Linear Programming Bounds for Robust Locally Repairable Storage Codes

M. Ali Tebbi[†], Terence H. Chan[†], Chi Wan Sung[‡]
[†]Institute for Telecommunications Research, University of South Australia
Email: {ali.tebbi, terence.chan}@unisa.edu.au
[‡]Department of Electronic Engineering, City University of Hong Kong
Email: albert.sung@cityu.edu.hk

***Abstract*—Locally repairable codes are used in distributed storage networks to minimise the number of survived nodes required to repair a failed node. However, the robustness of these codes is a main concern since locally repair procedure may fail when there are multiple node failures. This paper proposes a new class of robust locally repairable codes which guarantees that a failed node can be repaired locally even when there are multiple node failures. Upper bound on the size of robust locally repairable codes using linear programming tools are obtained and examples of robust locally repairable codes attaining these bounds are constructed.**

I. Introduction

Data storage networks (DSN) use multiple storage nodes to store a massive amount of information. The most important challenge in designing a storage network is to optimise the efficiency of the storage network in terms of the storage, repair, retrieve, and update costs. To ensure that data is stored reliably, redundancy is introduced such that any data loss in a failed node can be recovered from other surviving nodes. For example, suppose k data blocks are stored in a DSN with n storage nodes using a maximum distance separable (MDS) erasure code. The encoded data is then resilient against up to any $d = n - k$ simultaneous failures. Such a storage code is very efficient in minimising the amount of total data stored in the network. There is however a catch. Using such a code, the total amount of information being retrieved to repair a failure (i.e., the repair bandwidth) can be huge. Suppose a single storage node fails. To repair the failed node, it may require to retrieve all of the information content stored in any k surviving nodes in order to regenerate the lost content. This can be very inefficient in terms of the repair bandwidth.

Dimakis *et al.* in [1] introduced the regenerating codes and showed that the repair bandwidth can be greatly reduced by increasing the number of surviving nodes from which the failed node retrieves data. A tradeoff between the cost for storage and repair was also obtained. Although the regenerating codes can reduce the repair bandwidth, it also requires a high I/O overhead in DSNs, which is proportional to the number of nodes involved in repairing a failed node.

Locally repairable codes were later proposed in [2] to minimise the number of surviving nodes required to repair a failed node. In [3], an upper bound for the minimum distance d of these codes has been derived. In [4], a linear programming bound for locally repairable codes was obtained.

This work was partially supported by a grant from Australian Research Council (DP1094571) and the University Grants Committee of the Hong Kong Special Administrative Region, China (Project No. AoE/E-02/08).

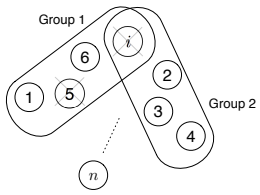

Fig. 1. A robust locally repairable DSN with multiple repairing groups

Since the regenerating codes are aimed at minimising the amount of downloaded data (i.e. repair bandwidth) to repair a failed node, locally repairable codes minimise the number of helper nodes to repair a failure. Several optimal code constructions have been presented in [5] which combine the features for both coding schemes known as local regenerating codes.

One significant issue of locally repairable codes is their robustness. Consider the DSN model with n storage nodes shown in Fig. 1. If node i fails, a subset of r nodes in Group 1 can repair the failed node. Now, if any of the nodes in Group 1 is not available (for example, due to that node is heavily loaded), then Group 1 cannot be used to repair the failed node anymore. To avoid this problem, it is essential to have multiple repairing groups which can be used for repairing if additional nodes fail or become unavailable. The concept of locally repairable codes with multiple repair groups (or alternatives) was studied in [6] and codes with multiple alternatives were also constructed. Locally repairable code construction with multiple disjoint repairing groups was investigated in [7]. In [8], locally repairable codes with multiple erasure tolerance have been constructed using combinatorial methods. A lower bound on the code size was derived and some code examples provided which are optimal with respect to this bound. A class of locally repairable codes with the capability of repairing a set of failed nodes by connecting to a set of survived nodes are studied in [9]. Bounds on code dimension and also families of explicit codes were provided.

Our contributions – In this paper, we consider "robust locally repairable codes" which guarantee that a node failure can be repaired locally even when a group of nodes are unavailable during the repair procedure. Following our work in [4], we have derived a linear programming bound for robust

978-1-4799-5999-0/14/$31.00 ©2014 IEEE

locally repairable codes. Finally, we will present examples of robust locally repairable codes and will show that these codes are optimal (that they satisfy our linear programming bound).

