Clustering-Correcting Codes

Tal Shinkar, *Student Member, IEEE* and Eitan Yaakobi, *Senior Member, IEEE*

***Abstract*—A new family of codes, called *clustering-correcting codes*, is presented in this paper. This family of codes is motivated by the special structure of data that is stored in DNA-based storage systems. The data stored in these systems has the form of unordered sequences, also called *strands*, and every strand is synthesized thousands to millions of times, where some of these copies are read back during sequencing. Due to the unordered structure of the strands, an important task in the decoding process is to place them in their correct order. This is usually accomplished by allocating a part of the strand for an index. However, in the presence of errors in the index field, important information on the order of the strands may be lost.**

Clustering-correcting codes ensure that if the distance between the index fields of two strands is small, then there will be a large distance between their data fields. It is shown how this property enables to place the strands together in their correct clusters even in the presence of errors. We present lower and upper bounds on the size of clustering-correcting codes and an explicit construction of these codes which uses only a single bit of redundancy.

I. Introduction

The idea of using DNA molecules as a volume for storing data was first introduced in 1959 [6]. DNA molecules have the unique qualities of density and durability that make them an attractive solution for storing archivable data. In 1990, the human genome project was initiated with the objective of determining the DNA sequence of the entire human genome, leading to a valuable progress in DNA sequencing and assembly methods. Understanding and manipulating DNA is catching a lot of focus and a rapid growth in DNA synthesis and sequencing methods (corresponds to DNA writing and reading) is still observed. DNA Synthesis is the process of creating DNA molecules. Arbitrary single-stranded DNA sequences of length a few hundreds can be synthesized chemically [7]. This process results in a DNA pool which consists of all the synthesized strands and does not permit structured addressing. Synthesis corresponds to writing data to the DNA storage. Sequencing is the process of reading sequences from the DNA pool. It is now acknowledged that within the next 10–15 years DNA storage may become a highly competitive archiving technology.

Progress in synthesis and sequencing technologies have paved the way for the development of a non-volatile data storage based on DNA molecules. The first large-scale experiments that demonstrated the potential of in vitro DNA storage were reported by Church et al. who recovered 643 KB of data [4] and Goldman et al. who accomplished the same task for a 739 KB message [8]. However, in both of these works the data was not recovered successfully due to the lack of using the appropriate coding solutions to correct errors. Since then, several more groups have demonstrated the ability to successfully store data of large scale using DNA molecules; see e.g. [1], [2], [5], [13], [18]. Other works developed coding solutions which are specifically targeted to correct the special types of errors inside DNA-based storage systems [10]–[12], [14], [16]–[18].

Church et al. had 10-bit errors and Goldman et al. lost two strands of 25 nucleotides. Later, in [19], Grass et al. reported the first system with a usage of error-correcting codes in DNA-based storage and managed to perfectly recover an 81 KB message. Bornholt et al. similarly retrieved a 42 KB message [2]. Since then, several groups have built similar systems, storing ever larger amounts of data. Among these, Erlich and Zielinski [5] stored 2.11MB of data with high storage rate, Blawat et al. [1] successfully stored 22MB, and more recently Organick et al. [13] stored 200MB. Yazdi et al. [18] developed a method that offers both random access and rewritable storage.

A DNA storage system consists of three steps. (see Fig. 1). First, a DNA synthesizer that produces strands that contain the encoded data to be stored in DNA. In order to produce strands with an acceptable error rate the length of the strands is typically limited to no more than 250 nucleotides. The second part is a storage container that stores the DNA strands unordered. The third part is a DNA sequencer that reads back the strands and restores the original, digital, data from them. The encoding and decoding are two external processes to the system that convert the data to DNA strands and back. The structure of a DNA storage system is different from all other existing storage systems. Since the strands are stored unordered, it is unclear what part of the data each strand represents, even if no error occurred. For more details we refer the reader to [9], [11] and referencers therein.

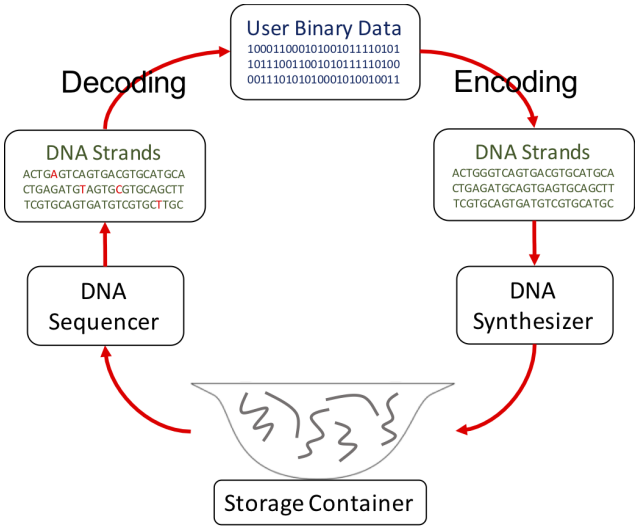

Fig. 1. Illustration of a DNA-based storage system.

Storing DNA strands in a way that will allow to reconstruct them back in the right order is an important task. The com-

mon solution to address this problem is to use indices, that are stored as part of the strand. Each strand is prefixed with some nucleotides that indicate the strand's location, with respect to all other strands. Although using indices is a simple solution it has several drawbacks. One of them is that in case of an error within the index, important information on the strand's location is lost as well as the ability to place it in the correct position between the other strands. In this paper a new coding scheme, called *clustering-correcting codes*, is presented which enables to combat errors with minimal redundancy.

In DNA storage systems, every strand is synthesized thousands of times (or even millions) and thus more than a single copy of each strand is read back upon sequencing. Thus, the first task based upon the sequencer's input is to partition all the reads into clusters such that all read strands at each cluster are copies of the same information strand. A possible solution is to use the indices in order to identify the strands and cluster them together, but in the presence of errors, this may result with misclustered strands which can cause errors in the recovered data. Hence, finding codes and algorithms for the clustering process is an important challenge. A naive solution is to add redundancy to the index part in order to correct potential errors in the index [3]. However, this will incur an unavoidable reduction in the storage rate of the DNA storage system. We will show in this paper how clustering-correcting codes can enable one to cluster all strands in the right clusters even with the presence of errors in the indices (see Fig. 2), while the redundancy is minimized. In fact, for a large range of parameters this can be done with only a single symbol of redundancy for all the strands together.

The rest of the paper is organized as follows. In Section II, some useful definitions that will be used throughout the paper are presented. In section III, the family of clustering-correcting codes and its capabilities are presented. In Section IV, we present explicit and asymptotic lower and upper bounds on the size of clustering-correcting codes. In Section V, we present an explicit construction of these codes which uses only a single symbol of redundancy. In Section VI we describe the modifications needed for out solution to work under the Edit distance metric. Lastly, Section VII concludes the paper.

II. Definitions and Preliminaries

The following notation will be used throughout the paper. For a positive integer n, the set $\{0,1,\ldots,n-1\}$ is denoted by $[n]$. For two vectors over the alphabet $\Sigma=\{0,1,\ldots,q\}$ of the same length, $\boldsymbol{x},\boldsymbol{y}\in[q]^n$, we denote the i-th symbol of $\boldsymbol{x}$ by x_i. The subvector of $\boldsymbol{x}$ starting at the i-th index of length ℓ is denoted by $\boldsymbol{x}_{[i,\ell]}$. We also denote the size of the vector $\boldsymbol{x}$ by $|\boldsymbol{x}|$. The Hamming distance between $\boldsymbol{x}$ and $\boldsymbol{y}$ is denoted by $\mathrm{d_H}(\boldsymbol{x},\boldsymbol{y})$ and the $L1$ weight of $\boldsymbol{x}$ is $w_1(\boldsymbol{x})\triangleq\sum_{i=0}^{n}x_i$. The radius-$r$ ball of a vector $\boldsymbol{x}\in[q]^n$ is $B_r(\boldsymbol{x})=\{\boldsymbol{y}\mid \mathrm{d_H}(\boldsymbol{x},\boldsymbol{y})\leqslant r\}$. Since the size of the ball $B_{\boldsymbol{x}}(r)$ does not depend on the choice of $\boldsymbol{x}$ we denote this size by $B_n(r)\triangleq\sum_{i=0}^{r}\binom{n}{i}(q-1)^i$ where n denotes the length of $\boldsymbol{x}$. The function $\mathcal{H}_q(x)$ for $0\leqslant x\leqslant 1$ denotes the q-ary entropy function, and the inverse function $\mathcal{H}_q^{-1}(x)$ for $0\leqslant x\leqslant 1$ is defined to return values between 0 and 1/2, throughout the paper, unless specified the $\log(a)\equiv log_q(a)$.

