Multiple Criss-Cross Deletion-Correcting Codes

Lorenz Welter, **Rawad Bitar**, **Antonia Wachter-Zeh**, and **Eitan Yaakobi**

***Abstract*—This paper investigates the problem of correcting multiple criss-cross deletions in arrays. More precisely, we study the unique recovery of $n \times n$ arrays affected by any combination of t_{r} row and t_{c} column deletions such that $t_{\mathrm{r}} + t_{\mathrm{c}} = t$ for a given t. We refer to these type of deletions as *t-criss-cross deletions*. We show that the asymptotic redundancy of a code correcting t-criss-cross deletions is at least $tn + t\log n - \log(t!)$. Then, we present an existential construction of a code capable of correcting t-criss-cross deletions where its redundancy is bounded from above by $tn + \mathcal{O}(t^2 \log^2 n)$. The main ingredients of the presented code are systematic binary t-deletion-correcting codes and Gabidulin codes. The first ingredient helps locating the indices of the deleted rows and columns, thus transforming the deletion-correction problem into an erasure-correction problem which is then solved using the second ingredient.**

I. Introduction

Deletion-correcting codes have recently witnessed an increased attention due to their application in DNA-based storage systems, file synchronization, and communication systems [1]–[7]. The problem of correcting deletions dates back to the 1960s. In [8], Levenshtein defined the notion of t-deletion-correcting codes and bounded from below the redundancy of any binary t-deletion-correcting code by $t\log n - \mathcal{O}(1)$. Moreover, he proved that the Varshamov-Tenengolts codes [9], originally designed to correct a single asymmetric error, can also correct a single deletion and have redundancy of roughly $\log(n+1)$ bits. Several recent works studied the problem of constructing binary t-deletion-correcting codes, for $t > 1$, with redundancy approaching Levenshtein's bound [10]–[16]. Of particular importance to us is the work of Sima *et al.* [17] in which the authors present a binary systematic t-deletion-correcting code with redundancy $4t\log(n) + o\log(n)$.

This paper considers the problem of coding for deletions in the two-dimensional space. The motivation stems from the two-dimensional erasure and substitution correction where it has been shown that leveraging the structure of the array is more beneficial than applying one dimensional error correcting codes on each dimension of the array. The deletion correction problem is however more involved due to the loss of synchronization in the locations of the deleted rows and columns. Along this line of thought, the trace-reconstruction problem, which is related to coding for deletions, is investigated for the two-dimensional space in [18]. Given a certain number of deletions t and an array $\mathbf{X}$, we assume that the array can be affected by any combination of t_{r} row and t_{c} column deletions such that $t_{\mathrm{r}} + t_{\mathrm{c}} = t$. This type of deletions are referred to as *t-criss-cross deletions*. Our goal is to construct codes that can uniquely recover the array $\mathbf{X}$ from any t-criss-cross deletion and we refer to these codes as *t-criss-cross codes*. We borrow this terminology from previous works that studied the problem of correcting criss-cross erasures and substitution errors in the two-dimensional space, e.g., [19]–[26].

The first works to study this problem were [27] and [28]. In [27], we investigated the problem of correcting exactly one row and one column deletion in arrays. We showed that the redundancy for this special case is bounded from below by $2n + 2\log n - \mathcal{O}(1)$ and presented an existential and an explicit construction with redundancy approximately $2\log n$ and $7\log n$ far from the lower bound, respectively. In [28], Hagiwara constructed codes for the problem of correcting criss-cross deletions with at most t_{r} row deletions and at most t_{c} column deletions, for given values of t_{r} and t_{c}. His construction splits the array into locators and information part. The locators are carefully structured arrays that can exactly recover the index of any deleted rows and columns in the array. Then, a tensor-product erasure-correcting code is used to recover the lost symbols in the information part.

Our contributions can be summarized as follows. We present an asymptotic upper bound (in the code length) on the cardinality of t-criss-cross codes. Our bound implies that the redundancy of any t-criss-cross code is bounded from below by approximately $tn + t\log n$. Then, we construct existential t-criss-cross codes based on locator arrays, binary systematic t-deletion correcting codes, and Gabidulin codes. The main improvement is to use a collection of binary deletion-correcting codes to locate the indices of the deleted columns and rows with less redundancy as compared to the locator arrays used in [28]. However, small locator arrays are still needed to complement the deletion-correcting codes. Then, the deletion-correction problem is transformed into a row/column erasure-correction problem which can be solved by using Gabidulin codes that have optimal redundancy for row/column erasure-correction [20]. The redundancy of the presented construction is $tn + \mathcal{O}(t^2\log^2 n)$. For the considered problem setting, we substantially improve upon the current state-of-the-art construction of [28] that needs a redundancy of approximately $2n \cdot (t^2 + t\log n)$.

II. Definitions and Preliminaries

This section formally defines the codes and notations that are used throughout this paper. Let $\Sigma \triangleq \{0,1\}$ be the binary alphabet. We denote by $\Sigma^{n\times n}$ the set of all binary arrays of dimension $n \times n$. All logarithms are base 2 unless otherwise indicated.

LW, RB and AW-Z are with the Institute for Communications Engineering, Technical University of Munich (TUM), Germany. Emails: {rawad.bitar, lorenz.welter, antonia.wachter-zeh}@tum.de.

EY is with the CS department of Technion — Israel Institute of Technology, Israel. Email: yaakobi@cs.technion.ac.il.

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 801434) and from the Technical University of Munich - Institute for Advanced Studies, funded by the German Excellence Initiative and European Union Seventh Framework Programme under Grant Agreement No. 291763.

