1} \frac{1}{(2^{k-i-1})^{1/3}} \in \mathcal{O}_d(n).$$

It remains to bound sum (3). We use Lemma 15 similarly as above:

$$(3) = 4 \log n \cdot \sum_{i=k}^{\ell-1} |\mathcal{F}_{m_i}(M)| \leq 4\alpha_d n \log n \cdot \sum_{i=k}^{\ell-1} \frac{1}{m_i} \leq 4\alpha_d n \log n \cdot \sum_{i=0}^{\infty} \frac{1}{(\frac{\log n}{2\beta_d}) \cdot 2^i} \in \mathcal{O}_d(n).$$

By summing up all the bounds we infer that the total number of bits occupied by our data structure is $\mathcal{O}_d(n)$.

References

1 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width II: small classes. In *2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 1977–1996. SIAM, 2021. doi:10.1137/1.9781611976465.118.

2 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width III: Max Independent Set, Min Dominating Set, and Coloring. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 35:1–35:20. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2021.

3 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Toruńczyk. Twin-width IV: ordered graphs and matrices. *CoRR*, abs/2102.03117, 2021. arXiv:2102.03117.

4 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, Stéphan Thomassé, and Rémi Watrigant. Twin-width and polynomial kernels. *CoRR*, abs/2107.02882, 2021. arXiv:2107.02882.

5 Édouard Bonnet, Eun Jung Kim, Stéphan Thomasse, and Rémi Watrigant. Twin-width I: tractable FO model checking. In *IEEE 61st Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 601–612. IEEE Computer Society, 2020.

6 Édouard Bonnet, Jaroslav Nešetřil, Patrice Ossona de Mendez, Sebastian Siebertz, and Stéphan Thomassé. Twin-width and permutations. *CoRR*, abs/2102.06880, 2021. arXiv:2102.06880.

7 Timothy M. Chan. Persistent Predecessor Search and Orthogonal Point Location on the Word RAM. *ACM Trans. Algorithms*, 9(3):22:1–22:22, 2013.

8 Josef Cibulka and Jan Kynčl. Better upper bounds on the Füredi-Hajnal limits of permutations, 2019. arXiv:1607.07491.

9 Jan Dreier, Jakub Gajarský, Yiting Jiang, Patrice Ossona de Mendez, and Jean-Florent Raymond. Twin-width and generalized coloring numbers. *CoRR*, abs/2104.09360, 2021. arXiv:2104.09360.

10 Jakub Gajarský, Michał Pilipczuk, and Szymon Toruńczyk. Stable graphs of bounded twin-width. *CoRR*, abs/2107.03711, 2021. arXiv:2107.03711.

11 Shahin Kamali. Compact representation of graphs of small clique-width. *Algorithmica*, 80(7):2106–2131, 2018.

12 Adam Marcus and Gábor Tardos. Excluded permutation matrices and the Stanley–Wilf conjecture. *Journal of Combinatorial Theory, Series A*, 107(1):153–160, 2004.

13 Mihai Pătraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *38th Annual ACM Symposium on Theory of Computing, STOC 2006*, pages 232–240. ACM, 2006.

14 Michał Pilipczuk, Marek Sokołowski, and Anna Zych-Pawlewicz. Compact representation for matrices of bounded twin-width. *CoRR*, abs/2110.08106, 2021. arXiv:2110.08106.

A Representation with bitsize $\mathcal{O}(n^{1+\varepsilon})$ and query time $\mathcal{O}(1/\varepsilon)$

In this section we provide a brief sketch of another data structure representing twin-ordered matrices. For any fixed $\varepsilon > 0$, we will construct a data structure that represents a given d-twin-ordered $n \times n$ matrix M in bitsize $\mathcal{O}(n^{1+\varepsilon})$, and can be queried for entries of M in worst-case time $\mathcal{O}(1/\varepsilon)$ per query.

Actually, the data structure solves the Orthogonal Point Location problem. An instance of the problem is a set of $\mathcal{O}(n)$ orthogonal rectangles with pairwise disjoint interiors, each with integer coordinates between 0 and n. In the problem, we are required to preprocess the input rectangles and construct a data structure that can can efficiently locate the rectangle containing a given query point. As the set of 1 entries in any d-twin-ordered matrix M admits a rectangle decomposition into $\mathcal{O}_d(n)$ rectangles (Lemma 10), this also yields a data structure representing M.