Assume M strands are stored in a DNA-based storage system where the size of every strand is L. We will assume that $M=q^{\beta L}$ for some $0<\beta<1$ and for simplicity, it is assumed that M is a power of q. For any integer $i\in[M]$, its q-ary representation of length $\log(M)$ is denoted by ind_i. Every length-L strand s that will be stored in the system is of the form $s=(\mathsf{ind},\boldsymbol{u})$, where ind is the length-$\log(M)$ *index field* of the strand (the representation of a number between 0 and $M-1$ above alphabet of size q) and $\boldsymbol{u}$ is the *data field* of $L-\log(M)$ symbols that are used to store the information or the redundancy of an error-correcting code and other coding schemes. Every stored message will have M strands of this form and the space of all possible messages that can be stored in the DNA storage system is denoted by

$$\mathcal{X}_{M,L}=\{\{(\mathsf{ind}_0,\boldsymbol{u}_0),\ldots,(\mathsf{ind}_{M-1},\boldsymbol{u}_{M-1})\}|\boldsymbol{u}_j\in[q]^{L-\log_q(M)}\}.$$

Clearly, $|\mathcal{X}_{M,L}|=q^{M(L-\log_q(M))}$. Under this setup, a code $\mathcal{C}$ will be a subset of $\mathcal{X}_{M,L,q}$, where each codeword $\mathcal{S}$ of $\mathcal{C}$ is a subset of the form $\mathcal{S}=\{(\mathsf{ind}_0,\boldsymbol{u}_0),\ldots,(\mathsf{ind}_{M-1},\boldsymbol{u}_{M-1})\}$. For shorthand, in the rest of the paper the term $L-\log_q(M)$ will be abbreviated by L_M.

When a set $\mathcal{S}=\{(\mathsf{ind}_0,\boldsymbol{u}_0),\ldots,(\mathsf{ind}_{M-1},\boldsymbol{u}_{M-1})\}$ is synthesized, each of its strands $(\mathsf{ind}_i,\boldsymbol{u}_i)$, which are called the *input strands*, has thousands to millions of copies and during the sequencing process a subset of these copies is read. Hence, the sequencer's output is another set G of some N strands, called the *output strands*, where N is significantly larger than M. Each output strand in the set G is a copy of one of the input strands in $\mathcal{S}$, however with some potential errors. A DNA-based storage system is called a (τ,ρ)-*DNA system* if it satisfies the following property: If the output strand $(\mathsf{ind}',\boldsymbol{u}')\in G$ is a noisy copy of the input strand $(\mathsf{ind},\boldsymbol{u})\in S$, then $\mathrm{d_H}(\mathsf{ind},\mathsf{ind}')\leqslant\tau$ and $\mathrm{d_H}(\boldsymbol{u},\boldsymbol{u}')\leqslant\rho$. That is, the index field has at most τ Hamming errors while the data field has at most ρ Hamming errors. We consider here only substitution errors while extensions for deletions, insertions, and more generally the Edit distance will be analyzed in section VI of the paper.

Since the set G contains several noisy copies of each input strand in $\mathcal{S}$, the first task in the decoding process is to partition the set of all N output strands into M cluster sets, such that the output strands in every cluster are noisy copies of the same input strand. Since every strand contains an index, the simplest way to operate this task is by partitioning the output strands into M sets based upon the index field in every output strand. This process will indeed be successful if there are no errors in the index field of every output strand, however other solutions are necessary since the error rates in DNA storage systems are not negligible [9]. Another approach to cluster the strands is based upon the distances between every pair of output strands, as was studied in [14]. However, this approach suffers extremely high computational complexity.

III. Clustering-Correcting Codes

In this work, we take a hybrid solution of these two approaches. First, we cluster the output strands based on the indices of the output strands. Then, we scan for output strands

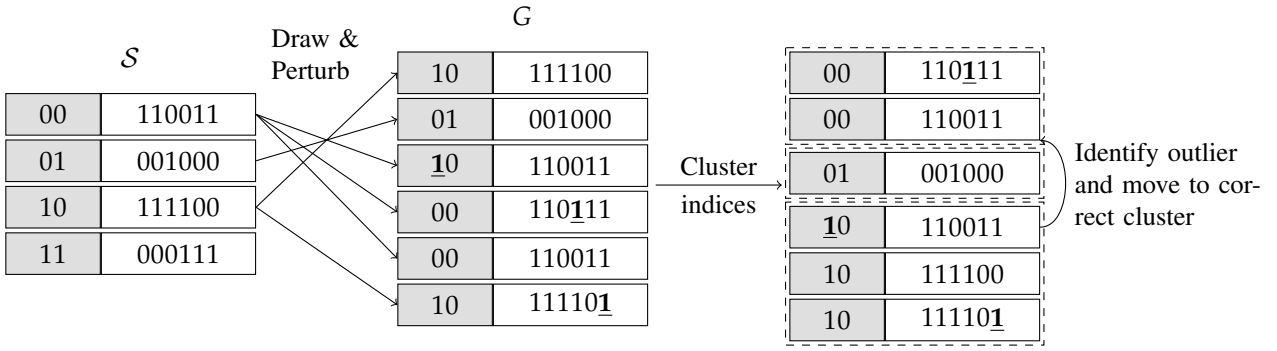

Fig. 2. Exemplary realization of the DNA channel model. A set $\mathcal{S}$ of $M = 4$ binary strands is stored and $N = 6$ strands are drawn with errors (highlighted in bold). The strands are clustered according to their indices. The outlier can be identified as it has large distance w.r.t. all other strands in the cluster and be put into the correct cluster.

which were mis-clustered, that is, were placed in the wrong cluster. This is accomplished by checking the distances between the output strands in every cluster in order to either remove output strands that were incorrectly placed in a cluster due to errors in their index or move them to their correct cluster set. Since we compute the distances only between pairs of strands that were placed in the same cluster (and not between all pairs of strands), this step will result in a significantly lower complexity compared to the solution from [14], In fact, not all pairwise distances in a cluster have to be computed for this operation. That is, the complexity of this step can be reduced even more. However, in order to succeed in this new approach we need the strands stored in the set $\mathcal{S}$ to satisfy several constraints. These constraints will be met by the family of *clustering-correcting codes* which are presented in this paper. Another assumption taken in this model, which will be referred to as the *majority assumption*, assumes that in every cluster the majority of the strands have the correct index. Since the number of strands is very large this assumption holds with high probability if not in all cases. Another assumption, which will be referred as the *dominance assumption*, assumes that if a cluster is partitioned into subsets based on the correct origin of each strand (their true index), the largest subset contains the strands that does not have errors in their index and therefore, clustered correctly. Hence, the strands with a correct index are dominant. Note that majority also implies dominance.

The main idea to move strands which were misplaced in a cluster due to errors in their index field works as follows. Assume the strand $s_i = (\mathsf{ind}_i, \boldsymbol{u}_i)$ has a noisy copy of the form $s'_i = (\mathsf{ind}'_i, \boldsymbol{u}'_i)$, and let j be such that $s_j = (\mathsf{ind}_j, \boldsymbol{u}_j)$ and $\mathsf{ind}_j = \mathsf{ind}'_i$. We need to make sure that the distance between $\boldsymbol{u}'_i$ and $\boldsymbol{u}_j$ is large enough as, this will allow to identify that the output strand s'_i is erroneous and therefore does not belong to the cluster of index ind_j; see Fig. 2. We will be interested in either identifying that the output strand s'_i does not belong to this cluster or more than that, place it in its correct cluster. This motivates us to study the following family of codes.