978-1-5386-8209-8/21/$31.00 ©2021 IEEE

For an integer $n \in \mathbb{N}$, the set $\{1, \ldots, n\}$ is denoted by $[n]$. For an array $\mathbf{X} \in \Sigma^{n \times n}$ and $i, j \in [n]$, we refer to the entry of $\mathbf{X}$ positioned at the i^{th} row and the j^{th} column by $X_{i,j}$. We denote the i^{th} row and the j^{th} column of $\mathbf{X}$ by $\mathbf{X}_{i,[n]}$ and $\mathbf{X}_{[n],j}$, respectively. Similarly, we denote by $\mathbf{X}_{[i_1:i_2],[j_1:j_2]}$ the subarray of $\mathbf{X}$ formed by rows i_1 to i_2 and their corresponding entries from columns j_1 to j_2. Moreover, for two arrays $\mathbf{X} \in \Sigma^{n \times m_1}$ and $\mathbf{Y} \in \Sigma^{n \times m_2}$ we denote by $\mathbf{Z} = (\mathbf{X} \mid \mathbf{Y})$ the concatenation of these two arrays with $\mathbf{Z} \in \Sigma^{n \times (m_1 + m_2)}$. For any binary array $\mathbf{X}$, we refer to the complement of $\mathbf{X}$, i.e., every bit in $\mathbf{X}$ is flipped, by $\overline{\mathbf{X}}$.

For a positive integer t, we define a *t-criss-cross deletion* in a binary array $\mathbf{X}$ to be the deletion of any combination of t_{r} rows and t_{c} columns of $\mathbf{X}$ such that $t_{\text{r}} + t_{\text{c}} = t$. We refer to $\widetilde{\mathbf{X}}$ as the array resulting from a t-criss-cross deletion in $\mathbf{X}$, where the number of deletions that happened in $\mathbf{X}$ is clear from the context. A code $\mathcal{C} \subseteq \Sigma^{n \times n}$ that can correct any t-criss-cross deletion is called a *t-criss-cross deletion-correcting code*. We abbreviate this code as *t-criss-cross code*. Throughout this paper we assume that t is a constant with respect to n. We write $f(n) \approx g(n)$, $f(n) \lesssim g(n)$, and $f(n) \gtrsim g(n)$ if the equality or inequality holds for $n \to \infty$.

III. Upper Bound on the Cardinality

This section presents an asymptotic upper bound on the cardinality of any t-criss-cross code. This bound implies an asymptotic lower bound on the redundancy of any binary t-criss-cross code, denoted by $R_{\text{B}}(n, t)$.

Lemma 1 *Any upper bound on the cardinality of a q-ary t-deletion-correcting code $\mathcal{C}_{q,n,t}$ with $q = 2^n$ is also an upper bound on the cardinality of a binary t-criss-cross code.*

Proof: Note that a 2^n-ary t-deletion-correcting code $\mathcal{C}_{2^n,n,t}$ can be seen also as a binary t column deletion-correcting code by interpreting the symbols as binary columns. Since a t-criss-cross code $\mathcal{C}$ can correct any combination of t_{r} row and t_{c} column deletions such that $t_{\text{r}} + t_{\text{c}} = t$, in particular it can also correct any t column deletions. Therefore, any upper bound on the size of $\mathcal{C}_{2^n,n,t}$ is also a valid upper bound on the size of $\mathcal{C}$. ∎

Corollary 2 *For any binary t-criss-cross code $\mathcal{C}$ it holds that*

$$|\mathcal{C}| \lesssim \frac{t! 2^{n^2}}{(2^n - 1)^t n^t}.$$

Consequently, we have $R_{\text{B}}(n, t) \gtrsim tn + t \log(n) - \log(t!)$.

Proof: From [29], we have for a q-ary t-deletion correcting code that $|\mathcal{C}_{q,n,t}| \lesssim \frac{t! q^n}{(q-1)^t n^t}$. Even though this bound was proved in [29] when q is fixed, one can verify that it holds true also for $q = 2^n$. Therefore, for any binary t-criss-cross code $\mathcal{C}$ it holds by Lemma 1 that

$$|\mathcal{C}| \leqslant |\mathcal{C}_{2^n,n,t}| \lesssim \frac{t! 2^{n^2}}{(2^n - 1)^t n^t}.$$

Therefore, we have

$$R_{\text{B}}(n, t) \gtrsim n^2 - \log(|\mathcal{C}|) \approx tn + t \log(n) - \log(t!).$$

∎

IV. Code Construction

In this section we present an existential construction of t-criss-cross codes. We start with an intuitive road map to our code construction and then formally define each ingredient.

A. Road Map

Our construction uses structured arrays so that the indices of the deleted rows and columns can be exactly recovered. Then, the set of structured arrays is intersected with arrays of a Gabidulin code (that can correct row/column erasures) to recover the arrays of the code. The structure is depicted in Figure 1.

We structure the $n \times n$ codewords $\mathbf{C}$ as follows. We protect the columns with indices between $t \log n + 1$ and $n - (t+1)^2$ using $t \log n$ codes where each one is a binary systematic t-deletion-correcting code. We divide those codes into t blocks each of size $\log n$. We impose what we call a *window constraint* on the columns of the systematic part of every block. This constraint ensures that every $t + 1$ consecutive columns are different. Therefore, the indices of the deleted columns within the systematic part can be located by using all $\log n$ deletion-correcting codes of any block (Claim 4).