Famously, Chan [7] designed a data structure for Orthogonal Point Location that can answer each query in worst-case time $\mathcal{O}(\log\log n)$ and can be constructed in time $\mathcal{O}(n \log\log n)$. In the same work, he also observed that achieving constant query time is much more difficult. Namely, Orthogonal Point Location can be reduced to the static variant of the Predecessor Search problem. Pătraşcu and Thorup proved that each data structure for Predecessor Search with $\mathcal{O}(n \log^{\mathcal{O}(1)} n)$ bitsize necessarily requires $\Omega(\log\log n)$ query time, even in a much more powerful cell probe model [13]. Therefore, for general Orthogonal Point Location, one cannot expect to achieve constant query time with bitsize significantly smaller than $\mathcal{O}(n^{1+\varepsilon})$.

Data structure for disjoint intervals. Consider integers $k, h \geq 1$, and let $n = k^h$. We will first sketch a data structure that maintains a set of disjoint integer intervals that are subintervals of $[0, n-1]$. The data structure shall allow adding or removing intervals in time $\mathcal{O}(kh)$ and querying whether a point is contained in any interval in time $\mathcal{O}(h)$.

Consider a perfect k-ary tree of depth h. The tree has k^h leaves, numbered from 0 to $n-1$ according to the pre-order traversal of the tree. Each internal node at depth $i \in \{0, 1, \ldots, h-1\}$ in the tree corresponds to a contiguous interval of leaves of length k^{h-i}. Each such interval is called a *base interval*. Each internal node contains an array of k pointers to the children in the tree, allowing access to the j-th child in constant time. Additionally, alongside each node v of the data structure, we store an additional bit b_v, initially set to 0.

Assume an interval $[\ell, r]$ is to be inserted to the set. We traverse the tree recursively, starting from the root, entering only nodes whose base intervals intersect $[\ell, r]$, and cutting the recursion at nodes whose base intervals are entirely within $[\ell, r]$. It can be shown that the recursion visits at most $\mathcal{O}(kh)$ nodes and decomposes $[\ell, r]$ into $\mathcal{O}(kh)$ disjoint base intervals. For each node v corresponding to such a base interval, we set $b_v \leftarrow 1$. Removing an interval from the set is analogous. Now, to verify whether an element y belongs to the set, we descend recursively from the root of the tree to the y-th leaf of the tree and verify if any of the visited nodes v has $b_v = 1$. This requires time $\mathcal{O}(h)$.

Since each update and query to the data structure is essentially a recursive search from the root of the tree, the data structure can be made persistent: on each update, we create a copy of each altered node and each of their ancestors, and we reset the pointers in the copies accordingly. As $\mathcal{O}(kh)$ nodes are updated at each query, and each internal node stores an array of $\mathcal{O}(k)$ pointers, the update time increases to $\mathcal{O}(k^2h)$ due to the copying of the nodes; and each update increases the bitsize of the data structure by $\mathcal{O}(k^2h \log n)$. Thus, after $\mathcal{O}_d(n)$ updates, the bitsize of the data structure is $\mathcal{O}_d(nk^2h \log n)$. The query time remains at $\mathcal{O}(h)$.

Orthogonal point location with small coordinates. Fix any $\varepsilon > 0$. Given a matrix M of order n, we set $h \coloneqq \lceil 2/\varepsilon \rceil + 1$ and $k \coloneqq \lceil n^{1/h} \rceil$. We instantiate a persistent k-ary tree of depth h as above. We sweep the set of rectangles from the left of the right, maintaining a vertical sweep line. The tree maintains an intersection of the sweep line with the union of rectangles as a set of disjoint intervals contained in $[0, n]$. Hence, for each rectangle, the tree is updated twice: a vertical interval is added when the sweep line reaches the left end of the rectangle, and is removed as soon as it reaches the right end of the rectangle. At each x coordinate, we store the pointer ver_x to the root of the current version of the tree. After the preprocessing, for each query (x, y), we fetch the pointer ver_x and check whether this version of the tree contains y as an element.

Let us analyze the query time and the bitsize of the data structure. For convenience, let $\delta \coloneqq 1/h$. We can see that $0 < \delta < \frac{\varepsilon}{2}$. Each query is performed in time $\mathcal{O}(h) = \mathcal{O}(1/\varepsilon)$. Storing pointers ver_x requires bitsize $\mathcal{O}(n \log n)$. Since we processed $\mathcal{O}_d(n)$ rectangles, the persistent tree has bitsize $\mathcal{O}_d(nk^2h \log n) = \mathcal{O}_d(n^{1+2\delta} \log n/\varepsilon) = \mathcal{O}_d(n^{1+\varepsilon})$.