Some Results on Update Complexity of a Linear Code Ensemble

Alan Jule∗†
∗R&D group
Axalot
92800 Puteaux, France
alan@axalot.com

Iryna Andriyanova†
†ETIS group
ENSEA/UCP/CNRS-UMR8051
95014 Cergy-Pontoise, France
{alan.jule, iryna.andriyanova}@ensea.fr

Abstract—In this paper, the update complexity of a linear code ensemble (binary or nonbinary) is considered. The update complexity has been proposed in [1] as a measure of the number of updates needed to be done within the bits of a codeword, if one of information bits, encoded in this codeword, has been changed. The update efficiency is a performance measure of distributed storage applications, that naturally use erasure-correction coding. The ensemble maximum complexity and the average complexity are distinguished in this paper. We first propose a simple lower bound on the average update complexity $\gamma_{\text{avg}}$ of a code ensemble and further evaluate a general expression for $\gamma_{\text{avg}}$. Finally, it has been shown that one can upper bound the average update complexity for binary LDPC codes, by using the computation tree approach.

We show that the code ensembles with polynomial minimum distance growth are not update-efficient, i.e. they have a high update complexity. It seems that only code families with sub-polynomial minimum distance (i.e. logarithmic) are update-efficient.

I. INTRODUCTION

The problem of information updating comes from the application of dynamical distributed storage. In a distributed storage system, the data is encoded and then saved on several servers so that each server contains a part of the encoded data block. Encoding the information prevents the loss of data in case one of the servers is not available. Access as many servers as needed to be able to decode the encoded data block will be sufficient to get all the data from the storage system, treating the non-accessed parts of the block as erased. The codes, usually used in distributed storage applications, are Reed-Solomon codes or sparse-graph codes. The encoding/decoding complexity of the code, chosen for use in the storage system, affects the data writing/reading time. Moreover, if the storage system is dynamical (i.e. multiple re-writings of the same data are accepted), one is interested to design a system that allows applying direct updates to the encoded information. Note that here one does not re-encode the whole information block but only a small part of it.

The updating time can be estimated through the update complexity of chosen erasure-correcting code. The notion of update complexity has been first proposed in [1], along with the notion of update-efficient codes. The authors of [1] pointed out that good erasure-correcting codes are not update-efficient, while bad erasure-correcting codes are. By a good erasure-correcting code we understand a code with good minimum distance properties.

This paper is devoted to the investigation of the average update complexity of given linear code ensemble. In the first part of the paper, we propose a simple lower bound on the average update complexity, evaluated using the average weight distribution of the ensemble. Applying this bound to code ensembles with polynomial growth of the minimum distance with the codelength $n$, one obtains that their update complexity is actually linear in $n$. Also, a general expression on the average update complexity is derived. In the second part, we focus ourselves on binary sparse-graph codes, namely on binary LDPC codes, and investigate its average update complexity.

The paper is organized as follows. In Section II, all necessary notions and definitions are given. Section III gives a simple lower bound on the average and maximum update complexity of a code ensemble. Section IV presents a general expression on the average update complexity. Section V proposes a computation tree approach to bound numerically the average update complexity of an LDPC ensemble. Several examples are also given in Sections III, IV, V. Finally, Section VII contains some conclusions.

II. PRELIMINARIES

Consider a code $C$ (binary or nonbinary) of codelength $n$ of dimension $k$ with minimum distance $d_{\text{min}}$. Let us define its maximum and average update complexity:

Definition 1 (Maximum update complexity, [1]): The maximum update complexity $\gamma_{\text{max}}$ of a code is defined as the maximum number of coded bits that must be updated when any single information bit is changed. A code is said to be maximum update efficient if its maximum update complexity is $o(n)$.

Definition 2 (Average update complexity): The average update complexity $\gamma_{\text{avg}}$ of a code is defined as the average number of coded bits that must be updated when any single information bit is changed, averaged over all the information symbols. A code is said to be average update efficient if its average update complexity is $o(n)$.
Clearly, the minimum update complexity of $C$ is $d_{\text{min}}$, and
\[ d_{\text{min}} \leq \gamma_{\text{avg}} \leq \gamma_{\text{max}} \leq n - k + 1. \]
So, if $d_{\text{min}} = O(n)$, the considered code is not update efficient. The interesting case to study is $d_{\text{min}} = o(n)$.

