Criss-Cross Deletion Correcting Codes

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

***Abstract*—This paper studies the problem of constructing codes correcting deletions in arrays. Under this model, it is assumed that an $n \times n$ array can experience deletions of rows and columns. These deletion errors are referred to as $(t_\mathrm{r}, t_\mathrm{c})$-*criss-cross deletions* if t_r rows and t_c columns are deleted, while a code correcting these deletion patterns is called a $(t_\mathrm{r}, t_\mathrm{c})$-*criss-cross deletion correcting code*. The definitions for *criss-cross insertions* are similar.**

Similar to the one-dimensional case, it is first shown that the problems of correcting criss-cross deletions and criss-cross insertions are equivalent. Then, we mostly investigate the case of $(1,1)$-criss-cross deletions. An asymptotic upper bound on the cardinality of $(1,1)$-criss-cross deletion correcting codes is shown which assures that the asymptotic redundancy is at least $2n - 2 + 2\log n$ bits. Finally, a code construction with an explicit decoding algorithm is presented. The redundancy of the construction is away from the lower bound by at most $2\log n + 9 + 2\log e$ bits.

I. Introduction

The problem of coding for the deletion channel dates back to the work of Levenshtein and others [1], [2] in the 1960s. It is well known that the Varshamov-Tenengolts (VT) code [2] can correct a single deletion, and that this code is nearly optimal with respect to the number of redundant bits. Recently, codes correcting insertions/deletions attract a lot of attention due to their relevance in many applications such as DNA-based data storage systems [3] and communication systems [4], [5].

This paper extends the one-dimensional study of deletion and insertion correction to two dimensions. A $(t_\mathrm{r}, t_\mathrm{c})$-*criss-cross deletion* is the event in which an $n \times n$ array experiences a deletion of t_r rows and t_c columns. A code capable of correcting all $(t_\mathrm{r}, t_\mathrm{c})$-criss-cross deletions is referred to as $(t_\mathrm{r}, t_\mathrm{c})$-*criss-cross deletion correcting code* and $(t_\mathrm{r}, t_\mathrm{c})$-*criss-cross insertion correcting codes* are defined similarly.

It is well known that in the one-dimensional case the size of the single-deletion ball equals the number of runs in the word. However, the characterization of the number of arrays that can be received by a $(1,1)$-criss-cross deletion is more complicated. We show that for almost all arrays it holds that *all* their $(1,1)$-criss-cross deletions are different. We then use this property in deriving an asymptotic upper bound on these codes. On the other hand, our construction of such codes heavily depends on the construction of non-binary single-insertion/deletion correcting codes by Tenengolts [6]. This is one of the ingredients of our construction. In the one-dimensional case, successful decoding from deletions in the transmitted word does not necessarily guarantee that the indices of the deleted symbols are known since the deletion of symbols from the same run results in the same output. While this does not impose a constraint in the one-dimensional case, we had to take this constraint into account when using non-binary single-deletion correcting codes as our component codes.

In Section II, we formally define the codes and notations used in the paper. Section III proves that the correction of criss-cross deletions and insertions is equivalent. In Section IV, we give an asymptotic upper bound on the cardinality of $(1,1)$-criss-cross deletion correcting codes. Then, in Section V, we construct $(1,1)$-criss-cross deletion correcting codes. The correctness of the codes is given by an explicit decoding algorithm. Finally, we analyze the code redundancy and show that it is at most $2\log n + 9 + 2\log e$ away from optimality. The missing proofs in the paper appear in its extended version [7].

II. Definitions and Preliminaries

Let $\Sigma \triangleq \{0,1\}$ and $\Sigma_q \triangleq \{0,\dots,q-1\}$ be, respectively, the binary and the q-ary alphabets. We denote by $\Sigma^{n\times n}$ the set of all binary arrays of dimension $n \times n$. For two integers $i, j \in \mathbb{N}$, $i \leqslant j$, the set $\{i,\dots,j\}$ is denoted by $[i,j]$ and the set $\{1,\dots,j\}$ is denoted by $[j]$. For an array $\mathbf{X} \in \Sigma^{n\times n}$, $X_{i,j}$ denotes its entry positioned at the i^{th} row and j^{th} column. We denote the i^{th} row, j^{th} column of $\mathbf{X}$ by $\mathbf{X}_{i,[n]}, \mathbf{X}_{[n],j}$, 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.

For two positive integers $t_\mathrm{r}, t_\mathrm{c} \leqslant n$, we define a $(t_\mathrm{r}, t_\mathrm{c})$-*criss-cross deletion* in an array $\mathbf{X} \in \Sigma^{n\times n}$ to be the deletion of any t_r rows and t_c columns of $\mathbf{X}$. We denote by $\mathbb{D}_{t_\mathrm{r},t_\mathrm{c}}(\mathbf{X})$ the set of all arrays that result from $\mathbf{X}$ after a $(t_\mathrm{r}, t_\mathrm{c})$-criss-cross deletion. In a similar way we define $(t_\mathrm{r}, t_\mathrm{c})$-*criss-cross insertion* and the set $\mathbb{I}_{t_\mathrm{r},t_\mathrm{c}}(\mathbf{X})$ for the insertion case. If $t_\mathrm{r} = t_\mathrm{c} = t$, we will use the notation of $\mathbb{D}_t(\mathbf{X})$, (t)-criss-cross deletion, (t)-criss-cross insertion, and $\mathbb{I}_t(\mathbf{X})$. Note that the order between the row and column deletions/insertions does not matter.