Notation: Let $\mathcal{N} = \{1, 2, 3, \ldots, n\}$ and $2^{\mathcal{N}}$ be the set of all possible subsets of $\mathcal{N}$. In this paper we represent a subset ν of $\mathcal{N}$ as a binary vector $\mathbf{w} = [w_1, w_2, w_3, \ldots, w_n]$ such that $w_i = 1$ if and only if $i \in \mathcal{N}$. In our convention, ν and $\mathbf{w}$ can be used interchangeably. Similarly, we use zero vector $\mathbf{0}$ to present an empty set $\varnothing$. Also, we will make no distinction between an index i and $\{i\}$ for a single element set.

II. Preliminaries and Definitions

A. Linear Storage Codes

Let $\mathbb{F}_q^n = \{[z_1, z_2, z_3, \ldots, z_n] : z_i \in \mathbb{F}_q \text{ for all } i \in \mathcal{N}\}$ be the vector space of all n-tuples over the field $\mathbb{F}_q$. A linear storage code $\mathcal{C}$ generated by a $k \times n$ full rank (i.e. rank k) generator matrix G is a subspace of $\mathbb{F}_q^n$ over the field $\mathbb{F}_q$. Here, k is the dimension of the subspace. The linear code $\mathcal{C} = \left\{\mathbf{u}G : \mathbf{u} \in \mathbb{F}_q^k\right\}$ is defined as the set of all linear combinations (i.e. spans) of the row vectors of the generator matrix G.

To store a data block of size $\mathcal{M}$ in a storage network using a linear code $\mathcal{C}$, the original data is first divided into k data blocks of size $\mathcal{M}/k$. We may assume without loss of generality that the original data is a matrix with $\mathcal{M}/k$ rows and k columns. Then each block (or row) will be encoded using the same code. In this paper, we may assume without loss of generality that there is only one row (and thus $\mathcal{M} = k$). Let $\mathbf{u} = \mathcal{M}$ be the information to be stored. Then the storage node i will store the symbol $\mathbf{u}\mathbf{g}_i$ where $\mathbf{g}_i$ is the i^{th} column of the generator matrix G. Notice that if X_i is the content that the storage node i stores, then $(X_1, \ldots, X_n)$ is a codeword in $\mathcal{C}$. To retrieve the stored information, it is sufficient to retrieve the information content of k nodes (which are indexed by $\mathcal{I}$) such that the columns $\mathbf{g}_i$ for $i \in \mathcal{I}$ are all independent. Specifically,

$$\mathbf{u} = \mathbf{c}_{\mathcal{I}} G_{\mathcal{I}}^{-1} \tag{1}$$

where $\mathbf{c}_{\mathcal{I}}$ is the row vector whose i^{th} entry equals to $\mathbf{u}\mathbf{g}_i$ and $G_{\mathcal{I}}$ is a submatrix of the generator matrix G formed by the columns $\mathbf{g}_i$ for $i \in \mathcal{I}$.

Definition 1 (Support)**.** *For any vector $\mathbf{c} = [c_1, \ldots, c_n] \in \mathbb{F}_q^n$, its support $\lambda(\mathbf{c})$ is defined as a subset of $\mathcal{N} = \{1, \ldots, n\}$ such that $i \in \lambda(\mathbf{c})$ if and only if $c_i \neq 0$.*

Remark 1. *For a linear code $\mathcal{C}$ over $\mathbb{F}_q$, if $\mathbf{c} \in \mathcal{C}$, then $a\mathbf{c} \in \mathcal{C}$ for all $a \in \mathbb{F}_q$ and $a \neq 0$. Therefore, except for the zero codeword, there exists at least $q - 1$ codewords which have the same support.*

Definition 2 (Support Enumerator)**.** *The support enumerator $\Lambda_{\mathcal{C}}(\mathbf{w})$ of the code $\mathcal{C}$ is the number of codewords with the same support $\mathbf{w}$, i.e.,*

$$\Lambda_{\mathcal{C}}(\mathbf{w}) \triangleq \left|\{\mathbf{c} \in \mathcal{C} : \lambda(\mathbf{c}) = \mathbf{w}\}\right|, \qquad \forall \mathbf{w} \subseteq \mathcal{N}.$$

Consider any linear code $\mathcal{C}$ specified by a $(n-k)\times n$ parity check matrix H. Then, the dual code $\mathcal{C}^{\perp}$ is also a linear code which is generated by H. It is well known that

$$\mathcal{C}^{\perp} = \left\{\mathbf{h} \in \mathbb{F}_q^n : \mathbf{h}\mathbf{c}^{\top} = 0 \text{ for all } \mathbf{c} \in \mathcal{C}\right\}.$$