N code is maximum/average update-efficient if it has a systematic generator matrix $G$ such that the maximum/average Hamming weight of its lines is $o(n)$. Therefore, to investigate if the code is update-efficient or not, one may consider the codewords in $C$ which have Hamming weight equal to 1 on information positions (there are exactly $k$ of them).

Let $A(W)$ be the weight polynomial of $C$, i.e., let $A(i)$ denote the number of codewords of weight $i$ in $C$ and let
\[ A(W) = 1 + \sum_{i=d_{\text{min}}}^{n} A(i)W^i. \]  
Let $X$ be a variable, corresponding to information positions in a codeword and $Y$ - to redundancy positions. Then $A(W)$ can be rewritten as
\[ A(X,Y) = 1 + \sum_{i=1}^{k} \sum_{j=d_{\text{min}}+i}^{n} \binom{d_{\text{min}}-i}{i} A(i,j)X^iY^j, \]
where $A(i,j)$ denotes the number of codewords of weight $i$ over the information positions and of weight $j$ over the redundancy positions. If $J = \{j_1, \ldots, j_{\text{max}}\}$ is the set of indices such that $A(1,j) > 0$ if $j \in J$, then one can re-define $\gamma_{\text{max}}$ and $\gamma_{\text{avg}}$ as follows:
\[ \gamma_{\text{max}} = j_{\text{max}} + 1; \]
\[ \gamma_{\text{avg}} = \frac{1}{k} \sum_{j \in J} j A(1,j) + 1. \]

The proposition above is valid for the update complexity in the case of one single code and also for the ensemble update complexity in the case of the code ensemble. It is also easy to obtain the expression of a lower bound on the asymptotic update complexity of a code ensemble. Using Proposition 1 and the definition of $\delta(\alpha)$, one obtains:

**Proposition 2:** Consider a code ensemble of rate $R$, having the growth rate $\delta(\alpha)$ and the average minimum distance $d_{\text{min}}$. Let $\alpha_{\text{min}} = \lim_{n \to \infty} (d_{\text{min}}/n)$ and let $\alpha_{\text{max}}$ satisfy
\[ \int_{\alpha_{\text{min}}}^{\alpha_{\text{max}}} e^{\delta(\alpha)} d\alpha = R. \]

Then the average maximum update complexity of the ensemble is lower bounded by:
\[ \gamma_{\text{max}} \geq n \alpha_{\text{max}} + o(n); \]
\[ \gamma_{\text{avg}} \geq \frac{n}{R} \int_{\alpha_{\text{min}}}^{\alpha_{\text{max}}} \alpha e^{\delta(\alpha)} d\alpha + o(n). \]

Even though the proposed lower bound is extremely simple, it gives a good insight on whether a code or a code ensemble is update-efficient or not. It is easy to see that the update-complexity grows linearly in $n$ for codes with linear minimum distance. In what follows, we are mostly interested in codes, whose $d_{\text{min}}$ grows sublinearly in $n$.

**A. Examples of Asymptotic Lower Bounds for $\gamma_{\text{max}}$ and $\gamma_{\text{avg}}$**

As an example, let us develop asymptotic lower bounds of Proposition 2 for two cases of $\delta(\alpha)$, corresponding to code ensembles with sublinear minimum distances – a polynomial $d_{\text{min}}$ and a logarithmic $d_{\text{min}}$. Note that, in general, the expression of $\delta(\alpha)$ for all values of $\alpha$ is cumbersome. However, as the evaluation of the lower bound corresponds to the integration over $\alpha$’s close to 0 only, one may only consider the expression for $\delta(\alpha)$ around 0.

Our two interesting cases are:
that the word has parity
\[ \gamma \]
Let us derive a general expression for the entry in the word be
\[ \gamma \]
dn which is linear in the codes with logarithmic complexity of the systematic generator matrix G, chosen for a code in this ensemble. Let us construct G line by line and see what is the smallest average weight that one can have. We are going to choose k linearly independent lines of weight w no greater than some fixed weight w0, under the condition that those lines are taken randomly from a distribution A(w).