Definition 1 (**$(t_\mathrm{r}, t_\mathrm{c})$-criss-cross deletion correcting code**) *A $(t_\mathrm{r}, t_\mathrm{c})$-criss-cross deletion correcting code $\mathcal{C}$ is a code that can correct any $(t_\mathrm{r}, t_\mathrm{c})$-criss-cross deletion. A $(t_\mathrm{r}, t_\mathrm{c})$-criss-cross insertion correcting code is defined similarly.*

For clarity of presentation, we refer to a $(1,1)$-criss-cross deletion as a *criss-cross deletion* and $(1,1)$-criss-cross deletion correcting code as a *criss-cross deletion correcting code*.

In our construction we use Tenengolts' single-deletion correcting codes [6], which is briefly explained. For a q-ary vector $\mathbf{x} = (x_1,\dots,x_n)$ we associate its *binary signature* $\boldsymbol{s} = (s_1,\dots,s_{n-1})$, where $s_1 = 1$ and $s_i = 1$ if and only if $x_i \geqslant x_{i-1}$. Thus, all q-ary vectors of length n can be split into disjoint cosets $\mathcal{VT}_{n,q}(a,b)$ defined as all vectors $\mathbf{x}$ with signature $\mathbf{s}$ satisfying

$$\sum_{i=1}^{n}(i-1)s_i \equiv a \mod n, \quad \sum_{i=1}^{n} x_i \equiv b \mod q,$$

RB, LW and AW-Z are with the ECE department at the Technical University of Munich. IS and EY are with the CS department of Technion — Israel Institute of Technology. Emails: {rawad.bitar, lorenz.welter,antonia.wachter-zeh}@tum.de, {ilia.smagloy, 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.

where $0 \leqslant a \leqslant n-1, 0 \leqslant b \leqslant q-1$. Each coset is a single deletion correcting code and there exist $a^\star, b^\star$ such that $|\mathcal{VT}_{n,q}(a^\star, b^\star)| \geqslant \frac{q^n}{q \cdot n}$.

III. Equivalence between Insertion and Deletion Correction

In this section, the following equivalence is shown.

Theorem 1 *A code $\mathcal{C}$ is a criss-cross deletion correcting code if and only if it is a criss-cross insertion correcting code.*

Note that in the one-dimensional case this property holds over any alphabet. Thus, the following lemma can be derived by considering the arrays as one dimensional vectors where each element is a row/column.

Lemma 2 *For an integer $m \geqslant 0$ and two words $\mathbf{X} \in \Sigma^{m\times m}$, $\mathbf{Y} \in \Sigma^{m\times m}$,*

$$\mathbb{D}_{1,0}(\mathbf{X}) \cap \mathbb{D}_{1,0}(\mathbf{Y}) \neq \emptyset \text{ if and only if } \mathbb{I}_{1,0}(\mathbf{X}) \cap \mathbb{I}_{1,0}(\mathbf{Y}) \neq \emptyset$$
$$\mathbb{D}_{0,1}(\mathbf{X}) \cap \mathbb{D}_{0,1}(\mathbf{Y}) \neq \emptyset \text{ if and only if } \mathbb{I}_{0,1}(\mathbf{X}) \cap \mathbb{I}_{0,1}(\mathbf{Y}) \neq \emptyset.$$

Lemma 3 on the other hand, requires a complete proof.

Lemma 3 *For an integer $m \geqslant 0$ and arrays $\mathbf{X} \in \Sigma^{(m+1)\times m}$, $\mathbf{Y} \in \Sigma^{m\times(m+1)}$ it holds that*

$$\mathbb{D}_{1,0}(\mathbf{X}) \cap \mathbb{D}_{0,1}(\mathbf{Y}) \neq \emptyset \text{ if and only if } \mathbb{I}_{0,1}(\mathbf{X}) \cap \mathbb{I}_{1,0}(\mathbf{Y}) \neq \emptyset.$$

Proof: We show the "if" direction while the "only if" part is proved similarly. That is, we prove that if $\mathbb{D}_{1,0}(\mathbf{X}) \cap \mathbb{D}_{0,1}(\mathbf{Y}) \neq \emptyset$ then $\mathbb{I}_{0,1}(\mathbf{X}) \cap \mathbb{I}_{1,0}(\mathbf{Y}) \neq \emptyset$. Assume that there exists $\mathbf{D} \in \Sigma^{m\times m}$ such that $\mathbf{D} \in \mathbb{D}_{1,0}(\mathbf{X}) \cap \mathbb{D}_{0,1}(\mathbf{Y})$ and by contradiction assume that $\mathbb{I}_{0,1}(\mathbf{X}) \cap \mathbb{I}_{1,0}(\mathbf{Y}) = \emptyset$. Let i_R, i_C be the index of the row, column deleted in $\mathbf{X}, \mathbf{Y}$ in order to obtain $\mathbf{D}$, respectively. Finally, $\boldsymbol{r}$ denotes the $i_R{}^{\text{th}}$ row of $\mathbf{X}$, i.e., $\mathbf{X}_{i_R,[m]}$, after an insertion of 0 in position i_C. Similarly, let $\boldsymbol{c}$ be the column $\mathbf{Y}_{[m],i_C}$ after an insertion of 0 in position i_R. Notice that it is also possible to insert 1 in both of the words, as long as the symbol inserted is the same one.