Proposition 1 (MacWilliams identity [10])**.** *Let $\mathcal{C}$ be an (n, k) linear code and $\mathcal{C}^{\perp}$ be its dual. Then for any $\mathbf{w} = [w_1, \ldots, w_n] \subseteq \mathcal{N}$,*

$$\Lambda_{\mathcal{C}^{\perp}}(\mathbf{w}) = \frac{1}{|\mathcal{C}|} \sum_{\mathbf{s}=[s_1,\ldots,s_n]\in 2^{\mathcal{N}}} \Lambda_{\mathcal{C}}(\mathbf{s}) \prod_{j=1}^{n} \kappa_q(s_j, w_j) \geq 0 \tag{2}$$

where

$$\kappa(s, w) = \begin{cases} 1 & \text{if } w = 0 \\ q - 1 & \text{if } s = 0 \text{ and } w = 1 \\ -1 & \text{otherwise.} \end{cases}$$

III. Local Repairability of Linear Storage Codes

A linear storage code has locality r, if for any failed node i, there exists a group of survived nodes of size at most r such that the failed node can be repaired by this group of nodes [3]. To be precise, consider a linear code $\mathcal{C}$ and its dual code $\mathcal{C}^{\perp}$. Let $\mathbf{h} = [h_1, \ldots, h_n]$ and $\mathbf{c} = [c_1, \ldots, c_n]$ be the codewords in $\mathcal{C}^{\perp}$ and $\mathcal{C}$, respectively. Then

$$\mathbf{h}\mathbf{c}^{\top} = 0, \quad \forall \mathbf{h} \in \mathcal{C}^{\perp}, \, \mathbf{c} \in \mathcal{C}. \tag{3}$$

As we mentioned earlier, $(X_1, \ldots, X_n) \in \mathcal{C}$. Then,

$$0 = \sum_{i\in\mathcal{N}} h_i X_i = \sum_{j\in\lambda(\mathbf{h})} h_j X_j \tag{4}$$

where $\lambda(\mathbf{h})$ is the support of the codeword in dual code. Consequently, if the node i fails and $i \in \lambda(\mathbf{h})$, the content of the failed node can be recovered from the set of surviving nodes indexed by $\lambda(\mathbf{h}) \setminus i$. In particular,

$$X_i = -h_i^{-1} \sum_{j\in\lambda(h)\setminus i} X_j h_j.$$

Here, the set $\lambda(\mathbf{h}) \setminus i$ is a repair group of node i.

Definition 3 (Locally repairable code [2])**.** *An (r, β) locally repairable code $\mathcal{C}$ is a linear code which satisfies*

1) ***Local Recovery (LR).*** *for any $i \in \mathcal{N}$, there exists $\mathbf{h} \in \mathcal{C}^{\perp}$ such that $i \in \lambda(\mathbf{h})$ and $|\lambda(\mathbf{h})| - 1 \leq r$.*
2) ***Global Recovery (GR).*** *$\Lambda_{\mathcal{C}}(\mathbf{w}) = 0$, for all $\mathbf{w} \subseteq \mathcal{N}$ such that $1 \leq |\mathbf{w}| \leq \beta$.*

For a (r, β) locally repairable code, the local recovery criterion implies that for any failed node i, it can always be repaired by retrieving the contents stored in at most r nodes. On the other hand, the global recovery criterion guarantees that the storage network can be regenerated as long as there are at most β simultaneous node failures.

Remark: The LR criterion requires the existence of a set of r surviving nodes which can be used to repair a failed node. However, under the criteria, there is no guarantee that a failed node can still be efficiently repaired[1] if a few additional nodes also fail (or become unavailable).

Therefore, it is essential to have alternative repair groups in case of multiple node failures.

[1]Strictly speaking, by the GR criterion, the failed node can still be repaired if there are no more than β node failures. However, the global recovery cannot guarantee that each failed node can be repaired efficiently. In other words, one may need a much larger set of surviving nodes to repair one failed node.

Definition 4 (Robust locally repairable code)**.** *An $(r, \beta, \Gamma, \zeta)$ robust locally repairable code is a linear code satisfying the following criteria:*

1) ***Robust Local Recovery (RLR).*** *for any $i \in \mathcal{N}$ and $\gamma \subset \mathcal{N} \backslash i$ such that $|\gamma| = \Gamma$, there exists $\mathbf{h}_1, \ldots, \mathbf{h}_\zeta \in \mathcal{C}^\perp$ such that for all $j = 1, \ldots, \zeta$,*
   a) *$i \in \lambda(\mathbf{h}_j)$, $\gamma \cap \lambda(\mathbf{h}_j) = \emptyset$, and $|\lambda(\mathbf{h}_j)| - 1 \leq r$.*
   b) *$\lambda(\mathbf{h}_j) \neq \lambda(\mathbf{h}_k)$ for $k \neq j$.*
2) ***Global Recovery (GR).*** *$\Lambda_{\mathcal{C}}(\mathbf{w}) = 0$, for all $\mathbf{w} \subseteq \mathcal{N}$ such that $1 \leq |\mathbf{w}| \leq \beta$.*