Among all the possible vectors of weight \( \leq w_0 \), conforming to A(w), some will be linearly dependent. Similarly to the approach used in [4], their number \( N_{ld} \) can be computed as
\[ N_{ld} = \sum_{t=2}^{k-1} \binom{k-1}{t} p_t. \]

We compute \( p_t \) as
\[ p_t = \sum_{\lambda_1, \ldots, \lambda_t} P(w(V_1) = \lambda_1, \ldots, w(V_t) = \lambda_t) p_t(\{\lambda_s\}_s), \]
with
\[ P(w(V_1) = \lambda_1, \ldots, w(V_t) = \lambda_t) = \prod_{s=1}^{t} P(w(V_s) = \lambda_s) \]
and
\[ P(w(V_s) = \lambda_s) = \frac{A(\lambda_s)}{\sum_{i=1}^{N} A(i)}. \]

Let us focus on \( p_t(\{\lambda_s\}_{s=1}^t) \). One has
\[ p_t(\{\lambda_s\}_{s=1}^t) = P(w(\oplus_{i=1}^{t} V_i) \leq w_0 | \{\lambda_s\}_{s=1}^t) \]
\[ = P \left( \sum_{i=1}^{n} (\oplus_{i=1}^{t} V_i) \leq w_0 | \{\lambda_s\}_{s=1}^t \right) \]
By averaging over the code ensemble, one gets that
\[ p_t(\{\lambda_s\}_{s=1}^t) = \frac{P(\oplus_{i=1}^{t} V_i) \leq w_0 | \{\lambda_s\}_{s=1}^t)}{\sum_{i=1}^{n} \lambda_i / p_0}, \]
where \( p_0 \) is the solution of (14), obtained by Proposition 1.

Now we are ready to state our result:

\[ \text{Theorem 1: Let } A(w) \text{ be the average weight distribution of a binary linear code ensemble of codelength } n \text{ and of dimension } k. \]
Putting everything together, one obtains

\[ p_t = \sum_{\lambda_1, \ldots, \lambda_t} \prod_{s=1}^{t} \frac{A(\lambda_s)}{\sum_{i=1}^{w_0} A(i)} \left( \frac{1}{t} \sum_{i=1}^{t} \frac{\lambda_i}{n} \leq p_0 \right) \]

\[ = \sum_{\lambda_1, \ldots, \lambda_t} \prod_{s=1}^{t} \frac{A(\lambda_s)}{\sum_{i=1}^{w_0} A(i)} \mathbb{1} \left( \frac{1}{t} \sum_{i=1}^{t} \frac{\lambda_i}{n} \leq p_0 \right). \quad (13) \]

Hence,

\[ N_{ld} = \sum_{t=2}^{k-1} \binom{k-1}{t} \sum_{\lambda_1, \ldots, \lambda_t} \prod_{s=1}^{t} \frac{A(\lambda_s)}{\sum_{i=1}^{w_0} A(i)} \mathbb{1} \left( \frac{1}{t} \sum_{i=1}^{t} \frac{\lambda_i}{n} \leq p_0 \right), \]

and the number of linearly independent lines is simply the difference of the number of codewords of weight \( w \), where \( 1 \leq w \leq w_0 \), and of \( N_{ld} \).

Similarly, one can prove a more general result for non-binary codes:

**Theorem 2:** Let \( A(w) \) be the average Hamming weight distribution of a linear code ensemble over \( F_q \) of codelength \( n \) and of dimension \( k \). Then the average update complexity \( \gamma_{avg} = w_0 \), where \( w_0 \) verifies (11) with \( p_0 \) being the solution of

\[ 1 - \left[ 1 - 2(q-1)p_0 \right]^{t/2} = \frac{w_0}{n}. \quad (14) \]

**B. An Example**

As an example, consider code ensembles with \( \delta(\alpha) = \kappa \alpha \) for values of \( \alpha \) close to 0 and some \( \kappa > 0 \). Note that such code ensembles have logarithmic \( d_{min} \) and \( A([\alpha n]) = e^{\kappa \alpha n(1+o(1))} \). Let us apply Theorem 1 and investigate the behavior of \( w_0 \) with respect to \( \kappa \).

Fig.1 shows the obtained results, assuming a quite large codelength \( n \), \( n = 10000 \). Quite interesting, in this case \( w_0 \) is of order \( 1/\kappa \). So, for large values of \( \kappa \), the update complexity is expected to be logarithmic in \( n \). Also note that \( \kappa = 1.4 \) (\( \approx \log_2 3 \)) corresponds to the case of binary \((x, x^3)\) LDPC codes.