Notice that from the definition of $\mathbf{D}$, it holds that

$$\begin{aligned} &X_{i,j} = D_{i,j} = Y_{i,j} \text{ for } 1 \leqslant i < i_R, 1 \leqslant j < i_C, \\ &X_{i+1,j} = D_{i,j} = Y_{i,j} \text{ for } i_R \leqslant i \leqslant m, 1 \leqslant j < i_C, \\ &X_{i,j} = D_{i,j} = Y_{i,j+1} \text{ for } 1 \leqslant i < i_R, i_C \leqslant j \leqslant m, \\ &X_{i+1,j} = D_{i,j} = Y_{i,j+1} \text{ for } i_R \leqslant i \leqslant m, i_C \leqslant j \leqslant m. \end{aligned} \tag{1}$$

Let $\mathbf{I}^x$ be the result of inserting column $\boldsymbol{c}$ at index i_C into $\mathbf{X}$. The array $\mathbf{I}^y$ is defined similarly by inserting row $\boldsymbol{r}$ at index i_R in $\mathbf{Y}$. Notice that $\mathbf{I}^x$ is a result of inserting a column to $\mathbf{X}$ and thus $\mathbf{I}^x \in \mathbb{I}_{0,1}(\mathbf{X})$. For the same reasons it holds that $\mathbf{I}^y \in \mathbb{I}_{1,0}(\mathbf{Y})$. We conclude the proof by showing that $\mathbf{I}^x = \mathbf{I}^y$. This will be done by considering the following cases.

- For $1 \leqslant i \leqslant i_R, 1 \leqslant j \leqslant i_C$ at most one of $I^x_{i,j}, I^y_{i,j}$ is affected by the insertions/deletions. Thus, the proof for the case $i = i_R, j < i_C$ is presented and the rest follows similarly. By definition it holds that $I^x_{i,j} = \boldsymbol{r}_j$. On the other hand, $I^y_{i,j}$ is an inserted symbol into $\mathbf{Y}$ defined to be $\boldsymbol{r}_j$. From these facts it follows that $I^y_{i,j} = \boldsymbol{r}_j = I^x_{i,j}$.
- For $i = i_R, j = i_C$, it holds that $I^x_{i,j} = \boldsymbol{c}_i = 0 = \boldsymbol{r}_j = I^y_{i,j}$.
- For $i > i_R$ and for $j > i_C$, from the definition of column and row insertions it follows that $I^x_{i,j} = X_{i,j-1}, I^y_{i,j} = Y_{i-1,j}$.

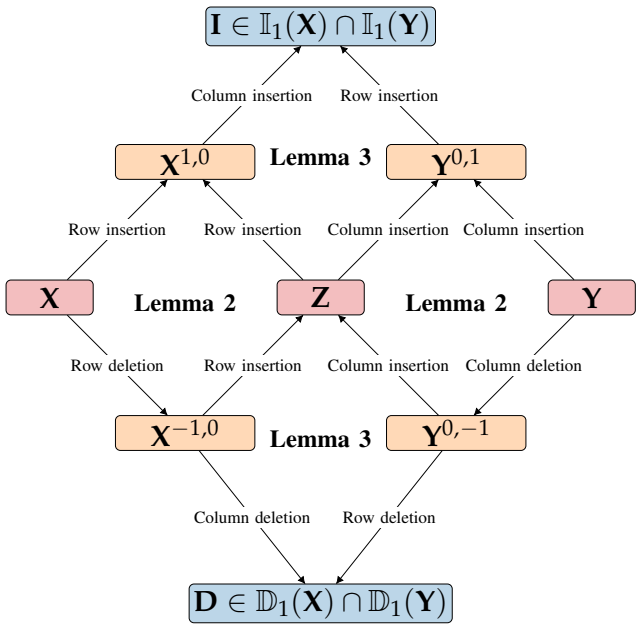

Fig. 1: A flowchart of the proof of Theorem 1.

From (1) we get $X_{i,j-1} = D_{i-1,j-1}, Y_{i-1,j} = D_{i-1,j-1}$. This results in $I^x_{i,j} = X_{i,j-1} = D_{i-1,j-1} = Y_{i-1,j} = I^y_{i,j}$.

- Cases $\{i > i_R, j \leqslant i_C\}, \{i \leqslant i_R, j > i_C\}$ are proven similarly.

This concludes that for all $i, j \in [m+1]$, $I^x_{i,j} = I^y_{i,j}$, which assures that $\mathbf{I}^x = \mathbf{I}^y$, and hence $\mathbb{I}_{0,1}(\mathbf{X}) \cap \mathbb{I}_{1,0}(\mathbf{Y}) \neq \emptyset$, that contradicts our assumption. ■

Now the proof for Theorem 1 can be presented.

Proof of Theorem 1: The proof follows by showing that for any $\mathbf{X}, \mathbf{Y} \in \Sigma^{n\times n}$, $\mathbb{D}_1(\mathbf{X}) \cap \mathbb{D}_1(\mathbf{Y}) = \emptyset$ if and only if $\mathbb{I}_1(\mathbf{X}) \cap \mathbb{I}_1(\mathbf{Y}) = \emptyset$. For the reader's convenience, a flowchart of the proof is presented in Fig. 1. We show only the "only if" part since the "if" part is proved similarly.

Assume that there exists an array $\mathbf{D} \in \Sigma_q^{(n-1)\times(n-1)}$ such that $\mathbf{D} \in \mathbb{D}_{1,1}(\mathbf{X}) \cap \mathbb{D}_{1,1}(\mathbf{Y})$. Hence, $\mathbf{D}$ can be obtained by deleting a row and then a column from $\mathbf{X}$ and by deleting a column and then a row from $\mathbf{Y}$ (note that the order of the row and column deletions does not matter and can be chosen arbitrarily). Denote the arrays created in the process by $\mathbf{X}^{-1,0}, \mathbf{Y}^{0,-1}$, so the following relation holds.

$$\mathbf{X} \xrightarrow{\text{Row Del}} \mathbf{X}^{-1,0} \xrightarrow{\text{Col Del}} \mathbf{D},$$
$$\mathbf{Y} \xrightarrow{\text{Col Del}} \mathbf{Y}^{0,-1} \xrightarrow{\text{Row Del}} \mathbf{D}.$$