In the redundancy part, runs may exist. Thus, the recovery of the index of the deleted columns is only guaranteed within

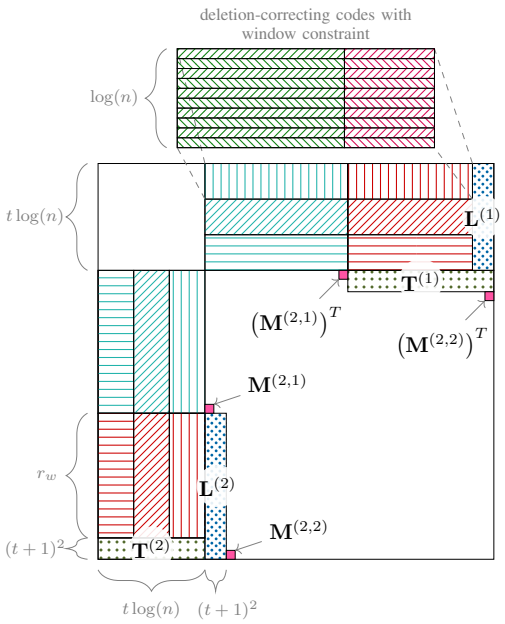

Fig. 1: Illustration of an array contained in the locator set $\mathcal{L}_t(n)$ for $t = 3$. In the first $t \log(n)$ rows there are t blocks each consisting of a systematic part (cyan) and a redundancy part (red). Each row is encoded using a systematic t-deletion-correcting code (zoomed in part). In addition, in the systematic part of each block a window constrained is imposed. Those blocks are used to locate column deletions. This structure is protected with the arrays $\mathbf{L}^{(1)}$ (blue) against row deletions and $\mathbf{T}^{(1)}$ (brown) against column deletions. Lastly, to locate the borders of $\mathbf{T}^{(1)}$ we use the marker arrays $\mathbf{M}^{(2,1)}$ and $\mathbf{M}^{(2,2)}$ (pink). A symmetric structure locates row deletions.

possible runs. To recover the exact location of the deleted columns here, we protect the redundancy part of the codes by appending (from below) what we call a *locator array* that can detect the exact positions of column deletions within this part. We call this array $\mathbf{T}^{(1)}$ (Claim 3).

Note that for the window constraint to work, we need to have all $\log n$ deletion-correcting codes of the considered block. Therefore, we use the subarray $\mathbf{C}_{[1:t\log n],[n-(t+1)^2:n]}$ as a locator array $\mathbf{L}^{(1)}$ that can detect the exact position of a deleted row within the first $t\log n$ rows (Claim 3). As a result, if all t deletions are row deletions within the first $t\log n$ rows, then the locator array is enough to recover all the indices (Lemma 6). Otherwise, we have at least one block of the t blocks that is not affected by a row deletion. This would be the block that will be used to recover any column deletion within the range between $t\log n+1$ and $n-(t+1)^2$ (Lemma 7).

One more step is needed. We must be able to locate the position of the locator arrays within the resulting $(n-t_{\mathrm{r}})\times(n-t_{\mathrm{c}})$ array $\widetilde{\mathbf{C}}$. Therefore, we put four *marker arrays* after the locators that are detectable even after t deletions. We call those arrays $\mathbf{M}^{(1,1)}$ and $\mathbf{M}^{(1,2)}$.

The same structure (transposed) is used to index the rows. In addition, the columns with indices between 1 and $t\log n$ are protected by the locator array used for protecting the deletion-correcting codes indexing the rows. Note the claims and lemmas mentioned before also include the statements to recover the row indices.

In the next subsections, we formally define the five main ingredients of our code: *(i)* the locator arrays; *(ii)* the binary systematic t-deletion-correcting codes with windows constraints; *(iii)* the marker arrays; *(iv)* the *locator set* which is the combination of all the previously mentioned parts; and *(v)* a Gabidulin code [30] that is used to correct row/column erasures.

B. Locator Arrays

For a positive integer a, we denote by $\mathbf{I}_a$ the identity array of dimension $a\times a$ and by $\mathbf{1}_a$ and $\mathbf{0}_a$ the all-one vector and all-zero vector of length a, respectively. We use $\otimes$ to indicate the Kronecker product. We thus have the following definition from [28].

Definition 1 (Locator arrays) *We set $\mathbf{L}'\in\Sigma^{(t+1)\times(t+1)^2}$ as $\mathbf{L}'\triangleq\mathbf{I}_{t+1}\otimes\mathbf{1}_{t+1}$. More precisely, $\mathbf{L}'$ has the following structure*

$$\mathbf{L}'=\begin{pmatrix}\mathbf{1}_{t+1}&\mathbf{0}_{t+1}&\dots&\mathbf{0}_{t+1}\\ \mathbf{0}_{t+1}&\mathbf{1}_{t+1}&\dots&\mathbf{0}_{t+1}\\ \vdots&\vdots&\ddots&\vdots\\ \mathbf{0}_{t+1}&\mathbf{0}_{t+1}&\dots&\mathbf{1}_{t+1}\end{pmatrix}.$$

Let s be a multiple of $(t+1)$ such that $s\geqslant(t+1)^2$. We define the locator array $\mathbf{L}_s\in\Sigma^{s\times(t+1)^2}$ as

$$\mathbf{L}_s\triangleq\mathbf{1}_{\frac{s}{t+1}}^{T}\otimes\mathbf{L}'.$$

Moreover, we define the locator array $\mathbf{T}_s\in\Sigma^{(t+1)^2\times s}$ to be the transpose of $\mathbf{L}_s$, i.e.,

$$\mathbf{T}_s\triangleq\mathbf{L}_s^T=\mathbf{1}_{\frac{s}{t+1}}\otimes\mathbf{L}'^T.$$

Throughout the paper we drop s in the notation $\mathbf{L}_s$ and $\mathbf{T}_s$ when the value of s is clear from the context.