**V. ON UPDATE COMPLEXITY OF BINARY LDPC CODES**

In the case of sparse-graph codes, instead of considering the ensemble of codes, one may focus on the properties of the corresponding ensemble of bipartite graphs, and to derive some expressions for update complexity. For example, the linear update complexity of Repeat-Accumulate codes can be shown by considering their computation graph. In this section, let us take a particular example of binary \((\lambda(x), \rho(x))\) LDPC codes of some codelength \( n \) and use the computation tree approach to find the update complexity order.

\[ q(p) = \sum_{j \text{ even}} \sum_{i} \hat{p}_i \binom{i-1}{j} p^j (1-p)^{i-j-1}. \]

It can be shown that \( q(p) \) is an unimodal function in \( p \), which attains its maximum for some \( p \in (0, 1) \). Let us start the calculation of \( \gamma_{avg} \). For this we will bound the number \( N_i \) of redundancy edges carrying 1’s at depth \( i \) of the computation tree, constructed above. Let depth 0 correspond to the root node, so that \( N_0 = 0 \). As the computation tree only includes redundancy edges \( \hat{x} \) and check nodes with

\( ^2 \) except the depth 0
Let us consider the order of their update complexity. logarithmic $d$-ensembles. It is well known that the parameters $B$. An Example easy and can be done directly. 

Unfortunately, no simple linear bound on $q$ gives any non-trivial bound on $\gamma_{avg}$. It can be easily shown that a linear upper bound on $q(p)$ gives a trivial bound $\gamma_{avg} \leq O(n)$. Fortunately for us, the numerical evaluation of the upper bound is relatively easy and can be done directly.

### B. An Example

Let us consider regular LDPC code ensembles with parameters $(x, x^3)$ and $(x^2, x^5)$, further noted as (2, 4) and (3, 6) ensembles. It is well known that the (2, 4) ensemble has a logarithmic $d_{min}$, while the (3, 6) ensemble – a linear $d_{min}$. Let us consider the order of their update complexity.

Fig.2 presents the upper bound on $\gamma_{avg}$ for (2, 4) and (3, 6) ensembles with respect to codeweight $n$. One can see that the bound grows linearly for the (3, 6) ensemble and sublinearly for the (2, 4) ensemble. This comes from the fact that, for (2, 4) codes, the decrease of $p_i$ with $i$ in (15) is much faster than the increase of $N_i$ with $i$ in (16). For (3, 6) codes, on the contrary, the decrease of $p_i$ does not compensate the exponential increase of $N_i$.

VI. ACKNOWLEDGMENT

The authors would like to thank E. Soljanin for valuable discussions and the introduction into the topic of update complexity. A.J. is financed by the ANRT under the CIFRE contract No.754/2010.

VII. DISCUSSION

In this paper we give an insight which type of codes is good for dynamical distributed storage applications, i.e. ones where a partial modification of storing data is allowed. In such type of applications, both erasure-correction capability and update complexity matter, and one would like to find a tradeoff between those two parameters. It has been already pointed out in [1] that codes with good erasure-correction capability (linear $d_{min}$) are not update efficient and codes with bad erasure-correction capability are.

In this work, we show that even code ensembles with minimum distance, growing polynomially, are not update-efficient, and hence the best that we can hope for update-efficient codes is to have a sub-polynomially growing $d_{min}$ (i.e. logarithmic growth).

A general expression for the average update complexity of a code ensemble is derived by means of the average weight distribution (AWD) of the ensemble. It can be used to evaluate $\gamma_{avg}$ of a large number of code ensembles, whose growth rate or AWD expressions are already available in the literature.

In the second part of the paper, we use a computation tree approach to bound the average update complexity of binary LDPC codes. Interestingly, in this case the growth of $\gamma_{avg}$ can be explained in terms of the ratio between the increase rate of the number of redundancy messages in the computation tree and the decrease rate of the probability $p_i$: if $p_i$ decreases faster than $N_i$ grows, then the sparse-graph ensemble will be update-efficient, and otherwise if the contrary. The case when $p_i$ and $N_i$ rates compensate each other remains an open question for the moment. Also note that the presented computation tree approach can be easily generalized to non-binary ensembles.

REFERENCES