Hence, it holds that $\mathbf{D} \in \mathbb{D}_{1,0}(\mathbf{Y}^{0,-1}) \cap \mathbb{D}_{0,1}(\mathbf{X}^{-1,0})$, and thus, from Lemma 3 there exists an array $\mathbf{Z} \in \Sigma^{n\times n}$, such that $\mathbf{Z} \in \mathbb{I}_{0,1}(\mathbf{Y}^{0,-1}) \cap \mathbb{I}_{1,0}(\mathbf{X}^{-1,0})$. By definition, $\mathbf{Z} \in \mathbb{I}_{1,0}(\mathbf{X}^{-1,0})$ is equivalent to $\mathbf{X}^{-1,0} \in \mathbb{D}_{1,0}(\mathbf{Z})$. But, it is also known that $\mathbf{X}^{-1,0} \in \mathbb{D}_{1,0}(\mathbf{X})$, which means that $\mathbf{X}^{-1,0} \in \mathbb{D}_{1,0}(\mathbf{Z}) \cap \mathbb{D}_{1,0}(\mathbf{X})$.

From Lemma 2 it follows that there exists some $\mathbf{X}^{1,0} \in \mathbb{I}_{1,0}(\mathbf{Z}) \cap \mathbb{I}_{1,0}(\mathbf{X})$. The same argument holds for $\mathbf{Y}$ and define its result by $\mathbf{Y}^{0,1}$. Next, notice that we can also conclude that $\mathbf{Z} \in \mathbb{D}_{1,0}(\mathbf{X}^{1,0}) \cap \mathbb{D}_{0,1}(\mathbf{Y}^{0,1})$, so from Lemma 3 it is deduced that there exists an array $\mathbf{I} \in \mathbb{I}_{0,1}(\mathbf{X}^{1,0}) \cap \mathbb{I}_{1,0}(\mathbf{Y}^{0,1})$.

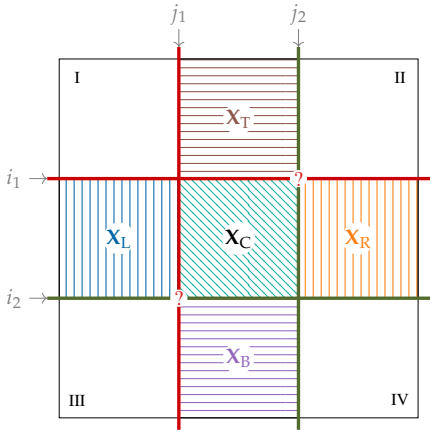

Fig. 2: Required pattern for $\mathbf{X}^{i_1,j_1} = \mathbf{X}^{i_2,j_2}$. Let $(i_1, j_1) \neq (i_2, j_2)$ be the indices of the deleted row and column in two different criss-cross deletions on the array $\mathbf{X}$. W.l.o.g $j_1 < j_2$ and assume $i_1 < i_2$ the constraints are: *1)* Each row of the subarrays $\mathbf{X}_{\mathrm{T}}$ and $\mathbf{X}_{\mathrm{B}}$ must be a row run of length $j_2 - j_1 + 1$ and each column of $\mathbf{X}_{\mathrm{L}}$ and $\mathbf{X}_{\mathrm{R}}$ must be a column run of length $i_2 - i_1 + 1$. *2)* Each diagonal of the subarray $\mathbf{X}_{\mathrm{C}}$ must be a diagonal run. *3)* The corner subarrays I, II, III and IV are outside of the region affected by the criss-cross deletions. Therefore, no constraints are imposed on those subarrays. The same holds for X_{i_1,j_2} and X_{i_2,j_1} since they are both deleted by criss-cross deletions. Note that for $i_1 > i_2$, all the requirements remain the same except for $\mathbf{X}_{\mathrm{C}}$. In this case, the bottom-left to top-right diagonals are diagonal runs.

Note that $\mathbf{I} \in \mathbb{I}_{0,1}(\mathbf{X}^{1,0})$ and $\mathbf{X}^{1,0} \in \mathbb{I}_{1,0}(\mathbf{X})$, which means that $\mathbf{I}$ is obtained by inserting a row and a column to $\mathbf{X}$, i.e., $\mathbf{I} \in \mathbb{I}_1(\mathbf{X})$. A symmetrical argument holds for $\mathbf{Y}$, which assures that $\mathbf{I} \in \mathbb{I}_1(\mathbf{X}) \cap \mathbb{I}_1(\mathbf{Y})$. ■

By induction over Theorem 1, Corollary 4 can be proven.

Corollary 4 *For all $t \in [n]$, a code $\mathcal{C}$ is a (t)-criss-cross deletion correcting code iff it is a (t)-criss-cross insertion correcting code.*

IV. Upper Bound on the Cardinality

In this section we prove an asymptotic upper bound on the cardinality of a criss-cross deletion correcting code. Let $\mathbf{X}^{i,j}$ be the array obtained from $\mathbf{X}$ after a deletion of the i^{th} row and the j^{th} column. Let $\mathbf{X} \in \Sigma^{n\times n}$ and let $i_1, i_2, j_1 \in [n]$ be such that $i_1 \leqslant i_2$. We define a *column run* of length $i_2 - i_1 + 1$ as a sequence of identical consecutive bits in a column j_1, i.e., $X_{i_1,j_1} = X_{i_1+1,j_1} = \cdots = X_{i_2,j_1}$. We define a *row run* similarly. A *diagonal run* of length δ is a sequence of identical bits situated on a diagonal of $\mathbf{X}$, i.e., $X_{i_1,j_1} = X_{i_1+1,j_1+1} = \cdots = X_{i_1+\delta-1,j_1+\delta-1}$.