Definition 1. *A word $\mathcal{S} = \{(\mathsf{ind}_0, \boldsymbol{u}_0), \ldots, (\mathsf{ind}_{M-1}, \boldsymbol{u}_{M-1})\} \in \mathcal{X}_{M,L,q}$ is said to satisfy the (e,t)-**clustering constraint** if for all $(\mathsf{ind}_i, \boldsymbol{u}_i), (\mathsf{ind}_j, \boldsymbol{u}_j) \in \mathcal{S}$ in which $i \neq j$ and $\mathrm{d_H}(\mathsf{ind}_i, \mathsf{ind}_j) \leqslant e$, it holds that $\mathrm{d_H}(\boldsymbol{u}_i, \boldsymbol{u}_j) \geqslant t$.*

*A code $\mathcal{C} \subseteq \mathcal{X}_{M,L,q}$ will be called an (e,t)-**clustering-correcting code (CCC)** if every $\mathcal{S} \in \mathcal{C}$ satisfies the (e,t)-clustering constraint.*

The *redundancy* of a code $\mathcal{C} \subseteq \mathcal{X}_{M,L}$ will be defined by

$$r = ML_M - \log_q |\mathcal{C}|.$$

Our goal in this work is to find (e,t)-CCCs for all e and t. We denote by $A_{M,L,q}(e,t)$ the size of the largest (e,t)-CCC in $\mathcal{X}_{M,L,q}$, and by $r_{M,L,q}(e,t)$ the optimal redundancy of an (e,t)-CCC, so $r_{M,L,q}(e,t) = ML_M - \log_q(A_{M,L,q}(e,t))$.

The clustering-correcting capabilities of CCCs are proved in the next theorem. We note that as a result of this theorem, if the number of errors is not too large, it is already possible to place every output strand in its correct cluster.

Theorem 2. *For fixed integers M, L, q, e, t, let $\mathcal{C}$ be an (e,t)-CCC. Assume that a set $\mathcal{S} \in \mathcal{C}$ is stored in a (τ,ρ)-DNA system. The following properties hold:*

1) *Under the majority assumption, If $\tau \leqslant e$ and $4\rho < t$ then every output strand can be detected to be placed in a wrong cluster.*
2) *Under the dominance assumption, If $\tau \leqslant e/2$ and $4\rho < t$ then every output strand can be placed in its correct cluster.*

Proof:

To prove 1), Let $(\mathsf{ind}'_i, \boldsymbol{u}'_i)$ be a noisy copy of the strand $(\mathsf{ind}_i, \boldsymbol{u}_i)$. Since the data is stored in a (τ,ρ)-DNA system, it holds that $\mathrm{d_H}(\mathsf{ind}_i, \mathsf{ind}'_i) \leqslant \tau$, and therefore $\mathrm{d_H}(\mathsf{ind}_i, \mathsf{ind}'_i) \leqslant e$. Also $\mathrm{d_H}(\boldsymbol{u}_i, \boldsymbol{u}'_i) \leqslant \rho$. Let $j \in [M]$ be such that $\mathsf{ind}_j = \mathsf{ind}'_i$. From the fact that $S \in \mathcal{C}$ we derive that $\mathrm{d_H}(\boldsymbol{u}_i, \boldsymbol{u}_j) \geqslant t > 4\rho$, and thus $\mathrm{d_H}(\boldsymbol{u}_j, \boldsymbol{u}'_i) \geqslant \mathrm{d_H}(\boldsymbol{u}_i, \boldsymbol{u}_j) - \mathrm{d_H}(\boldsymbol{u}_i, \boldsymbol{u}'_i) > 3\rho$. Let $(\mathsf{ind}_j, \boldsymbol{u}'_j)$ be a noisy copy strand of $(\mathsf{ind}_j, \boldsymbol{u}_j)$, that is, errors might occur in the data field but not in the index field. So, $\mathrm{d_H}(\boldsymbol{u}_j, \boldsymbol{u}'_j) \leqslant \rho$ which yields that $\mathrm{d_H}(\boldsymbol{u}'_i, \boldsymbol{u}'_j) > 2\rho$. On the other hand, the distance between the data fields of the two strands that belong to the same cluster is at most 2ρ. That is, under the majority assumption, a mis-clustered strand will have a distance of more than 2ρ from the majority of the strands in the cluster, and so it can be dropped instead of being mis-clustered.

To prove 2), Let $(\mathsf{ind}_j, \boldsymbol{u}'_i), (\mathsf{ind}_j, \boldsymbol{u}'_k)$ be two noisy copies of $(\mathsf{ind}_i, \boldsymbol{u}_i), (\mathsf{ind}_k, \boldsymbol{u}_k)$ respectively that ended up in the same cluster. It holds that $\mathrm{d_H}(\mathsf{ind}_j, \mathsf{ind}_i) \leqslant \tau \leqslant \frac{e}{2}$ and also, $\mathrm{d_H}(\mathsf{ind}_j, \mathsf{ind}_k) \leqslant \tau \leqslant \frac{e}{2}$. Hence, $\mathrm{d_H}(\mathsf{ind}_i, \mathsf{ind}_k) \leqslant e$, and therfore they satisfy the clustering constraint in respect to one another. That is, for $i \neq k$ it holds that $\mathrm{d_H}(\boldsymbol{u}_i, \boldsymbol{u}_k) \geqslant t$. Observe that if the Hamming distance between two data fields is $\leqslant 2\rho$, the two strand must be originated in the same cluster, hence, have the same index. This way, we can partition each cluster into homogenous subsets such that each subset contains copies of the same information strand. Only one of those sets contain strands with a correct index. Under the dominance assumption, we are able to identify which one by its size, as it must be the most dominant subset, the largest among all subsets.

After applying this partitioning to all clusters we want to move subsets that are not dominant their correct clusters. Let $s'_i = (\mathsf{ind}'_i, \boldsymbol{u}'_i)$ be a noisy copy of the strand $s_i = (\mathsf{ind}_i, \boldsymbol{u}_i)$, which is also a representative of one of those subsets. We have seen that if some other index ind_k can also end up as ind'_i, or equivalently $\mathrm{d_H}(ind'_i, ind_k) \leqslant \tau$, than strands for the corresponding indices ind_i and ind_k satisfy the clustering constraint in respect to one another. Therefore given $s'_k = (\mathsf{ind}_k, \boldsymbol{u}'_k)$, a noisy copy with a correct index, it holds that $\mathrm{d_H}(\mathsf{ind}_i, \mathsf{ind}_k) \leqslant e$, and hence, the strand will be detected as erroneous in any cluster k except for $k = i$. That is, we can find the correct cluster index i that the noisy strand (and the whole subset) belongs to.

■

The clustering algorithm according to Theorem 2 first partitions the strands to clusters according to the indices and then compares the distances between *only* strands in the same cluster. Hence, the number of comparisons is significantly smaller than the solution in [14], which compares between *all* strands.

Going over the proof of 1) it's clear now why reading data stored with an (e,t)- clustering constraint is efficient. After creating clusters using the read indices, comparing the strands in a cluster results with immediate identification of all strands that have correct index. For 2) the same process will generate subsets that are homogenous, That is, each subset can be associated with a single index. This makes the correction very efficient as it can be done with a single representative from each subset. The discussed method is more efficient in the number of comparisons needed to perform the clustering, compared to the solution in [14]. It's also results with better accuracy as defined in that paper. This is because in case 2) we get 100% accuracy.

Corollary 3. *For fixed integers M, L, q, e, t, let $\mathcal{C}$ be an (e,t)-CCC. Assume that a set $\mathcal{S} \in \mathcal{C}$ is stored in a (τ,ρ)-DNA system. In addition, let $\mathcal{R}$ be the number of reads upon sequencing.*

1) *under the same setting as Theorem 2 case (1), the complexity of identifying the strands with a correct index is $O(B_{\log(M)}(\tau) \cdot \mathcal{R} \cdot L)$ symbol operations.*
2) *under the same setting as Theorem 2 case (2), the complexity of placing the mis-clustered strands in the correct cluster is $O((B_{\log(M)}(\tau))^2 \cdot M \cdot L)$ symbol operations.*

Therefore, the whole process can be done with

$$O(B_{\log(M)}(\tau) \cdot L \cdot (\mathcal{R} + B_{\log(M)}(\tau) \cdot M))$$

symbol operations.

Proof: to prove 1), we recall that in Theorem 2 case (1) we proved that a strand with an error in its index can be identified by comparing its data part with the data parts of the rest of the strands in the cluster. If the strand has, indeed, an error in its index, it will have a distance greater than 2ρ from the majority of the strands in the cluster. Note that, in this scenario, any strand that has close data, hence, at most 2ρ must also have an error in the index. Otherwise, we get a contradiction for the clustering constraint itself.