The RLR criterion guarantees that a failed node can be repaired locally from any one of the ζ groups of surviving nodes of size r even if Γ extra nodes fail. In the special case when $\Gamma = 0$, then the robust locally repairable codes are reduced to locally repairable codes with multiple repair alternatives as in [6]. When $\zeta = 1$, then it further reduces to the traditional locally repairable codes. The GR criterion is the same as that for locally repairable codes.

IV. Linear Programming Bounds

One of the most fundamental problems in storage network design is to optimise the tradeoff between the costs for storage and repair. For robust locally repairable code, the storage cost (per information bit per node) is given by $1/\log|\mathcal{C}|$, while the repair cost is given by $r/\log|\mathcal{C}|$. Therefore, to minimise the costs is equivalent to maximise the codebook size $|\mathcal{C}|$.

In this section, we will obtain an upper bound for the maximal codebook size, subject to robust local recovery and global recovery criteria. It extends our previous work in [4] in which bounds for locally repairable codes (corresponding to the case when $\Gamma = 0$ and $\zeta = 1$) were developed. The proof technique used in this paper is also similar to that used in [4].

Theorem 1. *Consider any $(r, \beta, \Gamma, \zeta)$ robust locally repairable code $\mathcal{C}$. Then, $|\mathcal{C}|$ is upper bounded by the optimal value of the following optimisation problem.*

$$
\begin{aligned}
&\textbf{\textit{maximize}} \quad \textstyle\sum_{\mathbf{w}\subseteq\mathcal{N}} A_{\mathbf{w}} \\
&\textbf{\textit{subject to}} \\
&\quad A_{\mathbf{w}} \geq 0 && \forall \mathbf{w} \subseteq \mathcal{N} \\
&\quad B_{\mathbf{w}} = \tfrac{\sum_{\mathbf{s}\subseteq\mathcal{N}} A_{\mathbf{s}} \prod_{j=1}^{n} \kappa(s_j, w_j)}{\sum_{\mathbf{s}\subseteq\mathcal{N}} A_{\mathbf{s}}} && \forall \mathbf{w} \subseteq \mathcal{N} \\
&\quad B_{\mathbf{w}} \geq 0 && \forall \mathbf{w} \subseteq \mathcal{N} \\
&\quad A_{\mathbf{w}} = 0 && 1 \leq |\mathbf{w}| \leq \beta \\
&\quad A_{\varnothing} = 1 \\
&\quad \sum_{\mathbf{s}\in\Omega_i : \gamma\cap\mathbf{s}=\emptyset} B_{\mathbf{s}} \geq \zeta(q-1) && \forall i \in \mathcal{N}, \gamma \in \Delta_i
\end{aligned}
\tag{5}
$$

where Ω_i is the collection of all subsets of $\mathcal{N}$ that contains i and of size at most $r+1$ and Δ_i is the collection of all subsets of $\mathcal{N} \setminus i$ of size at most Γ.

Proof: Let $\mathcal{C}$ be a $(r, \beta, \Gamma, \zeta)$ robust locally repairable code. Then, we define

$$
\begin{aligned}
A_{\mathbf{w}} &\triangleq \Lambda_{\mathcal{C}}(\mathbf{w}), \quad \mathbf{w} \subseteq \mathcal{N} \\
B_{\mathbf{w}} &\triangleq \Lambda_{\mathcal{C}^\perp}(\mathbf{w}), \quad \mathbf{w} \subseteq \mathcal{N}.
\end{aligned}
$$

The objective function of the optimisation problem in (5) is the sum of the number of all codewords with support $\mathbf{w}$, which is clearly equal to the size of the code $\mathcal{C}$. Since $A_{\mathbf{w}}$ and $B_{\mathbf{w}}$ are enumerator functions of the code $\mathcal{C}$ and $\mathcal{C}^\perp$ respectively, $A_{\mathbf{w}}, B_{\mathbf{w}} \geq 0$ for all $\mathbf{w} \subseteq \mathcal{N}$. The constraint $A_{\varnothing} = 1$ follows from the fact that the zero codeword is contained in any code. The second constraint follows from (2) by substituting $|\mathcal{C}|$ and $\Lambda_{\mathcal{C}}(\mathbf{s})$ with $\sum_{\mathbf{s}\subseteq\mathcal{N}} A_{\mathbf{s}}$ and $A_{\mathbf{s}}$, respectively. The forth constraint followed directly from GR criterion. Finally, the last constraint follows from the RLR criterion and the fact that $\mathcal{C}$ is a linear code. For any nonzero $\mathbf{h}_j \in \mathcal{C}^\perp$, then $a\mathbf{h}_j \in \mathcal{C}^\perp$ for all nonzero $a \in \mathbb{F}_q$. Let $\mathbf{w}_j = \lambda(\mathbf{h}_j)$ for $j = 1, \ldots, \zeta$. Then, $B_{\mathbf{w}_j} \geq q - 1$. Consequently,