Lemma 5 *For $i_1, i_2, j_1, j_2 \in [n]$ such that $(i_1, j_1) \neq (i_2, j_2)$ and $n \geqslant 3$, the equality $\mathbf{X}^{i_1,j_1} = \mathbf{X}^{i_2,j_2}$ holds if and only if $\mathbf{X} \in \Sigma^{n\times n}$ satisfies the structure depicted in Fig. 2.*

Sketch of proof: Notice that for $\mathbf{X}^{i_1,j_1}$ the rows are shifted up for $i > i_1$ and the columns are shifted to the left for $j > j_1$. Similarly, this holds for $\mathbf{X}^{i_2,j_2}$. Therefore, one can check that $\mathbf{X}^{i_1,j_1} = \mathbf{X}^{i_2,j_2}$ if and only if $\mathbf{X}$ satisfies the structure described in Fig. 2. ■

An array $\mathbf{X} \in \Sigma^{n\times n}$ is called *good* if $|\mathbb{D}_1(\mathbf{X})| \geqslant \frac{n^2}{2}$ and $\mathbf{X}$ is called *bad* otherwise. Denote by $\mathcal{G}_n, \mathcal{B}_n$ the set of all good and bad arrays in $\Sigma^{n\times n}$, respectively.

Claim 6 *An array $\mathbf{X} \in \Sigma^{n\times n}$ is good if there exists at least $\frac{n}{2}$ consecutive columns that satisfy: 1) any two adjacent columns are different; and 2) when restricted to an interval of consecutive rows $[i_1, i_2]$, any two adjacent columns $\mathbf{X}_{[i_1:i_2],j}$ and $\mathbf{X}_{[i_1:i_2],j\pm1}$ are neither identical, nor identical up to a single up or down shift. Any column of those has $b_n \triangleq 1 + 3\binom{n}{2}$ forbidden choices that violate the imposed conditions.*

Proof: According to Lemma 5 (c.f. Fig. 2), if an array $\mathbf{X} \in \Sigma^{n\times n}$ is bad then there exist $\frac{n^2}{2}$ pairs of $i_1, i_2, j_1, j_2 \in [n]$ such that $(i_1, j_1) \neq (i_2, j_2)$ and $\mathbf{X}^{i_1,j_1} = \mathbf{X}^{i_2,j_2}$. Hence, every two adjacent columns between the j_1^{th} and the j_2^{th} columns satisfy: 1) they are identical in the areas $\mathbf{X}_{\mathrm{T}}$ and $\mathbf{X}_{\mathrm{B}}$; and 2) they are identical up to a single up or down shift within $\mathbf{X}_{\mathrm{C}}$. In addition, every column $j \notin [j_1, j_2]$ has a column run from i_1 to i_2 (c.f. areas $\mathbf{X}_{\mathrm{L}}$ and $\mathbf{X}_{\mathrm{R}}$). Hence, if an array $\mathbf{X}$ has at least $\frac{n}{2}$ consecutive columns that do not satisfy the aforementioned constraints, then it is necessarily a good array. For a given column $j \in [j_1, j_2]$, the number of choices of an adjacent forbidden column is at most $b_n = 1 + 3\binom{n}{2}$. The forbidden choices are the column itself; and any other column which is either identical to j, or identical to j up to a single up or down shift, between any two indices i_1 and i_2. There are $\binom{n}{2}$ options to choose the i_1, i_2 indices. ■

Lemma 7 *For $n \geqslant 28$ it holds that*

$$|\mathcal{B}_n| \leqslant \frac{n}{2} \cdot 2^{n^2-3n}.$$

Proof: If an array $\mathbf{X} \in \Sigma^{n\times n}$ satisfies the condition from Claim 6, then it is necessarily a good array. Otherwise, $\mathbf{X}$ can be either a good array or a bad array. Therefore, we can compute an upper bound on the cardinality of bad arrays as

$$|\mathcal{B}_n| \leqslant \sum_{j=\frac{n}{2}+1}^{n} \binom{n}{j} (b_n)^j 2^{n\cdot(n-j)} \leqslant \frac{n}{2} 2^n (2n^2)^n 2^{\frac{n^2}{2}-n}$$
$$= \frac{n}{2} 2^{\frac{n^2}{2}+n+2n\log(n)} \leqslant \frac{n}{2} \cdot 2^{n^2-3n},$$

where the last inequality holds for $n \geqslant 28$. The upper bound can be interpreted as summing over all arrays with at least $\frac{n}{2}+1$ forbidden columns which can be located at $\binom{n}{j}$ different positions. The other columns can be chosen arbitrarily. ■

In the following we write $f(n) \approx g(n)$ or $f(n) \lesssim g(n)$ if the equality or inequality holds for $n \to \infty$.

Theorem 8 *For any criss-cross deletion correcting code $\mathcal{C}$, it holds that*

$$|\mathcal{C}| \lesssim \frac{2^{n^2}}{2^{2n-1} \cdot \frac{n^2}{2}},$$

thus its asymptotic redundancy is at least $2n - 2 + 2\log n$.