Claim 3 *Given an array $\mathbf{L}_s$ affected by t_{r} row and t_{c} column deletions such that $t_{\mathrm{r}}+t_{\mathrm{c}}=t$. Divide $\mathbf{L}_s$ into $(t+1)$ subarrays each consisting of $(t+1)$ consecutive columns of $\mathbf{L}_s$. By examining $\widetilde{\mathbf{L}}_s$, we can locate the exact positions of the deleted rows. We can also determine the number of column deletions that happened in each subarray of $\mathbf{L}_s$.*

Given an array $\mathbf{T}_s$ affected by t_{r} row and t_{c} column deletions such that $t_{\mathrm{r}}+t_{\mathrm{c}}=t$. The same as described above for $\mathbf{L}_s$ can be done by switching rows for columns in the previous statement.

Proof: We prove the first part of the claim, the second part follows similarly since $\mathbf{T}_s=\mathbf{L}_s^T$.

By construction of $\mathbf{L}_s$, for any $i\in[s-t-1]$ and $j\in[t+1]$ it holds that $\mathbf{L}_{i,[(t+1)^2]}\neq\mathbf{L}_{i+j,[(t+1)^2]}$. This property holds true even in the presence of at most t column deletions in $\mathbf{L}_s$. Thus, due to the fixed structure of $\mathbf{L}_s$ one can uniquely determine the exact indices of the deleted rows.

Moreover, we divide $\mathbf{L}_s$ in subarrays consisting of $(t+1)$ columns. For any $a,b,c\in[t+1]$ we have $\mathbf{L}_{[s],(c-1)a}=\mathbf{L}_{[s],(c-1)a+b}$. In words, we have $(t+1)$ identical columns in a subarray. This property holds true even if there were at most t row deletions in $\mathbf{L}_s$. Therefore, we can determine the deleted columns within any subarray by counting the number of missing columns. ∎

C. Deletion-Correcting Codes with Window Constraints

Deletion-correcting codes: We use the construction of [17] for our binary systematic t-deletion correcting code. We briefly recall the results of [17]. Given a sequence $\boldsymbol{k}\in\Sigma^{\kappa}$, one can compute a redundancy vector $\boldsymbol{r_k}\in\Sigma^{\rho_\kappa}$ with $\rho_\kappa\leqslant 4t\log(\kappa)+o(\log(\kappa))$. The resulting sequence $(\boldsymbol{k}|\boldsymbol{r_k})$ can be uniquely recovered after t deletions. Note that $\boldsymbol{r_k}$ is a function of the information $\boldsymbol{k}$ and ρ_κ is a function of the information length κ and the number of deletions t.

Window constraint: We define the window constraint as the set $\mathcal{W}_t(\ell,w)\subseteq\Sigma^{\ell\times w}$, where for any $\mathbf{W}\in\mathcal{W}_t(\ell,w)$, $i\in[w-t]$ and $j\in[t]$, it holds that $\mathbf{W}_{[\ell],i}\neq\mathbf{W}_{[\ell],i+j}$.

For an array $\mathbf{W}\in\mathcal{W}_t(\ell,w)$, let $\mathbf{R_W}\in\Sigma^{\ell\times r_w}$ be the array formed such that for any $i\in[\ell]$ the i^{th} row of $\mathbf{R_W}$ is the redundancy vector corresponding to the i^{th} row of $\mathbf{W}$; computed using the construction in [17]. We refer to the array $\mathbf{R_W}\in\Sigma^{\ell\times r_w}$ as the redundancy array. Let $m\triangleq w+r_w$, we define $\mathcal{D}_t^{(1)}(\ell,m)$ as the set of all arrays resulting from the concatenation of $\mathbf{W}$ and $\mathbf{R_W}$, i.e.,

$$\mathcal{D}_t^{(1)}(\ell,m)\triangleq\left\{\mathbf{D}\in\Sigma^{\ell\times m}:\begin{array}{c}\mathbf{D}=(\mathbf{W}\mid\mathbf{R_W}),\\ \text{s.t.}\mathbf{W}\in\mathcal{W}_t(\ell,w)\end{array}\right\}.$$

In words, $\mathcal{D}_t^{(1)}(\ell,m)$ is the set of binary systematic t-deletion-correcting codes in which the systematic part satisfies the

imposed window constraint. This set will be used to index the columns of our arrays in the constructed code. We define $\mathcal{D}_t^{(2)}(\ell, m) \triangleq \left\{\mathbf{D}^T : \mathbf{D} \in \mathcal{D}_t^{(1)}(\ell, m)\right\}$. This set is going to be used for indexing the rows.

Claim 4 *Given an array $\mathbf{D} = (\mathbf{W} \mid \mathbf{R_W}) \in \mathcal{D}_t^{(1)}(\ell, m)$ affected by t column deletions and no row deletions, we can locate the exact positions of the deleted columns in the sub-array $\mathbf{W}$.*

The same holds for any array in $\mathcal{D}_t^{(2)}(\ell, m)$ by switching in the argument rows and columns.