Using this observation, we can pick some strand in the cluster and compare it with the others. If it has a correct index, all those that where far from it have an error in their indices and the error-identification step is done. In case we picked a strand that was mis-clustered, all strands that where close to it have an error in their index as well. In particular, this is true for the strands that originated in the same cluster as this mis-clustered strand.

Let $\mathcal{I} = \{\mathsf{ind}_{i_1}, \ldots, \mathsf{ind}_{i_k}\}$ be the set of all correct indices of the mis-clustered strands in the cluster. In each such pass, we clear one of these indices out of the cluster. As $|\mathcal{I}| \leqslant B_{\log(M)}(\tau)$ we can bound the number of iterations needed to successfully clear the nose out of the cluster, and its at most $B_{\log(M)}(\tau)$. Denote by $C_0, C_2, \ldots, C_{M-1}$ the sizes of the different read clusters, $\sum_{i=0}^{m-1} C_i = \mathcal{R}$. The amount of symbol operations required to clean the i_{th} cluster is

$$O(B_{\log(M)}(\tau) \cdot C_i \cdot L)$$

symbol operations. Summing up for all clusters we get overall complexity of

$$O(B_{\log(M)}(\tau) \cdot \mathcal{R} \cdot L).$$

to prove 2), we make the following observation: in the proof of case (2) of Theorem 2 we have seen that any two strands read with the same index satisfy the clustering constraint in respect to one another. That is, if two strands data parts have distance at most 2ρ, the two strands originated in the same cluster. Hence, the process described in the proof of 1) will result with homogenous subsets where the largest contains the strands with the correct index. At this point, the other strands true cluster can be found by taking a representative of each subset and compare it with a single strand from each of the neighbor clusters, as we have proved only the correct cluster will match. This step requires $(B_{\log(M)}(\tau))^2$ comparisons per cluster, and an overall of

$$O((B_{\log(M)}(\tau))^2 \cdot M \cdot L)$$

symbol operations over all clusters together.

■

IV. Bounds

Upper and lower bounds on $A_{M,L,q}(e,t)$ are presented. Let $D_{n,q}(d)$ be the size of the largest length-n error-correcting code

$C \subseteq [q]^n$ with minimum Hamming distance d. For the rest of the paper, let $B_1 = B_{\log(M)}(e) - 1, B_2 = B_{L_M}(t-1)$, $\mathsf{le} = \log(\exp(1)) \approx 1.44$, and it is also assumed that $\beta < 1/2$.

Theorem 4. *For all M, L, q, e, and t it holds that*

$$A_{M,L,q}(e,t) \geqslant q^{ML_M}\left(1 - \frac{B_1 B_2}{q^{L_M}}\right)^{M-D}$$

and hence

$$r_{M,L,q}(e,t) < \frac{\mathsf{le}\cdot(M-D)B_1B_2}{q^{L_M} - B_1B_2},$$

where $D = D_{\log(M),q}(e+1)$.

Proof: In order to verify the lower bound, we construct an (e,t)-CCC $\mathcal{C}$ that will yield a lower bound on $A_{M,L,q}(e,t)$. Let $C_1 \subseteq [q]^{\log(M)}$ be a length-$\log(M)$ code with minimum Hamming distance $e+1$ of size D. Each codeword $\mathcal{S} = \{(\mathsf{ind}_0, \boldsymbol{u}_0), \ldots, (\mathsf{ind}_{M-1}, \boldsymbol{u}_{M-1})\} \in \mathcal{C}$ is constructed in two steps. First, we choose the data field of strands with indices that belong to the code C_1, that is, all strands of the form $(\mathsf{ind}, \boldsymbol{u})$ such that $\mathsf{ind} \in C_1$. There are q^{L_M} options for each strand and thus $(q^{L_M})^D$ options for the first step. Since the Hamming distance between all pairs of indices of these strands is at least $e+1$, their data fields can be chosen independently.

For the rest of the strands we assume the worst case. That is, for each strand left, all of its neighbors are chosen and their radius-$(t-1)$ balls are mutually disjoint. Thus, there are at least

$$q^{L_M} - (B_{\log(M)}(e) - 1)\cdot B_{L_M}(t-1) = q^{L_M} - B_1B_2$$

options to choose the data field of each remaining strand. In conclusion, there are $q^{L_M D}\left(q^{L_M} - B_1B_2\right)^{M-D}$ options for choosing a valid set $\mathcal{S} \in \mathcal{C}$, and hence

$$A_{M,L,q}(e,t) \geqslant q^{ML_M}\left(1 - \frac{B_1B_2}{q^{L_M}}\right)^{M-D}.$$

We can also deduce an upper bound on the redundancy

$$\begin{aligned} r_{M,L,q}(e,t) &= M\cdot L_M - \log(A_{M,L}(e,t)) \\ &\leqslant -(M-D)\cdot\log\left(1 - \frac{B_1\cdot B_2}{2^{L_M}}\right) \\ &< \frac{\mathsf{le}\cdot(M-D)B_1B_2}{q^{L_M} - B_1B_2}, \end{aligned}$$

where here the inequality $-\log(1-x) \leqslant \mathsf{le}\cdot\frac{x}{1-x}$ for all $0 < x < 1$ is used. ∎

The next corollary follows directly from Theorem 4.

Corollary 5. *For all t such that*

$$t \leqslant L_M\cdot\mathcal{H}_q^{-1}\left(\frac{1-2\beta}{1-\beta} - \frac{\log(\mathsf{le}) + e\log(\beta L(q-1))}{(1-\beta)L}\right)$$

it holds that $r_{M,L,q}(e,t) < 1$.

Proof: From Theorem 4 it holds that

$$r_{M,L,q}(e,t) < \frac{\mathsf{le}\cdot(M-D)B_1B_2}{q^{L_M} - B_1B_2}.$$

Hence, $r_{M,L,q}(e,t) < 1$ if $B_2 \leqslant \frac{q^{L_M}}{B_1(\mathsf{le}\cdot(M-D)+1)}$. We know that $B_1 \leqslant (\log(M)(q-1))^e$. Also, according to Lemma 4.7 in [15], $B_2 \leqslant q^{L_M\mathcal{H}_q\left(\frac{t-1}{L_M}\right)}$ and hence it is enough to require that $q^{L_M\mathcal{H}_q\left(\frac{t-1}{L_M}\right)} \leqslant \frac{q^{L_M}}{B_1(\mathsf{le}\cdot(M-D)+1)}$, i.e.,

$$\begin{aligned} L_M\mathcal{H}_q\left(\frac{t-1}{L_M}\right) &\leqslant L_M - \log(M) - e\log\log(M) - e\log(q-1) \\ &\leqslant L_M - \log(\mathsf{le}) - \log(M) - \log(B_1) \\ &\leqslant L_M - \log(1 + \mathsf{le}\cdot(M-D)) - \log(B_1). \end{aligned}$$

For $M = q^{\beta L}$, this holds for all

$$t \leqslant L_M\mathcal{H}_q^{-1}\left(\frac{1-2\beta}{1-\beta} - \frac{\log(\mathsf{le}) + e\log(\beta L(q-1))}{(1-\beta)L}\right).$$

∎

A similar upper bound on $A_{M,L,q}(e,t)$ is presented next.