Proof: Let $\mathcal{C}_{\mathcal{B}} \triangleq \mathcal{C} \cap \mathcal{B}_n$ and $\mathcal{C}_{\mathcal{G}} \triangleq \mathcal{C} \cap \mathcal{G}_n$. By the sphere packing bound we have

$$\frac{n^2}{2}|\mathcal{C}_{\mathcal{G}}| = \sum_{\mathbf{X}\in\mathcal{C}_{\mathcal{G}}} |\mathbb{D}_1(\mathbf{X})| \leqslant \sum_{\mathbf{X}\in\mathcal{C}} |\mathbb{D}_1(\mathbf{X})| \leqslant 2^{(n-1)^2}.$$

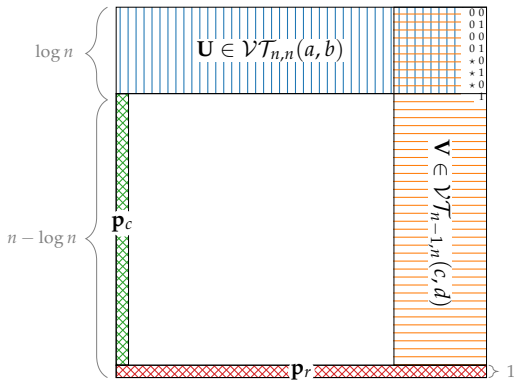

Fig. 3: The structure of the codewords of our CrissCross code. $\mathbf{U}$ is the binary representation of a q-ary vector $\mathbf{u} \in \mathcal{VT}_{n,q}(a,b)$ with $q = n$. Each column is viewed as a symbol of the VT code. The last column of $\mathbf{U}$ is an alternating sequence and the second to last column must start with 4 consecutive 0's. $\mathbf{V}$ is defined similarly to $\mathbf{U}$ where each row is a symbol of the VT code $\mathcal{VT}_{n-1,n}(c,d)$. The alternating sequence of $\mathbf{U}$ is extended by 1 bit in $\mathbf{V}$. $\mathbf{p}_c$ is a parity column consisting of the sum of all columns of its size (and position). $\mathbf{p}_r$ is a parity row consisting of the sum of all rows.

Hence, $|\mathcal{C}_{\mathcal{G}}| \leqslant \frac{2^{(n-1)^2}}{\frac{n^2}{2}}$. Since $|\mathcal{B}_n| \leqslant \frac{n}{2}2^{n^2-3n}$, we can write

$$|\mathcal{C}| = |\mathcal{C}_{\mathcal{G}}| + |\mathcal{C}_{\mathcal{B}}| \leqslant |\mathcal{C}_{\mathcal{G}}| + |\mathcal{B}_n| \leqslant \frac{2^{(n-1)^2}}{\frac{n^2}{2}} + \frac{n}{2}2^{n^2-3n}$$
$$\leqslant \frac{2^{n^2}}{2^{2n-1}\cdot\frac{n^2}{2}}\left(1 + \frac{n^3}{2^{n+3}}\right) \approx \frac{2^{n^2}}{2^{2n-1}\cdot\frac{n^2}{2}}.$$

■

V. Construction

In this section we present our *CrissCross codes* that can correct a criss-cross deletion and state their main properties. For the rest of this section we assume that a, b, c, d are nonegative integers such that $0 \leqslant a, b, d \leqslant n-1$ and $0 \leqslant c \leqslant n-2$. We also assume that n is a power of 2 so that $\log n$ is an integer, while the extension for other values of n will be clear from the context. The main results of this section are summarized in the following theorem and corollary.

Theorem 9 *The CrissCross code $\mathcal{C}_n(a,b,c,d)$ (defined in Construction 1) is a criss-cross deletion correcting code that has an explicit decoder.*

Corollary 10 *There exist integers a, b, c, d for which the redundancy of the CrissCross code $\mathcal{C}_n(a,b,c,d)$ is at most $2n + 4\log n + 7 + 2\log e$ bits and is therefore at most $2\log n + 9 + 2\log e$ bits away from optimality.*

We prove Theorem 9 through a detailed explanation of the code construction in Section V-A. In Section V-B, we show how the decoding works. We compute an upper bound on the redundancy in Section V-C, and thus prove Corollary 10. An intuitive explanation of the proofs is provided in the extended version of this article [7].

A. The Construction

The CrissCross code $\mathcal{C}$ (in Figure 3) can be seen as an intersection of four codes over $\Sigma^{n\times n}$ that define the constraints imposed on the codewords of $\mathcal{C}$. Let $\ell \triangleq \log n$ and we define $\mathbf{W}$ to be the all zero array except for the first $\ell+1$ bits of the last column to be the alternating sequence, i.e., $\mathbf{W}_{[\ell+1],n} = [01010101\cdots]^T$ and $W_{i,j} = 0$ otherwise. We denote by $\mathbf{X} \in \mathcal{VT}_{n,q}(a,b)$ the binary representation of a q-ary vector $\mathbf{x} \in \Sigma_q^n$, such that $\mathbf{x} \in \mathcal{VT}_{n,q}(a,b)$.

$$\mathcal{U}(a,b) \triangleq \left\{ \mathbf{X}; \begin{array}{l} \mathbf{X}_{[\ell],j} \neq \mathbf{X}_{[\ell],j+1}, \quad j \in [n-1] \\ \mathbf{X}_{[4],n-1} = [0000]^T, \\ \mathbf{X}_{[\ell+1],n} = [010101\cdots]^T, \\ \mathbf{X}_{[\ell],[n]} \in \mathcal{VT}_{n,2^\ell}(a,b) \end{array} \right\},$$

$$\mathcal{V}(c,d) \triangleq \left\{ \mathbf{X}; \begin{array}{l} \mathbf{X}_{i,[n-\ell+1:n]} \neq \mathbf{X}_{i+1,[n-\ell+1:n]}, \quad i \in [n-1] \\ \mathbf{X}_{[\ell+1],n} = [0000\cdots]^T, \\ \mathbf{X}_{[n-\ell+1:n],[n-1]} \in \mathcal{VT}_{n-1,2^\ell}(c,d) \end{array} \right\},$$

$$\mathcal{V}'(c,d) \triangleq \left\{ \mathbf{Y}; \begin{array}{l} \mathbf{Y}_{[\ell+1],n} = [010101\cdots]^T, \\ \mathbf{Y} \oplus \mathbf{W} \in \mathcal{V}(c,d) \end{array} \right\},$$

$$\mathcal{P}_c \triangleq \left\{ \mathbf{X}; x_{i,1} = \sum_{j=2}^{n} x_{i,j}, \quad i = \ell+1, \ldots, n-1 \right\},$$

$$\mathcal{P}_r \triangleq \left\{ \mathbf{X}; x_{n,j} = \sum_{i=1}^{n-1} x_{i,j}, \quad j \in [n] \right\}.$$