Proof: Assume $\widetilde{\mathbf{D}} = (\widetilde{\mathbf{W}} \mid \widetilde{\mathbf{R}}_{\mathbf{W}})$ is the array obtained after the deletions. For each row in $\widetilde{\mathbf{W}}$ we can use the corresponding redundancy in $\widetilde{\mathbf{R}}_{\mathbf{W}}$ to correct the deletions that happened in this row [17]. We start by looking at the position of the first recovered bit in each row. In each row, this position may be unique or may be in an interval of possible positions (run). The exact location of the column is then determined by the unique position in which all runs (of all rows) intersect. The intersection is guaranteed to be unique by the imposed window constraint; since for any $i \in [w-t]$, and $j \in [t]$, it holds that $\mathbf{W}_{[\ell],i} \neq \mathbf{W}_{[\ell],i+j}$. This process is repeated for all recovered bits until all t positions are determined.

A similar argument follows for the second statement of the claim. ∎

D. Marker Arrays

We define the following arrays of dimension $(t+1)\times(t+1)$ which will operate as *markers* to locate the position of the locator arrays in the resulting $\widetilde{\mathbf{C}}$. Recall that we use four locator arrays in our construction, namely $\mathbf{L}^{(1)}$, $\mathbf{L}^{(2)}$, $\mathbf{T}^{(1)}$, and $\mathbf{T}^{(2)}$, cf. Figure 1. We only need marker arrays for $\mathbf{T}^{(1)}$ and $\mathbf{L}^{(2)}$. The position of $\mathbf{L}^{(1)}$ and $\mathbf{T}^{(2)}$ can be then determined. The first marker array $\mathbf{M}^{(2,1)}$, put on top of $\mathbf{L}^{(2)}$, consists of the first $t+1$ columns of $\mathbf{L}'$. The second marker array $\mathbf{M}^{(2,2)}$, put on the right of $\mathbf{L}^{(2)}$, consists of the complement of the last $t+1$ columns of $\mathbf{L}'$. The marker arrays $\mathbf{M}^{(1,1)}$ and $\mathbf{M}^{(1,2)}$ are the transpose of $\mathbf{M}^{(2,1)}$ and $\mathbf{M}^{(2,2)}$, respectively.

E. Locator Set

We formally define the sets of arrays in $\Sigma^{n\times n}$ that form our code. Let $\mathbf{X} \in \Sigma^{n\times n}$, we start with the set of arrays that are used to index the columns. This set is denoted by $\mathcal{H}_t(\ell, n)$. The arrays in this set have the first $t\ell$ columns divided into t blocks. The columns whose indices are between $t\ell+1$ and $n-(t+1)^2$ of each row consist of a systematic t-deletion-correcting code in which the systematic part satisfies the window constraint. We can write

$$\mathcal{H}_t(\ell, n) \triangleq \left\{\mathbf{X}\colon \begin{array}{l}\mathbf{X}_{[(a-1)\ell+1:a\ell],[t\ell+1:n-(t+1)^2]} \\ \quad \in \mathcal{D}_t^{(1)}(\ell, n-t\ell-(t+1)^2)\ \forall\, a \in [t]\end{array}\right\}.$$

The set of arrays $\mathcal{V}_t(\ell, n)$ that are used to index the rows is defined similarly to $\mathcal{H}_t(\ell, n)$ by replacing columns with rows.

$$\mathcal{V}_t(\ell, n) \triangleq \left\{\mathbf{X}\colon \begin{array}{l}\mathbf{X}_{[t\ell+1:n-(t+1)^2],[(b-1)\ell+1:b\ell]} \\ \quad \in \mathcal{D}_t^{(2)}(\ell, n-t\ell-(t+1)^2)\ \forall\, b \in [t]\end{array}\right\}.$$

For a value of r_w that divides[1] $t+1$, the set of arrays $\mathcal{E}_t(\ell, n)$ that contains the locator arrays in the positions shown in Figure 1 is defined as follows.

$$\mathcal{E}_t(\ell, n) \triangleq \left\{\mathbf{X}\colon \begin{array}{l}\mathbf{X}_{[1:t\ell],[n-(t+1)^2+1:n]} = \mathbf{L}_{t\ell}, \\ \mathbf{X}_{[n-r_w-(t+1)^2+1:n],[t\ell+1:t\ell+(t+1)^2]} = \mathbf{T}_{r_w+(t+1)^2}, \\ \mathbf{X}_{[n-(t+1)^2+1:n],[1:t\ell]} = \mathbf{T}_{t\ell}, \\ \mathbf{X}_{[t\ell+1:t\ell+(t+1)^2],[n-r_w-(t+1)^2+1:n]} = \mathbf{L}_{r_w+(t+1)^2},\end{array}\right\}.$$

The set of arrays that contains the marker arrays in the positions shown in Figure 1, is defined as follows.

$$\mathcal{M}_t(\ell, n) \triangleq \left\{\mathbf{X}\colon \begin{array}{l}\mathbf{X}_{[t\ell+1:t\ell+(t+1)],[n-r_w-(t+1)^2-(t+1)+1:n-r_w-(t+1)^2]} \\ \qquad = \mathbf{M}^{(1,1)}, \\ \mathbf{X}_{[t\ell+(t+1)^2+1:t\ell+(t+1)^2+(t+1)],[n-(t+1)+1:n]} \\ \qquad = \mathbf{M}^{(1,2)}, \\ \mathbf{X}_{[n-r_w-(t+1)^2-(t+1)+1:n-r_w-(t+1)^2],[t\ell+1:t\ell+(t+1)]} \\ \qquad = \mathbf{M}^{(2,1)}, \\ \mathbf{X}_{[n-(t+1)+1:n],[t\ell+(t+1)^2+1:t\ell+(t+1)^+(t+1)]} \\ \qquad = \mathbf{M}^{(2,2)}\end{array}\right\}.$$

We can conclude this subsection by defining the *locator set* that is the set of all arrays that have the structure required by our code to recover the indices of the deleted columns and rows. The locator set is the intersection of all the previously defined sets.