Theorem 6. *For all M, L, e and t it holds that*

$$A_{M,L,q}(e,t) \leqslant q^{M\cdot L_M}\left(1 - \frac{B_2}{q^{L_M}}\right)^{M-1},$$

and hence

$$r_{M,L}(e,t) > \frac{\mathsf{le}\cdot(M-1)\cdot B_2}{q^{L_M}}$$

Proof: Let $\mathcal{C}$ be an (e,t)-CCC of maximal size $A_{M,L}(e,t)$. For the upper bound, we count the number of vectors we can see as the data field in every strand while scanning the strands according to the indices. This scanning is operated by the weight of the index vector and arbitrarily in every group of indices with the same weight. For the first strand of weight zero, any length-L_M vector can be assigned for its data field. For every other strand we only take into account the constraint by the strands that were already scanned and assigned with their data field. Furthermore, we assume the worst case in which the neighbors of all these strands are the same. Hence, for each strand, besides the first one, we have at most $q^{L_M} - B_2$ options for its data field, and all together we get

$$\begin{aligned} A_{M,L}(e,t) &\leqslant q^{L_M}\left(q^{L_M} - B_2\right)^{M-1} \\ &= q^{M\cdot L_M}\left(1 - \frac{B_2}{q^{L_M}}\right)^{M-1}. \end{aligned}$$

Lastly, the lower bound on the redundancy is derived to be

$$\begin{aligned} r_{M,L}(e,t) &= M\cdot L_M - \log(A_{M,L}(e,t)) \\ &\geqslant -(M-1)\cdot\log\left(1 - \frac{B_2}{q^{L_M}}\right) \\ &> \frac{\mathsf{le}\cdot(M-1)\cdot B_2}{q^{L_M}}, \end{aligned}$$

where in the last step we used the inequality $-\log(1-x) > x\cdot\mathsf{le}$ for all $0 < x < 1$. ∎

Corollary 7. *For all t such that*

$$t \geqslant L_M\mathcal{H}_q^{-1}\left(\frac{1-2\beta}{1-\beta} + \frac{\log(L_M)}{(1-\beta)L}\right)$$

it holds that $r_{M,L,q}(e,t) > 1$.

Proof: From Theorem 6 it holds that

$$r_{M,L}(e,t) > \frac{\mathsf{le}\cdot(M-1)\cdot B_2}{q^{L_M}}$$

Hence, $r_{M,L,q}(e,t) > 1$ if $B_2 \geqslant \frac{q^{L_M}}{\mathsf{le}\cdot(M-1)}$. According to Lemma 4.8 in [15], $B_2 \geqslant \frac{1}{L_M+1}\cdot q^{L_M\mathcal{H}_q\left(\frac{t-1}{L_M}\right)}$ and hence it is enough to require that $\frac{1}{L_M+1}\cdot q^{L_M\mathcal{H}_q\left(\frac{t-1}{L_M}\right)} \geqslant \frac{q^{L_M}}{\mathsf{le}\cdot(M-1)}$, i.e.,

$$\begin{aligned} L_M\mathcal{H}_q\left(\frac{t-1}{L_M}\right) &\geqslant L_M - \log(M) + \log(L_M+1) \\ &\geqslant L_M - \log(\mathsf{le}) - \log(M) + \log(L_M+1). \end{aligned}$$

For $M = q^{\beta L}$, this holds for all

$$t \geqslant L_M\mathcal{H}_q^{-1}\left(\frac{1-2\beta}{1-\beta} + \frac{\log(L_M)}{(1-\beta)L}\right).$$

■

From Corollary 5 and Corollary 7 we get that for L_M large enough and $t \approx L_M\cdot\mathcal{H}^{-1}\left(\frac{1-2\beta}{1-\beta}\right)$, we get that $r_{M,L}(e,t) \approx 1$.

Next we have a slightly different lower bound that is more similar to the upper bound we achieved. Also, this bound for $r_{M,L,q}(e,t)$ is actually more tight as D is at most $M/2$.

Theorem 8. *For all $M, L, q, e,$ and t it holds that*

$$A_{M,L,q}(e,t) \geqslant q^{ML_M}\left(1-\frac{B_2}{q^{L_M}}\right)^{B1\cdot M/2}$$

and hence

$$r_{M,L,q}(e,t) \leqslant \frac{\mathsf{le}\cdot M\cdot B_1B_2}{2q^{L_M}-2B_2}.$$

Proof: In order to achieve this lower bound we use the probabilistic method. We construct a message $\mathcal{S} \in \mathcal{X}_{M,L,q}$ by choosing the values for the data parts of all strands in the codeword uniformly and independently. Next we bound the probability that $\mathcal{S}$ is a valid $(e,t)-$clustering-constraint codeword from below. Lastly, we use this probability to achieve a lower bound on the size of $A_{M,L,q}(e,t)$.

We define $E_{i,j}$ to be the event where the two strands $(\mathsf{ind}_i,\boldsymbol{u}_i),(\mathsf{ind}_j,\boldsymbol{u}_j)$ violates the clustering constraint in respect to one another. Hence, $E_{I,j}$ is the event that satisfies $\mathrm{d_H}(\mathsf{ind}_i,\mathsf{ind}_j) \leqslant e \wedge \mathrm{d_H}(\boldsymbol{u}_i,\boldsymbol{u}_j) < t$. The probability for each of those events is not dependent on i,j and equals to

$$Pr[E_{i,j}] = \frac{B_2}{q^{L_M}} = p.$$

Also, the number of events as desribed is $\frac{M\cdot B_1}{2}$. A codeword $\mathcal{S}$ formed this way is a valid clustering-correcting codeword if for all i,j the event $E_{i,j}$ does not occur. We trivially have this probability $Pr[\wedge\overline{E_{i,j}}] \geqslant (1-p)^{B_1\cdot M/2}$ and therefore,

$$A_{M,L,q}(e,t) \geqslant q^{ML_M}\left(1-\frac{B_2}{q^{L_M}}\right)^{B1\cdot M/2}.$$

The last transition is valid as if the size of $A_{M,L,q}(e,t)$ is lower, the probability of the chosen message $\mathcal{S}$ to be a valid $(e,t)-$clustering-correcting codeword would be lower as well.

We can also deduce an upper bound on the redundancy

$$\begin{aligned} r_{M,L,q}(e,t) &= M\cdot L_M - \log(A_{M,L}(e,t)) \\ &\leqslant -\left(\frac{M\cdot B_1}{2}\right)\cdot\log\left(1-\frac{B_2}{q^{L_M}}\right) \\ &< \frac{\mathsf{le}\cdot M\cdot B_1B_2}{2q^{L_M}-2B_2}, \end{aligned}$$

where here the inequality $-\log(1-x) \leqslant \mathsf{le}\cdot\frac{x}{1-x}$ for all $0 < x < 1$ is used. ■

V. A Construction of CCCs

In this section we propose a construction of CCCs. It is shown that with a single symbol of redundancy it is possible to construct CCCs for relatively large values of t.

The algorithm will use the following functions:

- The function $w_\ell(S,t)$ is defined over a set of vectors S and a positive integer t and outputs a vector $\boldsymbol{w} \in \{0,1\}^\ell$ which satisfies the following condition. For all $\boldsymbol{v} \in S$, $\mathrm{d_H}(\boldsymbol{w},\boldsymbol{v}_{[\log(M),\ell]}) \geqslant t$. The value of ℓ will be determined later as a function of e, t, and M.
- The function $\Delta_1(\mathsf{ind}_i,\mathsf{ind}_j)$ encodes the difference between the two indices i and j of Hamming distance at most e using $e\lceil\log(\log(M)\cdot(q-1))\rceil$ symbols which mark the positions where the indices $\mathsf{ind}_i,\mathsf{ind}_j$ differ.
- The function $\Delta_2(\boldsymbol{u}_i,\boldsymbol{u}_j)$ encodes the difference between the two data fields $\boldsymbol{u}_i,\boldsymbol{u}_j \in [q]^{L_M}$ of Hamming distance at most $t-1$ using $(t-1)\lceil\log(L_M\cdot(q-1))\rceil$ symbols which mark the positions where they differ. Its possible that there are less than $t-1$ such indices. In such case we replicate the last index to get the desired length. All indices written in an increasing order, that way if $\boldsymbol{u}_i = \boldsymbol{u}_j$, $\Delta_2(\boldsymbol{u}_i,\boldsymbol{u}_j)$ output will be an unsorted sequence of indices.

The input to the algorithm is M vectors $\boldsymbol{v}_0,\ldots,\boldsymbol{v}_{M-1}$. All vectors are of length L_M symbol, except for $\boldsymbol{v}_{M-1}$ which has length of L_M-1 symbols. The idea behind the presented algorithm is to find all pairs of vectors that do not satisfy the clustering constraint, and correct them in a way that they satisfy the constraint and yet the original data can be uniquely recovered. A symbol is added to $\boldsymbol{v}_{M-1}$, hence, the code has a single symbol of redundancy, to mark whether some vectors were altered by the algorithm.