$$
\sum_{\mathbf{s}\in\Omega_i : \gamma\cap\mathbf{s}=\emptyset} B_{\mathbf{w}} \geq \zeta(q-1).
$$

Therefore, $A_{\mathbf{w}}, B_{\mathbf{w}} : \mathbf{w} \subseteq \mathcal{N}$ satisfies the constraints in the maximisation problem in (5) and the theorem follows. ∎

The maximisation problem in (5) is not a linear programming problem, but it can be converted to one as follows:

$$
\begin{aligned}
&\textbf{maximize} \quad \textstyle\sum_{\mathbf{w}\subseteq\mathcal{N}} A_{\mathbf{w}} \\
&\textbf{subject to} \\
&\quad A_{\mathbf{w}} \geq 0 && \forall \mathbf{w} \subseteq \mathcal{N} \\
&\quad C_{\mathbf{w}} = \textstyle\sum_{\mathbf{s}\subseteq\mathcal{N}} A_{\mathbf{s}} \prod_{j=1}^{n} \kappa(s_j, w_j) && \forall \mathbf{w} \subseteq \mathcal{N} \\
&\quad C_{\mathbf{w}} \geq 0 && \forall \mathbf{w} \subseteq \mathcal{N} \\
&\quad A_{\mathbf{w}} = 0 && 1 \leq |\mathbf{w}| \leq \beta \\
&\quad A_{\varnothing} = 1 \\
&\quad \sum_{\mathbf{s}\in\Omega_i : \gamma\cap\mathbf{s}=\emptyset} C_{\mathbf{s}} \geq \zeta(q-1) \sum_{\mathbf{w}\subseteq\mathcal{N}} A_{\mathbf{w}} && \forall i \in \mathcal{N}, \gamma \in \Delta_i.
\end{aligned}
\tag{6}
$$

The problem can be further simplified to

$$
\begin{aligned}
&\textbf{maximize} \quad \textstyle\sum_{\mathbf{w}\subseteq\mathcal{N}} A_{\mathbf{w}} \\
&\textbf{subject to} \\
&\quad A_{\mathbf{w}} \geq 0 && \forall \mathbf{w} \subseteq \mathcal{N} \\
&\quad \textstyle\sum_{\mathbf{s}\subseteq\mathcal{N}} A_{\mathbf{s}} \prod_{j=1}^{n} \kappa(s_j, w_j) \geq 0 && \forall \mathbf{w} \subseteq \mathcal{N} \\
&\quad A_{\mathbf{w}} = 0 && 1 \leq |\mathbf{w}| \leq \beta \\
&\quad A_{\varnothing} = 1 \\
&\quad \sum_{\mathbf{w}\in\Omega_i : \gamma\cap\mathbf{w}=\emptyset} \left(\sum_{\mathbf{s}\subseteq\mathcal{N}} A_{\mathbf{s}} \prod_{j=1}^{n} \kappa(s_j, w_j) \right) \\
&\qquad \geq \zeta(q-1) \textstyle\sum_{\mathbf{w}\subseteq\mathcal{N}} A_{\mathbf{w}} && \forall i \in \mathcal{N}, \gamma \in \Delta_i.
\end{aligned}
\tag{7}
$$

The complexity of the linear programming problem in (7) will increase exponentially with the number of storage nodes n. In the following, we will reduce the complexity of the Linear Programming problem in (7) by exploiting the symmetries in the problem [11].