Definition 2 (Locator Set) *We define the following set:*

$$\mathcal{L}_t(n) \triangleq \mathcal{H}_t(\ell, n) \cap \mathcal{V}_t(\ell, n) \cap \mathcal{E}_t(\ell, n) \cap \mathcal{M}_t(\ell, n).$$

For an illustration of such arrays we refer to Figure 1. The defining parameters of $\mathcal{L}_t(n)$ are only t and n. By fixing those, all other parameters can be obtained from the imposed constraints. Most noteworthy parameters are w and r_w, which are functions of n and t.

F. Construction

We write $\mathcal{C}_{\mathrm{Gab}}(n, t)$ to refer to a linear Gabidulin code which is able to correct any pattern of t_{r} row and t_{c} column erasures in an $n \times n$ array as long as $t_{\mathrm{r}} + t_{\mathrm{c}} = t$ [20]. Now we are able to present our existential construction.

Construction 1 *The code $\mathcal{C}_{t,n} \subseteq \Sigma^{n\times n}$ is the set of arrays that belong to*

$$\mathcal{L}_t(n) \cap \mathcal{C}_{\mathrm{Gab}}(n, t).$$

Theorem 5 *The code $\mathcal{C}_{t,n}$ described in Construction 1 is a t-criss-cross code.*

A rough concept of our construction is as follows. In our codewords, we first introduce the structure $\mathcal{L}_t(n)$ to locate the

[1] If the value of r_w does not divide $t+1$, then one can simply expand the dimension of the locator arrays in $\mathcal{E}_t(\ell, n)$ to the next multiple of $t+1$ that is greater than $r_w + (t+1)^2$.

indices of the deleted columns and rows. With this knowledge we can introduce erasures into the missing rows and columns and convert the deletion problem into an erasure problem which can be solved by the Gabidulin code $\mathcal{C}_{\mathrm{Gab}}(n,t)$ [20]. We call this type of decoding the *locate-decode strategy*. Theorem 5 will be proven by providing a generic decoding strategy in the next section.

V. Decoder

Assume we have a codeword $\mathbf{C} \in \mathcal{C}_{t,n}$. The decoder receives an array $\widetilde{\mathbf{C}} \in \Sigma^{(n-t_{\mathrm{r}})(n-t_{\mathrm{c}})}$ obtained from $\mathbf{C}$ by t_{r} row and t_{c} column deletions. Let us denote the set of indices of the rows and columns that got deleted by $\mathcal{I}^{(t_{\mathrm{r}})} \subset [n]$ and $\mathcal{I}^{(t_{\mathrm{c}})} \subset [n]$, respectively, with $|\mathcal{I}^{(t_{\mathrm{r}})}| + |\mathcal{I}^{(t_{\mathrm{c}})}| = t_{\mathrm{r}} + t_{\mathrm{c}} = t$. As mentioned before we first focus on locating the indices of the row and column deletions.

Lemma 6 *Given the array $\widetilde{\mathbf{C}}$, any row index $i \in \mathcal{I}^{(t_{\mathrm{r}})}$ such that $1 \leqslant i \leqslant t\ell$ or $n - r_w - (t+1)^2 < i \leqslant n$ can be recovered. Similarly, any column index $j \in \mathcal{I}^{(t_{\mathrm{c}})}$ such that $1 \leqslant j \leqslant t\ell$ or $n - r_w - (t+1)^2 < j \leqslant n$ can be recovered.*

Proof: We focus on how to recover the row indices $i \in \mathcal{I}^{(t_{\mathrm{r}})}$ and the column indices $j \in \mathcal{I}^{(t_{\mathrm{c}})}$ that satisfy $1 \leqslant i \leqslant t\ell$ and $n - r_w - (t+1)^2 < j \leqslant n$. Recovering the remaining indices of the statement follows by the symmetry of the construction.

It can be shown (and is omitted for brevity) that the boundaries of $\widetilde{\mathbf{L}}^{(1)}$ and $\widetilde{\mathbf{T}}^{(1)}$ in $\widetilde{\mathbf{C}}$ can be exactly recovered by leveraging the structure of $\mathbf{L}^{(1)}$, $\mathbf{T}^{(1)}$, and the imposed markers $\mathbf{M}^{(1,1)}$ and $\mathbf{M}^{(1,2)}$. Therefore, by Claim 3 we can locate any column deletions with indices $n - r_w - (t+1)^2 < j \leqslant n$ by decoding $\widetilde{\mathbf{T}}^{(1)}$. Consequently, having the boundaries of $\widetilde{\mathbf{L}}^{(1)}$ and using Claim 3, we can recover the indices of the deleted rows that satisfy $1 \leqslant i \leqslant t\ell$. ∎

Lemma 7 *Given the array $\widetilde{\mathbf{C}}$, any row index $i \in \mathcal{I}^{(t_{\mathrm{r}})}$ such that $t\ell < i \leqslant n - r_w - (t+1)^2$ can be recovered. Similarly, any column index $j \in \mathcal{I}^{(t_{\mathrm{c}})}$ such that $t\ell < j \leqslant n - r_w - (t+1)^2$ can be recovered.*