For $i \in [M]$, the notation $S(e,i)$ in the algorithm will be used as a shortcut to the set $\{\boldsymbol{u}_j \mid \mathrm{d_H}(\mathsf{ind}_i,\mathsf{ind}_j) \leqslant e\}$ of data fields corresponding to indices ind_j of Hamming distance at most e from ind_i. At any iteration of the while loop, when the i-th strand is corrected, the function $w_\ell(S(e,i),t)$ will be used to update the data field $\boldsymbol{u}_i$ such that it does not violate the constraint and yet can be decoded. In order to make room for the vector generated by the function $w_\ell(S(e,i),t)$, we will encode $\boldsymbol{u}_i$ based on its similarity to one of its close neighbors $\boldsymbol{u}_j$. These modifications are encoded together as a repelling vector

of length $len = \ell + e\lceil\log(\log(M))\rceil + (t-1)\log(L_M)$. Finally we are chaining the vectors that we have altered, starting at $\boldsymbol{u}_{M-1}$, so we can traverse those and restore the original data.

Algorithm 1 $(e,t) - CCC$ Construction

Input: M vectors $\boldsymbol{v}_0,\ldots,\boldsymbol{v}_{M-1}$ such that $\boldsymbol{v}_0,\ldots,\boldsymbol{v}_{M-2} \in [q]^{L_M}$ and $\boldsymbol{v}_{M-1} \in [q]^{L_M-1}$

Output: a codeword $\mathcal{S} = \{(\mathsf{ind}_0,\boldsymbol{u}_0),\ldots,(\mathsf{ind}_{M-1},\boldsymbol{u}_{M-1})\}$

1: $\forall i \in \{0,\ldots,M-2\} : \boldsymbol{u}_i = \boldsymbol{v}_i, \boldsymbol{u}_{M-1} = (\boldsymbol{v}_{M-1},0)$
2: $p = M-1$
3: $B = \{(i,j) \mid i<j, \mathrm{d_H}(\mathsf{ind}_i,\mathsf{ind}_j) \leqslant e \wedge \mathrm{d_H}(\boldsymbol{u}_i,\boldsymbol{u}_j) < t\}$
4: **while** $B \neq \emptyset$, let $(i,j) \in B$ **do**
5: $\quad(\boldsymbol{u}_p)_{L_{M-1}} = 1$
6: $\quad(\boldsymbol{u}_p)_{[0,\log(M)]} = \mathsf{ind}_i$
7: $\quad p = i$
8: $\quad repl = (w_\ell(S(e,i),t), \Delta_1(\mathsf{ind}_i,\mathsf{ind}_j), \Delta_2(\boldsymbol{u}_i,\boldsymbol{u}_j))$
9: $\quad(\boldsymbol{u}_i)_{[\log(M),len]} = repl$
10: $\quad B = \{(i,j) \mid (i,j) \in B \wedge \mathrm{d_H}(\boldsymbol{u}_i,\boldsymbol{u}_j) < t\}$
11: $(\boldsymbol{u}_p)_{L_M-1} = 0$
12: $(\boldsymbol{u}_p)_{[0,\log(M)]} = (\boldsymbol{v}_{M-1})_{[0,\log(M)]}$

Theorem 9. *For any input vectors $\boldsymbol{v}_0,\ldots,\boldsymbol{v}_{M-1}$, Algorithm 1 returns a valid (e,t)-CCC codeword for any t that satisfies:*

$$L - 2\log(M) \geqslant \ell + \log(B_{\log(M)}(e)\cdot B_{L_M}(t-1)) + 1.$$

Furthermore, it is possible to decode the vectors $\boldsymbol{v}_0,\ldots,\boldsymbol{v}_{M-1}$, and Algorithm 1 uses a single symbol of redundancy.

Proof: Algorithm 1 starts by initializing the data fields $\boldsymbol{u}_0,\ldots,\boldsymbol{u}_{M-1}$ of the output set $\mathcal{S}$, with the input vectors $\boldsymbol{v}_0,\ldots,\boldsymbol{v}_{M-1}$, while adding the zero symbol at the end of the data field $\boldsymbol{u}_{M-1}$ to mark that no operation is needed for the decoding at this point. in Step 3 the algorithm gathers the indices of all pairs of strands that both their index and data fields are too close to each other, hence, violating the constraint. The algorithm iterates over the set B, handling one pair at a time. In Step 10, this set is updated and the algorithm stops when the set B is empty, i.e., there are no bad pairs and so $\{(\mathsf{ind}_0,\boldsymbol{u}_0),\ldots,(\mathsf{ind}_{M-1},\boldsymbol{u}_{M-1})\}$ satisfies the constraint.

On each iteration of the while loop, the algorithm takes a pair of strands, say of indices i and j, where $i<j$, which violates the constraint and changes the data field in the i-th strand. First, the flag symbol at the end of the previous strand is changed to 1 (Step 5). This denotes that it is not the last strand in the decoding chain. In Step 8, the algorithm calculates the vector $\boldsymbol{w} = w_\ell(S(e,i),t)$ and embeds it in the data field of the i-th strand. The vector $\boldsymbol{w}$ satisfies that for all $\boldsymbol{u} \in S(e,i)$, $\mathrm{d_H}(\boldsymbol{w},\boldsymbol{u}_{[\log(M),\ell]}) \geqslant t$. Then, for all $\boldsymbol{u} \in S(e,i)$ we have that $\mathrm{d_H}((\boldsymbol{u}_i)_{[\log(M),\ell]},\boldsymbol{u}_{[\log(M),\ell]}) \geqslant t$ and lastly for all $\boldsymbol{u} \in S(e,i)$, $\mathrm{d_H}(\boldsymbol{u}_i,\boldsymbol{u}) \geqslant t$. Therefore the i-th and the j-th strands satisfy the constraint and thus do not belong to the set B when it is updated in Step 10. In fact any bad pair of indices which includes the i-th strand will be removed as well from the set B. Furthermore, since the i-th strand has been updated in such a way that it satisfies the constraint with respect to all of its neighbors, and the p-th strand in this iteration already satisfies the constraint according to his last $L_M - \log(M)$ symbols, no bad pairs have been created. That is, the size of the set B decreases in each iteration, and the algorithm terminates. The constraint

$$L - 2\log(M) \geqslant \ell + \log(B_{\log(M)}(e)\cdot B_{L_M}(t-1)) + 1,$$

guarantees that the data field is large enough in order to write the information required on each update step of the while loop.

The chain always starts with $\boldsymbol{u}_{M-1}$. We can not alter $\boldsymbol{u}_{M-1}$, because it results with loss of parts of the chain. To make sure it will not happen we always choose to alter the data field of the smaller index in the pair. We need additional $\log(M)$ symbols to encode the index of the first strand in the chain. For this purpose, we take the first $\log(M)$ symbols of $\boldsymbol{u}_{M-1}$ and we move them to the end of the chain. The last strand in the chain has $\log(M)$ spare symbols, as we do not need to encode the index for the next one.

The idea of the decoding process is to track the updates chain of the strands and then traverse the chain in the opposite direction while recovering the input vectors. The decoding process starts with the $(M-1)$-st strand. If the flag at the end of the data field $\boldsymbol{u}_{M-1}$ is zero then there is nothing to be done. Otherwise, the first $\log(M)$ symbols of $\boldsymbol{u}_{M-1}$ indicate the index ind_i of the first altered vector $\boldsymbol{u}_i$. This process repeats until we encounter a vector with a flag symbol of value zero. This is the last vector in the chain, and furthermore, its first $\log(M)$ bits of the data field are the first $\log(M)$ bits of $\boldsymbol{u}_{M-1}$ before encoding. Now, we traverse the chain in the opposite direction, fixing each vector using the vectors $\Delta_1(\mathsf{ind}_i,\mathsf{ind}_j), \Delta_2(\boldsymbol{u}_i,\boldsymbol{u}_j)$ that are encoded in it. We stop at $\boldsymbol{u}_{M-1}$ which requires only to place back the original $\log(M)$ bits. This works as the order we go through the chain in the second time is exactly the inverse of the encoding order. This way all the encoding operations can be reverted, just as running the algorithm backwards. Note that the decoding is done backwards as each of the altered vectors is encoded with respect to some neighbor that might have changed later on. Since the decoding is operated in a reverse order, this guarantees that each vector is recovered with respect to the exact same vector in which it was encoded. ∎

Theorem 10. *Algorithm 1 complexity is $O(M\cdot(B_{\log(M)}(e))^2\cdot B_\ell(t-1)\cdot L)$ symbol operations. Furthermore, the complexity of the decoding process is $O(M\cdot L)$ symbol operations.*