To this end, suppose a $(r, \beta, \Gamma, \zeta)$ linear storage code $\mathcal{C}$ is generated by the matrix G with columns $\mathbf{g}_i,\ i = 1, 2, \ldots, n$. Let $S_{\mathcal{N}}$ be the symmetric group on $\mathcal{N}$ whose elements are all the permutations of the elements in $\mathcal{N}$ which are treated as bijective functions from the set of symbols to itself. Clearly, $|S_{\mathcal{N}}| = n!$. Let σ be a permutation on $\mathcal{N}$ (i.e., $\sigma \in S_{\mathcal{N}}$). Together with the code $\mathcal{C}$, each permutation σ defines a new code $\mathcal{C}^\sigma$ specified with the generator matrix columns $\mathbf{f}_i,\ i = 1, 2, \ldots, n$, such that for all $i \in \mathcal{N}$, $\mathbf{f}_i = \mathbf{g}_{\sigma(i)}$. Since all the

codewords are the linear combinations of the generator matrix rows and the permutation σ just changes the generator matrix columns position, the permutation cannot affect the minimum distance of the code (i.e., every codeword $\mathbf{c}^\sigma \in \mathcal{C}^\sigma$ is just a permuted version of the corresponding codeword $\mathbf{c} \in \mathcal{C}$). Therefore, the code $\mathcal{C}^\sigma$ is still a $(r, \beta, \Gamma, \zeta)$ linear storage code and we have the following proposition.

Proposition 2. *Suppose $(a_{\mathbf{w}} : \mathbf{w} \subseteq \mathcal{N})$ satisfies the constraint in the optimisation problem* (7)*. For any $\sigma \in S_{\mathcal{N}}$, let*

$$(a^\sigma_{\mathbf{w}} : \mathbf{w} \subseteq \mathcal{N}) = a^\sigma_{\mathbf{w}} = a_{\sigma(\mathbf{w})},$$

where $\sigma(\mathbf{w}) \triangleq \{\sigma(i) : i \in \mathbf{w}\}$. Then $(a^\sigma_{\mathbf{w}} : \mathbf{w} \subseteq \mathcal{N})$ also satisfies the constraint in (7)*.*

Corollary 1. *Let*

$$a^*_{\mathbf{w}} = \frac{1}{|S_{\mathcal{N}}|} \sum_{\sigma \in S_{\mathcal{N}}} a^\sigma_{\mathbf{w}}.$$

Then $(a^*_{\mathbf{w}} : \mathbf{w} \subseteq \mathcal{N})$ satisfies the constraint in (7) *and*

$$\sum_{\mathbf{w} \subseteq \mathcal{N}} a^*_{\mathbf{w}} = \sum_{\mathbf{w} \subseteq \mathcal{N}} a_{\mathbf{w}}.$$

Proof: From Proposition 2, for any feasible solution $(a_{\mathbf{w}} : \mathbf{w} \subseteq \mathcal{N})$ in (7), there exists $|S_{\mathcal{N}}|$ feasible solutions $(a^{\sigma^{(i)}}_{\mathbf{w}} : \mathbf{w} \subseteq \mathcal{N}$, and $\sigma^{(i)} \in S_{\mathcal{N}})$. Since (7) is a linear programming problem, $a^*_{\mathbf{w}}$ (the average of all feasible solutions) is also a feasible solution. The result then follows. ∎

Proposition 2 can be used to reduce the complexity in solving the optimisation problem (7).

Proposition 3. *If $|\mathbf{w}| = |\mathbf{s}|$, then $a^*_{\mathbf{w}} = a^*_{\mathbf{s}}$.*

Proof: Direct verification. ∎

By Proposition 2, it is sufficient to consider only "symmetric" feasible solution $(a^*_{\mathbf{w}} : \mathbf{w} \subseteq \mathcal{N})$. Therefore, we can impose additional constraint

$$A_{\mathbf{w}} = A_{\mathbf{s}}, \quad \forall |\mathbf{w}| = |\mathbf{s}|.$$

to (7) without affecting the bound. These equality constraint will significantly reduce the number of variable in the optimisation problem. Specifically, we have the following theorem.

Theorem 2. *Consider a $(r, \beta, \Gamma, \zeta)$ robust locally repairable code $\mathcal{C}$. Then, $|\mathcal{C}|$ is upper bounded by the optimal value in the following maximisation problem*

$$\begin{aligned}
&\textbf{\textit{maximize}} \quad \textstyle\sum_{t=0}^{n} \binom{n}{t} a_t \\
&\textbf{\textit{subject to}} \\
&a_t \geq 0, \quad \forall t = 0, \ldots, n \\
&b_t = \textstyle\sum_{i=0}^{t} \sum_{j=0}^{n-t} \binom{t}{i} \binom{n-t}{j} a_{i+j} (-1)^i (q-1)^{t-i} \\
&\qquad\qquad \forall t = 0, \ldots, n \\
&b_t \geq 0, \quad \forall t = 0, \ldots, n \\
&\textstyle\sum_{t=1}^{\beta} a_t = 0 \\
&a_0 = 1 \\
&\displaystyle\sum_{t=1}^{r} \binom{n-1-\Gamma}{t} b_{t+1} \geq \zeta (q-1) \sum_{t=0}^{n} \binom{n}{t} a_t.
\end{aligned} \tag{8}$$

Remark: The number of variables and constraint now scales only linearly with n (the number of nodes in the network).