Proof: We start by proving that the column indices can be recovered. We want to leverage the structure imposed by the set $\mathcal{H}_t(\ell, n)$. For an array $\mathbf{C} \in \mathcal{H}_t(\ell, n)$, each row of the subarray $\mathbf{C}_{[1:t\ell],[t\ell+1:n-(t+1)^2]}$ is encoded using a binary systematic t-deletion-correcting code. In addition, the columns $\mathbf{C}_{[1:t\ell],j}$ such that $t < j \leqslant n - r_w - (t+1)^2$ are the systematic part of this code. Recall that the rows are divided into t blocks, each of size ℓ, where in each block the columns $t < j \leqslant n - r_w - (t+1)^2$ satisfy the window constraint. We assume that at least one column in this interval is deleted. Therefore, at most $(t-1)$ rows can be deleted in $\mathbf{C}$. This means, that there exists at least one block of ℓ rows that is not affected by any row deletions. By Lemma 6 we can locate this block. By Claim 4 we can recover the indices of the columns deleted within the range $t < j \leqslant n - r_w - (t+1)^2$. Similarly, we can obtain the indices with $t\ell < i \leqslant n - r_w - (t+1)^2$ by leveraging the structure imposed by $\mathcal{V}_t(\ell, n)$, Lemma 6, and Claim 4. ∎

Now we can present the full proof of our code construction.

Proof of Theorem 5: By applying Lemma 6 and Lemma 7, we can determine the sets of indices $\mathcal{I}^{(t_{\mathrm{r}})}$ and $\mathcal{I}^{(t_{\mathrm{c}})}$. For all $i \in \mathcal{I}^{(t_{\mathrm{r}})}$ and $j \in \mathcal{I}^{(t_{\mathrm{c}})}$ we insert row or column erasure in $\widetilde{\mathbf{C}}$ starting from the smallest index. Now we can apply a Gabidulin criss-cross erasure decoder to determine the values of the erased symbols [20]. ∎

VI. Redundancy

In this section we perform an analysis of the redundancy of our code denoted by $R(n,t)$. We will refer to the redundancy of each individual set $\mathcal{C}_{\mathrm{Gab}}(t,n)$, $\mathcal{L}_t(n)$, $\mathcal{H}_t(\ell,n)$, $\mathcal{V}_t(\ell,n)$, $\mathcal{E}_t(\ell,n)$, $\mathcal{W}_t(\ell,w)$ and $\mathcal{M}_t(\ell,n)$ by $R_*(n,t)$, where $*$ is replaced with the corresponding set letter. For the sake of brevity, we omit the proofs of the provided claims and lemmas. The detailed proofs can be found in [31]. In the following, we give an intuition behind the computations of the redundancy.

Since $\mathcal{C}_{t,n} = \mathcal{L}_t(n) \cap \mathcal{C}_{\mathrm{Gab}}(t,n)$ and due to the fact that the Gabidulin code is a linear code, we can compute the code redundancy as follows.

$$R(n,t) = R_{\mathcal{L}}(n,t) + R_{\mathrm{G}}(n,t)$$

Moreover, since the intersected sets in the locator set $\mathcal{L}_t(n)$ impose constraints on disjoint positions in the $n \times n$ arrays, we can further split the redundancy as follows.

$$R_{\mathcal{L}}(n,t) = R_{\mathcal{H}}(n,t) + R_{\mathcal{V}}(n,t) + R_{\mathcal{E}}(n,t) + R_{\mathcal{M}}(n,t)$$

The sets $\mathcal{H}_t(\ell,n)$ and $\mathcal{V}_t(\ell,n)$ impose similar constraints: t disjoint subarrays constrained with the window constraint where each row is protected by a systematic t-deletion correcting code from [17].

Claim 8 *The redundancy resulting from the constraints imposed by the two sets $\mathcal{H}_t(\ell,n)$ and $\mathcal{V}_t(\ell,n)$ is bounded as*

$$R_{\mathcal{H}}(n,t) + R_{\mathcal{V}}(n,t) \leqslant 2t(R_{\mathcal{W}}(\ell,w) + \log(n) \cdot r_w),$$

where $w = n - t\log(n) - r_w - (t+1)^2$ and $r_w \leqslant 4t\log(n) + o(\log n)$.

Observe that the constraints for the remaining sets fix values for certain subarray boundaries. Therefore, the following can be obtained.

Claim 9 *The redundancy $R_{\mathcal{L}}(n,t)$ resulting from the constraints imposed by the set $\mathcal{L}_t(n)$ is bounded as*

$$R_{\mathcal{L}}(n,t) \leqslant (8t^2 + 2t)\log^2(n) + o(\log^2(n)).$$

We can conclude this section with the statement on the redundancy $R(n,t)$ of the code $\mathcal{C}_{t,n}$ presented in Construction 1. Note that the redundancy added by the Gabidulin code is tn.

Lemma 10 *The redundancy of the code $\mathcal{C}_{t,n}$ is bounded as*

$$R(n,t) \leqslant tn + (8t^2 + 2t)\log^2(n) + o(\log^2(n)).$$

REFERENCES

[1] R. Heckel, G. Mikutis, and R. N. Grass, "A Characterization of the DNA Data Storage Channel," *Scientific Reports*, vol. 9, no. 1, p. 9663, 2019. [Online]. Available: https://doi.org/10.1038/s41598-019-45832-6

[2] F. Sala, C. Schoeny, N. Bitouzé, and L. Dolecek, "Synchronizing files from a large number of insertions and deletions," *IEEE Transactions on Communications*, vol. 64, no. 6, pp. 2258–2273, June 2016.

[3] R. Venkataramanan, H. Zhang, and K. Ramchandran, "Interactive low-complexity codes for synchronization from deletions and insertions," in *48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2010, pp. 1412–1419.