Proof: For the Encoding, we first note that each vector $\boldsymbol{u}_i$ is modified at most once. This is due to the fact. that after the modification it satisfies the constraint in respect to any of its neighbors. Hence, its enough to go through each $\boldsymbol{u}_i$ one by one, and check if any fix is needed, in other words, check whether this vector violates the constraint in respect to some other vector $\boldsymbol{u}_j$. This check requires comparison with $B_{\log(M)}(e)$ other vectors. In case the vector has to be modified, compressing it requires $O(L)$ symbol operations. Lastly, finding $w_\ell(S(e,i),t)$ can be done with a brute force search that is guaranteed to end after at most $B_{\log(M)}(e)\cdot B_\ell(t-1)$ iterations, each requires $B_{\log(M)}(e)$ comparisons. Therefore, each vector requires at most

$$O(B_{\log(M)}(e)\cdot L + L + (B_{\log(M)}(e))^2\cdot B_\ell(t-1)\cdot\ell)$$

which are also

$$O((B_{\log(M)}(e))^2 \cdot B_\ell(t-1) \cdot L)$$

symbol operations. Over all M strands, the overall Complexity of the encoding process is

$$O(M \cdot (B_{\log(M)}(e))^2 \cdot B_\ell(t-1) \cdot L).$$

For the decoding, we need to traverse the chain starting at $\boldsymbol{u}_{M-1}$. this requires reading if at most M indices, or $M\log(M)$ symbol operations. Next, going backwards in the chain, we need to extract the compressed data. That is, for some $\boldsymbol{u}_i$ that compressed in respect to $\boldsymbol{u}_j$, reading $\Delta_1(\mathsf{ind}_i, \mathsf{ind}_j), \Delta_2(\boldsymbol{u}_i, \boldsymbol{u}_j)$, followed by reading $\boldsymbol{u}_j$ at flipping at most t of it's bits. Overall each vector requires

$$e\lceil \log(\log(M)\cdot(q-1))\rceil + (t-1)\lceil \log(L_M\cdot(q-1))\rceil + L + t$$

or, $O(L)$ symbol operations for extraction. Therefore, the overall complexity of the decoding process is $O(M \cdot L)$. ∎

Next we discuss the function $w_\ell(S,t)$. This function takes a set of vectors S as input, and outputs a vector $\boldsymbol{w} \in [q]^\ell$ such that for all $\boldsymbol{v} \in S_{[\log(M),\ell]} \triangleq \{\boldsymbol{v}_{[\log(M),\ell]} | \boldsymbol{v} \in S\}$, it holds that $\mathrm{d_H}(\boldsymbol{w},\boldsymbol{v}) \geqslant t$. The length ℓ of the vector $\boldsymbol{w}$ is determined by the any value of ℓ for which

$$q^\ell > |S_{[\log(M),\ell]}| \cdot B_\ell(t-1).$$

That is, a length that allows us to choose a vector that does not fall into any of the radius-$(t-1)$ balls of the vectors in the set $S_{[\log(M),\ell]}$. The size of $S_{[\log(M),\ell]}$ is at most $B_{\log(M)}(e)-1$, and we denote by $\ell(e,t,M)$ the smallest value of ℓ such that $q^\ell > (B_{\log(M)}(e)-1)\cdot B_\ell(t-1)$.

We first show how such vector w can be constructed efficiently (Also see Figure 3). The presented method is not optimal in the output's length. However, it does improves the encoding complexity presented in Theorem 10.

Lemma 11. *For all e,t,M, a vector $w_\ell(S,t)$ can be constructed for $\ell = t\cdot(\log(|S|)+1)$. The complexity of the construction process is $O(t\cdot|S|\cdot\log(|S|))$ symbol operations.*

Proof: Denote the required output by w. In order to construct w as required, we split each $v \in S$ into t windows, each of length $\log(S)+1$. Note that for each such window we can choose $q^{\log(S)+1} = q\cdot|S|$. On the other hand, for the i-th window, we have at most $|S|$ different values. That is, we can find a value that does not appear in this window, and hence it has Hamming distance at least 1 from any other value in this window. Concatenating the values of the t different windows we get a vector w that has distance at least t from each of the vectors in $S_{[\log(M),\ell]}$.

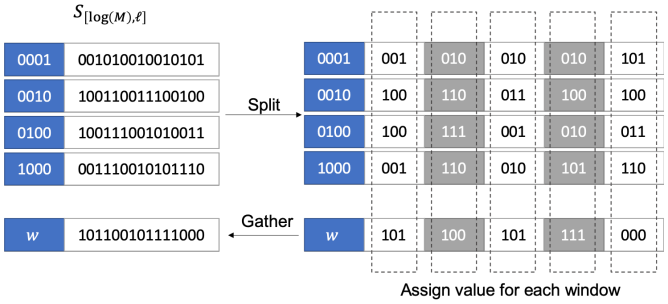

Fig. 3. Construction of $w_\ell(S(e,0),t)$ for $M=16$, $\ell=15$, $e=1$, and $t=5$.

For the complexity, its clear that each window can be handled independently. for each of those windows we can have a histogram of the values appear in that window. it's size would be $q^{|S|}$, and it will take $|S|$ iterations to fill it, each requires $\log(|S|)+1$ symbol operations to find the right entry in the histogram. Next we search for the first available entry in the histogram (value 0) and translate it's index back to a vector over $[q]$. After $|S|$ failures its guaranteed that the next guess would suffice. Therefore, at most $|S|+1$ guesses are needed. We get an overall of

$$|S|\cdot(\log(S)+1) + (|S|+1) + (\log(S)+1) = O(|S|\cdot\log(S))$$

symbol operations. And for all windows together,

$$O(t\cdot|S|\cdot\log(|S|)).$$

∎

From Lemma 11 we derive a construction for $w_\ell(S,t)$ of length $t\cdot(\log(B_{\log(M)}(e))+1)$. The following upper bound $B_{\log(M)}(e) \leqslant (\log(M)\cdot(q-1))^e$ also suggests that

$$\ell(e,t,M) \leqslant t\cdot(e\cdot\log\log(M) + e\cdot\log(q-1)+1).$$

The next lemma provides a better upper bound on the value of $\ell(e,t,M)$.

Lemma 12. *For all e,t,M it holds that*

$$\ell(e,t,M) \leqslant 3t + 2e\cdot\log\log(M) + 2e\cdot\log(q-1).$$

Proof: We start by denoting the size of $S_{[\log(M),\ell]}$ by N. We show that for $w \geqslant 3t+2\log(N)$ the inequality mentioned above is satisfied, hence

$$q^w > N\cdot B_w(t-1),$$

and therefore for $|S_{[\log(M),w]}| = B_{\log(M)}(e)-1$, using the inequality $B_{\log(M)}(e) \leqslant (\log(M)\cdot(q-1))^e$, we achieve the desired result:

$$\begin{aligned}\ell(e,t,M) &\leqslant 3t + 2\log(N)\\ &< 3t + 2e\cdot\log\log(M) + 2e\cdot\log(q-1).\end{aligned}$$

Let $w \geqslant 3t+2\log(N) \geqslant 2t+\log(N)+2\sqrt{t\cdot\log(N)}$. Its easy to verify that the following inequality holds

$$w^2 - (4t+2\log(N))\cdot w + (\log(N))^2 + 4t^2 > 0.$$

Rearranging this inequality we get

$$1 - \frac{2\log(N)}{w} + \left(\frac{\log(N)}{w}\right)^2 > 4\cdot\frac{t\cdot(w-t)}{w^2}.$$

Also, observing that

$$1-\frac{2\log(N)}{w}+\left(\frac{\log(N)}{w}\right)^2=\left(1-\frac{\log(N)}{w}\right)^2,$$

we get,

$$1-\frac{\log(N)}{w}>2\sqrt{\frac{t}{w}\cdot\left(1-\frac{t}{w}\right)}>2\sqrt{\frac{t-1}{w}\cdot\left(1-\frac{t-1}{w}\right)}.$$