Construction 1 *The CrissCross code $\mathcal{C}_n(a,b,c,d)$ is the set of arrays $\mathbf{C} \in \Sigma^{n\times n}$ that belong to $\mathcal{U}(a,b) \cap \mathcal{V}'(c,d) \cap \mathcal{P}_c \cap \mathcal{P}_r$.*

B. Decoder

In this section, we show how the CrissCross code construction leverages the structure of a codeword $\mathbf{C}$ in $\mathcal{C}_n(a,b,c,d)$ to detect and correct a criss-cross deletion. Formally, we prove the following lemma.

Lemma 11 *The code $\mathcal{C}_n(a,b,c,d)$ is a criss-cross deletion correcting code with an explicit decoder.*

Proof: The decoder for $\mathcal{C}_n(a,b,c,d)$ receives as input an $(n-1)\times(n-1)$ array $\widetilde{\mathbf{C}}$ resulting from a criss-cross deletion of an array $\mathbf{C}$ of $\mathcal{C}_n(a,b,c,d)$ and works as follows. The decoder starts by looking at the first $\log n \times (n-1)$ subarray of $\widetilde{\mathbf{C}}$ and examining the last column.

Case 1: Assume the last column of $\mathbf{C}$ is not deleted. Using the alternating sequence, the decoder can detect whether or not there was a row deletion in $\mathbf{U}$ and locate its index. This is done by locating a run of length 2 in the alternating sequence. The last bit of the alternating sequence falling in $\mathbf{V}$ and not in $\mathbf{U}$ ensures that the decoder can detect whether the last row of $\mathbf{U}$ is deleted or not.

Case 1 (a): If there was a row deletion in $\mathbf{U}$, the decoder uses the non deleted part of $\mathbf{p}_r$ to recover the deleted row. The decoder can now use the properties of $\mathcal{VT}_{n,n}(a,b)$ to decode the column deletion in $\mathbf{U}$. Since any two consecutive columns

in $\mathbf{U}$ are different, the decoder can locate the exact position of the deleted column and recover its value. The position of the deleted column in $\mathbf{U}$ is the same as the deleted column in the whole array. Using $\mathbf{p}_c$, the decoder can now recover the remaining part of the deleted column.

Case 1 (b): If the deleted row was not in $\mathbf{U}$, the decoder uses $\mathcal{VT}_{n,n}(a,b)$ to recover the index of the deleted column and its value within $\mathbf{U}$ and uses $\mathbf{p}_c$ to recover the value of the deleted column outside of $\mathbf{U}$. Then, the decoder uses $\mathcal{VT}_{n-1,n}(c,d)$ to recover the index of the deleted row. Again, since any two consecutive rows in $\mathbf{V}$ are different the decoder can recover the exact position of the deleted row. Using $\mathbf{p}_r$, the decoder recovers the value of the deleted row.

Case 2: Now assume that the last column of $\mathbf{C}$ is deleted. By looking at the last column of $\mathbf{C}'$, the decoder knows that the alternating sequence is missing thanks to the run of 0's inserted in the beginning of the second to last column of $\mathbf{C}$. Note that irrespective of the location of the row deletion, the last column will have a run of at least three 0's which cannot happen in the alternating sequence. Therefore, the decoder knows that the last column is deleted and starts by looking at $\mathbf{V}$. Using the parity $\mathbf{p}_c$, the decoder recovers the missing part of the deleted column that is in $\mathbf{V}$ but not in $\mathbf{U}$. By construction, the first $\log n$ bits of the last column of $\mathbf{V}$ are set to 0. Thus, the decoder recovers the whole missing column. By using the property of $\mathcal{VT}_{n-1,n}(c,d)$, the decoder recovers the index of the missing row and uses $\mathbf{p}_r$ to recover its value. After recovering the deleted row the decoder adds the alternating sequence to $\mathbf{U}$ and recovers the whole array $\mathbf{C}$. ■

C. Redundancy of the Code

In this section we show that there exist a,b,c,d for which $R_{\mathcal{C}_n(a,b,c,d)} \leqslant 2n + 4\log n + 7 + 2\log e$. We do so by computing a lower bound on $\log|\mathcal{C}_n(a,b,c,d)|$. To that end we count the number of $n \times n$ binary arrays that satisfy all the requirements imposed on the codewords $\mathbf{C}$ in $\mathcal{C}_n(a,b,c,d)$.

Since the constraints imposed on the codes $\mathcal{U}(a,b) \cap \mathcal{V}'(c,d)$, $\mathcal{P}_c$, and $\mathcal{P}_r$ are disjoint, we have that

$$\begin{aligned} R_{\mathcal{C}_n(a,b,c,d)} &= R_{\mathcal{U}(a,b)\cap\mathcal{V}'(c,d)} + R_{\mathcal{P}_c} + R_{\mathcal{P}_r} \\ &= R_{\mathcal{U}(a,b)\cap\mathcal{V}'(c,d)} + 2n - \log n - 1. \end{aligned} \quad (2)$$

Equation (2) follows from the fact that the $n - \log n - 1$ bits of $\mathbf{p}_c$ and the n bits of $\mathbf{p}_r$ are fixed to predetermined values.