[4] R. Venkataramanan, V. Narasimha Swamy, and K. Ramchandran, "Low-complexity interactive algorithms for synchronization from deletions, insertions, and substitutions," *IEEE Transactions on Information Theory*, vol. 61, no. 10, pp. 5670–5689, 2015.

[5] S. S. T. Yazdi and L. Dolecek, "A deterministic polynomial-time protocol for synchronizing from deletions," *IEEE Transactions on Information Theory*, vol. 60, no. 1, pp. 397–409, 2013.

[6] N. Ma, K. Ramchandran, and D. Tse, "Efficient file synchronization: A distributed source coding approach," in *IEEE International Symposium on Information Theory Proceedings*, 2011, pp. 583–587.

[7] L. Dolecek and V. Anantharam, "Using Reed–Muller RM (1, m) codes over channels with synchronization and substitution errors," *IEEE Transactions on Information Theory*, vol. 53, no. 4, pp. 1430–1443, April 2007.

[8] V.I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals (in Russian)," *Doklady Akademii Nauk SSR*, vol. 163, no. 4, pp. 845–848, 1965.

[9] R. R. Varshamov and G. M. Tenengolts, "Codes which correct single asymmetric errors (in Russian)," *Automatika i Telemkhanika*, vol. 161, no. 3, pp. 288–292, 1965.

[10] V. Guruswami and C. Wang, "Deletion codes in the high-noise and high-rate regimes," *IEEE Transactions on Information Theory*, vol. 63, no. 4, pp. 1961–1970, Apr. 2017.

[11] J. Brakensiek, V. Guruswami, and S. Zbarsky, "Efficient low-redundancy codes for correcting multiple deletions," *IEEE Transactions on Information Theory*, vol. 64, no. 5, pp. 3403–3410, 2017.

[12] S. K. Hanna and S. El Rouayheb, "Guess & check codes for deletions, insertions, and synchronization," *IEEE Transactions on Information Theory*, vol. 65, no. 1, pp. 3–15, 2018.

[13] R. Gabrys and F. Sala, "Codes correcting two deletions," *IEEE Transactions on Information Theory*, vol. 65, no. 2, pp. 965–974, Feb 2019.

[14] J. Sima, N. Raviv, and J. Bruck, "Two deletion correcting codes from indicator vectors," *IEEE Transactions on Information Theory*, pp. 1–1, 2019.

[15] J. Sima and J. Bruck, "On optimal k-deletion correcting codes," *IEEE Transactions on Information Theory*, pp. 1–1, 2020.

[16] V. Guruswami and J. Hρastad, "Explicit two-deletion codes with redundancy matching the existential bound," *arXiv preprint arXiv:2007.10592*, 2020.

[17] J. Sima, R. Gabrys, and J. Bruck, "Optimal systematic t-deletion correcting codes," in *2020 IEEE International Symposium on Information Theory (ISIT)*, 2020, pp. 769–774.

[18] A. Krishnamurthy, A. Mazumdar, A. McGregor, and S. Pal, "Trace reconstruction: Generalized and parameterized," *arXiv preprint arXiv:1904.09618*, 2019.

[19] R. M. Roth, "Maximum-rank array codes and their application to crisscross error correction," *IEEE Transactions on Information Theory*, vol. 37, no. 2, pp. 328–336, 1991.

[20] E. M. Gabidulin and N. I. Pilipchuk, "Error and erasure correcting algorithms for rank codes," *Designs, Codes and Cryptography*, vol. 49, pp. 105–122, 2008.

[21] D. Lund, E. M. Gabidulin, and B. Honary, "A new family of optimal codes correcting term rank errors," in *IEEE International Symposium on Information Theory*, June 2000, p. 115.

[22] V. R. Sidorenko, "Class of correcting codes for errors with a lattice configuration," *Problemy Reredachi Informatsii*, vol. 12, no. 3, pp. 165–171, Mar. 1976.

[23] M. Blaum and J. Bruck, "MDS array codes for correcting a single criss-cross error," *IEEE Transactions on Information Theory*, vol. 46, no. 3, pp. 1068–1077, May 2000.

[24] E. M. Gabidulin, "Optimum codes correcting lattice errors," *Problemy Peredachi Informatsii*, vol. 21, no. 2, pp. 103–108, 1985.

[25] R. M. Roth, "Probabilistic crisscross error correction," *IEEE Transactions on Information Theory*, vol. 43, no. 5, pp. 1425–1438, Sep. 1997.

[26] A. Wachter-Zeh, "List decoding of crisscross errors," *IEEE Transactions on Information Theory*, vol. 63, no. 1, pp. 142–149, 2017.

[27] R. Bitar, I. Smagloy, L. Welter, A. Wachter-Zeh, and E. Yaakobi, "Criss-cross deletion correcting codes," *International Symposium on Information Theory and Its Applications*, 2020.

[28] M. Hagiwara, "Conversion method from erasure codes to multi-deletion error-correcting codes for information in array design," *International Symposium on Information Theory and Its Applications (ISITA)*, 2020.

[29] A. A. Kulkarni and N. Kiyavash, "Nonasymptotic upper bounds for deletion correcting codes," *IEEE Transactions on Information Theory*, vol. 59, no. 8, pp. 5115–5130, Aug 2013.

[30] E. M. Gabidulin, "Theory of codes with maximum rank distance," *Problemy Peredachi Informatsii*, vol. 21, no. 1, pp. 3–16, 1985.

[31] L. Welter, R. Bitar, A. Wachter-Zeh, and E. Yaakobi, "Multiple criss-cross deletion-correcting codes," *Extended version*, https://sites.google.com/site/rawadbitar1/documents/multiplecrisscross.