V. Examples

In this section, we present two examples of robust locally repairable codes. We will show that these codes can repair a failed node locally when some of the survived nodes are not available. Using our bound, we prove that these codes are optimal.

Example 1: Consider a binary linear storage code of length $n = 16$ with $k = 9$ as defined by the following parity check equations

$$0 = P_1 + C_1 + C_2 + C_3 \tag{9}$$
$$0 = P_2 + C_4 + C_5 + C_6 \tag{10}$$
$$0 = P_3 + C_7 + C_8 + C_9 \tag{11}$$
$$0 = P_4 + C_1 + C_4 + C_7 \tag{12}$$
$$0 = P_5 + C_2 + C_5 + C_8 \tag{13}$$
$$0 = P_6 + C_3 + C_6 + C_9 \tag{14}$$
$$0 = P_7 + C_1 + C_2 + C_3 + C_4 + C_5 + C_6 + C_7 + C_8 + C_9. \tag{15}$$

Here, $(C_1, \ldots, C_9, P_1, \ldots, P_7)$ correspond to the content stored at the 16 nodes. In particular, we might interpret that $C_1, \ldots, C_9$ are the systematic bits while $P_1, \ldots, P_7$ are the parity check bits. The code can also be represented by Figure 2. In this case, the parity check equations are equivalent to that the sum of rows and the sum of columns are all zero.

According to (9)-(15), every failed node (either systematic or parity) can be recovered by two different repair groups of size $r = 3$. For instance, if C_1 is failed, it can be repaired by repair group $R^1_{C_1} = \{C_2, C_3, P_1\}$ or repair group $R^2_{C_1} = \{C_4, C_7, P_4\}$. Notice that the two repair groups are disjoint. Therefore, even if one of the repair groups is not available, there exists an alternative repair group to locally repair the failed node.

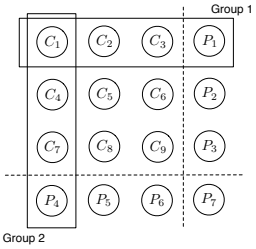

Fig. 2. A (3,3,1,1) robust locally repairable code of length $n = 16$ and dimension $k = 9$.

It can be verified easily that our code has a minimum distance of 4 and it is a $(3, 3, 1, 1)$ robust locally repairable code. Therefore, any failed node can be repaired by at least $\zeta = 1$ repairing group of size $r = 3$ in the presence of any $\Gamma = 1$ extra failure. Also, the original data can be recovered even if there are $\beta = 3$ simultaneous failure. In fact, it is also a $(3, 3, 0, 2)$ robust locally repairable code.

Now, we will use the bound obtained earlier to show that our code is optimal. We have plotted our bound in Figure 3 when $n = 16$ and $\beta = 3$. The horizontal axis is the code locality (i.e., r) and the vertical axis is the dimension of the

code (or $\log|\mathcal{C}|$). From the figure, when $r = 3$, the dimension of a $(3, 3, 1, 1)$ robust locally repairable code is at most 9. And our constructed code has exactly 9 dimension. Therefore, our code meets the bound and is optimal. In fact, our bound also indicates that the dimension of a $(3, 3, 0, 2)$ robust locally repairable code is also at most 9. Therefore, our code is in fact an optimal $(3, 3, 1, 1)$ and $(3, 3, 0, 2)$ robust locally repairable code.

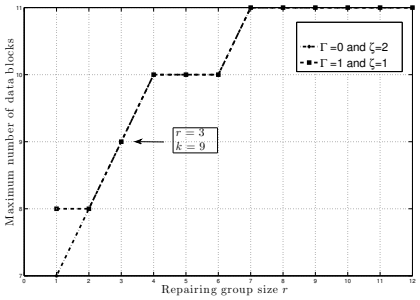

Fig. 3. Upper bounds for $(3, 3, \Gamma, \zeta)$ binary robust locally repairable code of length $n = 16$.

Example 2: Consider a binary linear code of length $n = 8$ and dimension $k = 4$ defined by the following parity check equations:

$$P_1 = C_1 + C_2 + C_3 \tag{16}$$
$$P_2 = C_1 + C_2 + C_4 \tag{17}$$
$$P_3 = C_2 + C_3 + C_4 \tag{18}$$
$$P_4 = C_1 + C_3 + C_4 \tag{19}$$

Again, $C_1, \ldots, C_4$ are the information bits while $P_1, \ldots, P_4$ are the parity check bits. The code can be represented by Figure 4. The above parity check equations imply that the sum of any node and its three adjacent nodes in Figure 4 is always equal to zero.