We now compute an upper bound on the redundancy of the code $\mathcal{U}(a,b) \cap \mathcal{V}'(c,d)$.

Proposition 12 *There exists four values $a^\star, b^\star, c^\star$ and $d^\star$ for which the redundancy R_1 of $\mathcal{U}(a^\star,b^\star) \cap \mathcal{V}'(c^\star,d^\star)$ is bounded from above by*

$$R_1 < (2n - 2\log n - 3)\log\left(\frac{n}{n-1}\right) + 5\log n + 6. \quad (3)$$

From (2) and (3) we obtain $R_{\mathcal{C}_n(a,b,c,d)} < 2n + 4\log n + 5 + 2\log 2e$, where we use the inequalities $2n - 2\log n - 3 < 2n$ and $2n\log\left(\frac{n}{n-1}\right) \leqslant 2\log 2e = 2 + 2\log e$. This completes the proof of Corollary 10. We conclude this section with the proof of Proposition 12.

Proof of Proposition 12: We start with counting the number of arrays that satisfy all the imposed constraints except for the VT constraints in the codes $\mathcal{U}(a^\star,b^\star)$ and $\mathcal{V}'(c^\star,d^\star)$. To that end, we define the following three sets over $\Sigma^{n\times n}$.

$$\mathcal{U}_\perp \triangleq \left\{\mathbf{X}; \mathbf{X}_{[\ell],j} \neq \mathbf{X}_{[\ell],j+1}, \quad j \in [n-\ell-1]\right\},$$

$$\mathcal{V}_\perp \triangleq \left\{\mathbf{X}; \begin{array}{l} \mathbf{X}_{i,[n-\ell+1:n]} \neq \mathbf{X}_{i+1,[n-\ell+1:n]}, \quad \ell < i < n-1 \\ X_{\ell+1,n} \equiv \ell \mod 2 \end{array}\right\},$$

$$\mathcal{S}_\cap \triangleq \left\{\mathbf{X}; \begin{array}{l} \mathbf{X}_{[\ell],j} \neq \mathbf{X}_{[\ell],j+1}, \quad n-\ell \leqslant j < n, \\ \mathbf{X}_{i,[n-\ell+1:n]} \neq \mathbf{X}_{i+1,[n-\ell+1:n]}, i \in [\ell], \\ \mathbf{X}_{[4],n-1} = [0000]^T, \\ \mathbf{X}_{[\ell+1],n} = [010101\cdots]^T \end{array}\right\}.$$

Claim 13 *The redundancies of $\mathcal{U}_\perp$ and $\mathcal{V}_\perp$ are given by $R_{\mathcal{U}_\perp} = (n - \log n - 1)\log\left(\frac{n}{n-1}\right)$ and $R_{\mathcal{V}_\perp} = (n - \log n - 2)\log\left(\frac{n}{n-1}\right) + 1$, respectively.*

The intuition behind Claim 13 is that the first $\log n$ bits of any two consecutive columns of $\mathbf{U}$ (last $\log n$ bits of any two consecutive rows of $\mathbf{V}$) must be different.

Claim 14 *The redundancy of $\mathcal{S}_\cap$ is upper bounded by $R_{\mathcal{S}_\cap} < \log n + 5$.*

The intuition behind Claim 14 is that with at most one bit of redundancy we can guarantee that every two consecutive rows and every two consecutive columns of the $\log n \times \log n$ square are different.

We now count the number of arrays that satisfy the above requirements and have $\mathbf{U} \in \mathcal{VT}_{n,n}(a,b)$ and $\mathbf{V} \in \mathcal{VT}_{n-1,n}(c,d)$. Using the same arguments explained in Section II, we note that the VT constraints partition the set $\mathcal{U}_\perp \cap \mathcal{V}_\perp \cap \mathcal{S}_\cap$ into $(n^3)(n-1)$ disjoint cosets. Therefore, there exist $a^\star, d^\star, c^\star, d^\star$ for which

$$|\mathcal{U}(a^\star,b^\star) \cap \mathcal{V}(c^\star,d^\star)| \geqslant \frac{|\mathcal{U}_\perp \cap \mathcal{V}_\perp \cap \mathcal{S}_\cap|}{(n^3)(n-1)}.$$

In other words, the redundancy R_1 of $\mathcal{U}(a^\star,b^\star) \cap \mathcal{V}(c^\star,d^\star)$ is bounded from above by $R_1 < R_{\mathcal{U}_\perp\cap\mathcal{V}_\perp\cap\mathcal{S}_\cap} + \log(n^4)$, where we use $\log(n-1) < \log n$.

All the constraints in $\mathcal{U}_\perp$, $\mathcal{V}_\perp$, $\mathcal{S}_\cap$ are disjoint, thus $R_{\mathcal{U}_\perp\cap\mathcal{V}_\perp\cap\mathcal{S}_\cap} = R_{\mathcal{U}_\perp} + R_{\mathcal{V}_\perp} + R_{\mathcal{S}_\cap}$. We substitute the results from Claim 13 and Claim 14 to obtain the desired result. ■

REFERENCES

[1] 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.

[2] 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.

[3] 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

[4] 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.

[5] 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.

[6] G. Tenengolts, "Nonbinary codes, correcting single deletion or insertion (corresp.)," *IEEE Trans. Inf. Theory*, vol. 30, no. 5, pp. 766–769, 1984.

[7] R. Bitar, I. Smagloy, L. Welter, A. Wachter-Zeh, and E. Yaakobi, "Criss-cross deletion correcting codes (extended version)," *preprint*, 2020. [Online]. Available: https://sites.google.com/site/rawadbitar1/documents/CrissCross