Now, we observe that the entropy function $\mathcal{H}_q(x)$ always reaches its maximum for $x=1-1/q$ with value of 1. In addition, it is creasing over $[0,1-1/q]$, therefore for any $p\in[0,1/2]$ it holds that $\mathcal{H}_q(p)\leqslant\mathcal{H}_2(p)\leqslant 2\sqrt{p(1-p)}$. We can deduce the following:

$$1-\frac{\log(N)}{w}>\mathcal{H}_q\left(\frac{t-1}{w}\right).$$

Rearranging again, we get

$$w>\log(N)+w\cdot\mathcal{H}_q\left(\frac{t-1}{w}\right)=\log\left(N\cdot q^{w\cdot\mathcal{H}_q\left(\frac{t-1}{w}\right)}\right),$$

and lastly

$$q^w>N\cdot B_w(t-1),$$

using the inequality $B_w(t-1)\leqslant q^{w\cdot\mathcal{H}_q\left(\frac{t-1}{w}\right)}$. ∎

Corollary 13. *For all*

$$\begin{aligned}t&\leqslant 1+\frac{L_M-2\log(M)-3e\log\log(M)-3e\log(q-1)-4}{\log(L_M)+\log(q-1)+3}\\&=1+\frac{L(1-2\beta)-3e\log(\beta)-3e\log(L)-3e\log(q-1)-4}{\log(L)+\log(1-\beta)+\log(q-1)+3},\end{aligned}$$

there exists an explicit construction of an (e,t)-CCC using Algorithm 1 which uses a single bit of redundancy.

Proof: From Lemma 12 we can use $w_\ell(S,t)$ in Algorithm 1 with $\ell=3t+2e\cdot\log\log(M)+2e\cdot\log(q-1)$. In addition, from Theorem 9 the value of t should satisfy

$$\ell+\log(B_{\log(M)}(e)\cdot B_{L_M}(t-1))+1\leqslant L-2\log(M).$$

Thus, it is enough to show that

$$\begin{aligned}&\ell+e\log(\log(M)\cdot(q-1))\\&\quad+(t-1)\log(L_M\cdot(q-1))+1\leqslant L-2\log(M),\end{aligned}$$

while using $B_n(r)\leqslant(n\cdot(q-1))^r$. Therefore, t should satisfy the following:

$$\begin{aligned}t\cdot(3+\log(L_M)+\log(q-1))&\leqslant L-2\log(M)\\&\quad-3e\log\log(M)-(3e-1)\log(q-1)+\log(L_M)-1.\end{aligned}$$

Rearranging this inequality yields the following bound on the value t that is valid for construction presented in Algorithm 1:

$$t\leqslant 1+\frac{L-2\log(M)-3e\log\log(M)-3e\log(q-1)-4}{\log(L_M)+\log(q-1)+3}.$$

∎

According to Section IV, $r_{M,L}(e,t)=1$ when t is approximately $L_M\cdot\mathcal{H}_q^{-1}\left(\frac{1-2\beta}{1-\beta}\right)$. However, this is not achieved by an explicit construction of such codes. Here, we presented an explicit construction for $e=1$ in which the maximum value of t is roughly $\frac{L_M}{\log(L_M)}\frac{1-2\beta}{1-\beta}$.

VI. Edit Distance Extensions

TBW

VII. Conclusion

In this paper we presented a new family of codes, called clustering-correcting codes. These codes are beneficial in DNA-based storage systems in order to cluster the strands in the correct groups. We showed upper and lower bound on these codes as well as an explicit construction which uses a single bit of redundancy. We then discussed extensions of those results from the Hamming distance metric into the Edit distance metric.

References

[1] M. Blawat, K. Gaedke, I. Hütter, X.-M. Chen, B. Turczyk, S. Inverso, B.W. Pruitt, and G.M. Church, "Forward error correction for DNA data storage," *Int. Conf. on Computational Science*, vol. 80, pp. 1011–1022, 2016.

[2] J. Bornholt, R. Lopez, D. M. Carmean, L. Ceze, G. Seelig, and K. Strauss, "A DNA-based archival storage system," *Proc. of the Twenty-First Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 637–649, Atlanta, GA, Apr. 2016.

[3] Y.M. Chee, H.M. Kiah, and H. Wei, "Efficient and explicit balanced primer codes," https://arxiv.org/abs/1901.01023, 2019.

[4] G. M. Church, Y. Gao, and S. Kosuri, "Next-generation digital information storage in DNA," *Science*, vol. 337, no. 6102, pp. 1628–1628, Sep. 2012.

[5] Y. Erlich and D. Zielinski, "DNA fountain enables a robust and efficient storage architecture," *Science*, vol. 355, no. 6328, pp. 950–954, 2017.

[6] R. Feynman, "There's plenty of room at the bottom," *Engineering and Science, California Institute of Technology*, vol. 23, pp. 22–36, 1960.

[7] D. G. Gibson, J. I. Glass, C. Lartigue, V. N. Noskov, R.-Y. Chuang, M. A. Algire, G. A. Benders, M. G. Montague, L. Ma, M. M. Moodie, C. Merryman, S. Vashee, R. Krishnakumar, N. Assad-Garcia, C. Andrews-Pfannkoch, E. A. Denisova, L. Young, Z.-Q. Qi, T. H. Segall-Shapiro, C. H. Calvey, P. P. Parmar, C. A. Hutchison, H. O. Smith, and J. C. Venter., "Creation of a bacterial cell controlled by a chemically synthesized genome," *Science*, vol. 329, no. 5987, pp. 52–56, Jul. 2010.

[8] N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. M. LeProust, B. Sipos, and E. Birney, "Towards practical, high-capacity, low-maintenance information storage in synthesized DNA," *Nature*, vol. 494, no. 7435, pp. 77–80, 2013.

[9] R. Heckel, G. Mikutis, and R.N. Grass, "A characterization of the DNA data storage channel," arxiv.org/pdf/1803.03322.pdf, 2018.

[10] S. Jain, F. Farnoud, M. Schwartz, and J. Bruck, "Duplication-correcting codes for data storage in the DNA of living organisms," *IEEE Trans. on Inf. Theory*, vol. 63, no. 8, pp. 4996–5010, Mar. 2017.

[11] A. Lenz, P. H. Siegel, A. Wachter-Zeh, and E. Yaakobi, "Coding over sets for DNA storage," submitted to *IEEE Trans. on Inf. Theory*, Available online https://arxiv.org/abs/1812.02936, 2018.

[12] M. Levy and E. Yaakobi, "Mutually uncorrelated codes for DNA storage," to appear *IEEE Trans. Inform. Theory*.

[13] L. Organick, S.D. Ang, Y.-J. Chen, R. Lopez, S. Yekhanin, K. Makarychev, M.Z. Racz, G. Kamath, P. Gopalan, B. Nguyen, C. Takahashi, S. Newman, H.-Y. Parker, C. Rashtchian, K. Stewart, G. Gupta, R. Carlson, J. Mulligan, D. Carmean, G. Seelig, L. Ceze, and K. Strauss, "Scaling up DNA data storage and random access retrieval," *bioRxiv*, Mar. 2017.

[14] C. Rashtchian, K. Makarychev, M. Racz, S. Ang, D. Jevdjic, S. Yekhanin, L. Ceze, and K. Strauss, "Clustering billions of reads for DNA data storage," *NIPS*, 2017.

[15] R.M. Roth, *Introduction to Coding Theory*, Cambridge University Press, 2005.

[16] S. H. T. Yazdi, H. M. Kiah, E. Garcia-Ruiz, J. Ma, H. Zhao, and O. Milenkovic, "DNA-based storage: Trends and methods," *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, vol 1, no. 3, pp. 230–248, 2015.

[17] S. M. H. T. Yazdi, H. M. Kiah, and O. Milenkovic, "Weakly mutually uncorrelated codes," *IEEE Int. Symp. Inf. Theory*, pp. 2649–2653, Barcelona, Spain, Jul. 2016.

[18] S. M. H. T. Yazdi, Y. Yuan, J. Ma, H. Zhao, and O. Milenkovic, "A rewritable, random-access DNA-based storage system," *Nature Scientific Reports*, vol. 5, no. 14138, Aug. 2015.

[19] R.N. Grass, R. Heckel, M. Puddu, D. Paunescu, and W.J. Stark, "Robust chemical preservation of digital information on DNA in silica with error-correcting codes," *Angewandte Chemie International Edition*, vol. 54, no. 8, pp. 2552–2555, Feb. 2015.