This code has a minimum distance of 4. According to the Equations (16)-(19), for every single node failure, there exists 7 repair groups with size $r = 3$. For example, suppose C_1 fails. Then, the repair groups are

$$R_1^{C_1} = \{3, 4, 8\}, R_2^{C_1} = \{2, 4, 6\}, R_3^{C_1} = \{2, 3, 5\}$$
$$R_4^{C_1} = \{2, 7, 8\}, R_5^{C_1} = \{3, 6, 7\}, R_6^{C_1} = \{4, 5, 7\}$$
$$R_7^{C_1} = \{5, 6, 8\}$$

Hence, our code is a $(3, 3, 0, 7)$ robust locally repairable code. Furthermore, it can be directly verified that for any distinct $j, k \neq 1$,

$$\left|\left\{\ell : j \notin R_\ell^{C_1}\right\}\right| = 4 \tag{20}$$

and

$$\left|\left\{\ell : \{j, k\} \cap R_\ell^{C_1} = \emptyset\right\}\right| = 2. \tag{21}$$

Therefore, if any one of the surviving nodes is not available, (20) implies that there are still 4 alternative repair groups. This means that our code is also a $(3, 3, 1, 4)$ robust locally repairable code. Similarly, by (21), our code is also a $(3, 3, 2, 2)$ robust locally repairable code.

As shown in our bound (see Figure 5), our code has the highest codebook size for the given parameters among all binary $(3, 3, 0, 7)$, $(3, 3, 1, 4)$ and $(3, 3, 2, 2)$ robust locally repairable codes.

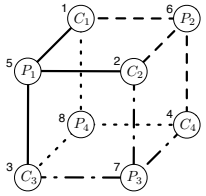

Fig. 4. A robust locally repairable code of length $n = 8$ and dimension $k = 4$ with $\zeta = 7$ repair group for any failure.

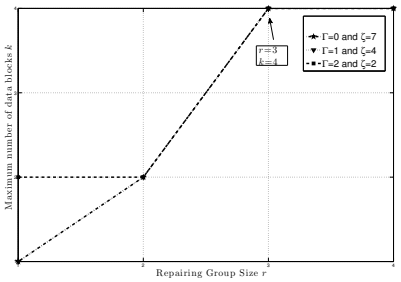

Fig. 5. Upper bounds for $(3, 3, \Gamma, \zeta)$ binary robust locally repairable code of length $n = 8$.

REFERENCES

[1] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 56, no. 9, pp. 4539–4551, Sep 2010.

[2] D. S. Papailiopoulos and A. G. Dimakis, "Locally repairable codes," in *proc. IEEE Int. Symp. Information Theory*, Cambridge, MA, July 2012, pp. 2771–2775.

[3] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, "On the locality of codeword symbols," *IEEE Trans. Inf. Theory*, vol. 58, no. 11, pp. 6925–6934, Nov. 2012.

[4] T. H. Chan, M. A. Tebbi, and C. W. Sung, "Linear programming bounds for storage codes," in *9th International Conference on Information, Communication, and Signal Processing (ICICS 2013)*, Dec. 2013.

[5] G. M. Kamath, N. Prakash, V. Lalitha, and P. V. Kumar, "Codes with local regeneration and erasure correction," *IEEE Trans. Inf. Theory*, vol. 60, no. 8, pp. 4637–4660, August 2014.

[6] L. Pamies-Juarez, H. D. Hollmann, and F. Oggier, "Locally repairable codes with multiple repair alternatives," in *proc. IEEE Int. Symp. Information Theory*, 2013, pp. 892–896.

[7] A. S. Rawat, D. S. Papailiopoulos, A. G. Dimakis, and S. Vishwanath, "Locality and availability in distributed storage," in *proc. IEEE Int. Symp. Information Theory*, 2014, pp. 681–685.

[8] A. Wang and Z. Zhang, "Repair locality from a combinatorial perspective," in *proc. IEEE Int. Symp. Information Theory*, July 2014, pp. 1972–1976.

[9] A. S. Rawat, A. Mazumdar, and S. Vishwanath, "On cooperative local repair in distributed storage," in *48th Annual Conference on Information Sciences and Systems (CISS)*, March 2014, pp. 1–5.

[10] F. J. Macwilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*, North Holland, Amsterdam, 1977.

[11] S. Thakor, T. H. Chan, and K. W. Shum, "Symmetry in distributed storage systems," in *proc. IEEE Int. Symp. Information Theory*, 2013, pp. 1242 – 1246.