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 γ_{avg} of a code ensemble and further evaluate a general expression for γ_{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_{min}. Let us define its maximum and average update complexity:

Definition 1 (Maximum update complexity, [1]): The *maximum update complexity* γ_{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* γ_{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_{min}, and

$$d_{min} \leq \gamma_{avg} \leq \gamma_{max} \leq n-k+1.$$

So, if $d_{min} = O(n)$, the considered code is not update efficient. The interesting case to study is $d_{min} = o(n)$.

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[1] in C and let

$$A(W) = 1 + \sum_{i=d_{min}}^{n} A(i)W^i. \quad (1)$$

Let X be a variable, corresponding to information positions in a codeword and Y - to redundancy positions. Then $A(W)$ can be rewritten as

$$A(X,Y) = 1 + \sum_{i=1}^{k} \sum_{j=d_{min}-i}^{n-k} \left(\begin{array}{c} k \\ i \end{array} \right) A(i,j)X^iY^j, \quad (2)$$

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 $J = \{j_1, \ldots, j_{max}\}$ is the set of indices such that $A(1,j) > 0$ if $j \in J$, then one can re-define γ_{max} and γ_{avg} as follows:

$$\begin{aligned} \gamma_{max} &= j_{max} + 1; & (3) \\ \gamma_{avg} &= \frac{1}{k} \sum_{j \in J} jA(1,j) + 1. & (4) \end{aligned}$$

The main issue in determining γ_{max} and γ_{avg} is therefore to determine $\{A(1,j)\}_{j \in J}$ for some given code. This is a difficult problem. To simplify the problem, instead of considering one fixed code C, we are interested in the update complexity of the code ensemble $\mathcal{C}(n,k)$, to which the code C belongs to. If the code ensemble is well defined, the average performance of $\mathcal{C}$ represents the typical ensemble performance. This is the case of many sparse-graph codes [2], interesting from the point of view of distributed storage applications, e.g. LDPC codes, LDGM codes etc. The average/maximum update complexity of a code ensemble $\mathcal{C}$, averaged over all the codes in $\mathcal{C}$, is defined similarly to the case of one single code, and we will denote it by γ_{avg}/γ_{max}. With some abuse of notation, let $A(W)$ also define the average weight distribution of the code ensemble.

Sometimes it is instructive to evaluate the ensemble update complexity in the limit of large codelengths n. In this case γ_{max}/γ_{avg} are rather given in terms of the asymptotic growth rate than of $A(W)$:

Definition 3 (Asymptotic growth rate): Given the average weight distribution $\bar{A}(W)$ of $\mathcal{C}(n,k)$, the asymptotic growth rate of $\mathcal{C}$ is defined as

$$\delta(\alpha) = \lim_{n \to \infty} \frac{\log A(\lfloor \alpha n \rfloor)}{n}$$

[1]for nonbinary codes, one consider the Hamming weight

for $\alpha \in [0,1]$. Hence, in the case of large n,

$$A(\lfloor \alpha n \rfloor) = e^{n\delta(\alpha)+o(n)}.$$

III. Lower bound on γ_{max} and γ_{avg} of a Code and a Code Ensemble

As the update complexity of a code is directly related with the choice of its systematic generator matrix G, one gets a simple lower bound on γ_{max} and γ_{avg} by assuming that the lines of G are the most low-weight codewords in the weight distribution $A(W)$:

Proposition 1: Let i_{max} be the maximum integer satisfying

$$\sum_{i=d_{min}}^{i_{max}} A(i) \leq k. \quad (5)$$

Then

$$\begin{aligned} \gamma_{max} &\geq i_{max}; & (6) \\ \gamma_{avg} &\geq \frac{1}{k} \sum_{i=d_{min}}^{i_{max}} iA(i). & (7) \end{aligned}$$

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_{min}. Let $\alpha_{min} = \lim_{n \to \infty} (d_{min}/n)$ and let α_{max} satisfy

$$\int_{\alpha_{min}}^{\alpha_{max}} e^{n\delta(\alpha)} d\alpha = R. \quad (8)$$

Then the average/maximum update complexity of the ensemble is lower bounded by:

$$\begin{aligned} \gamma_{max} &\geq n\alpha_{max} + o(n); & (9) \\ \gamma_{avg} &\geq \frac{n}{R} \int_{\alpha_0}^{\alpha_{max}} \alpha e^{n\delta(\alpha)} d\alpha + o(n). & (10) \end{aligned}$$

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_{min} grows sublinearly in n.

A. Examples of Asymptotic Lower Bounds for γ_{max} and γ_{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_{min} and a logarithmic d_{min}. Note that, in general, the expression of $\delta(\alpha)$ for all values of α is cumbersome. However, as the evaluation of the lower bound corresponds to the integration over α's close to 0 only, one may only consider the expression for $\delta(\alpha)$ around 0.

Our two interesting cases are:

(a) $\delta(\alpha) = 0$: this case basically corresponds to code ensembles, having $d_{min} = n^a$, $0 < a < 1$, e.g. to Repeat-Accumulate sparse-graph codes. From (8), one obtains $\alpha_{max} = R + \alpha_{min}$, R being the code rate. Then, by (9,10)

$$\begin{aligned} \gamma_{max} &\geq n(R + \alpha_{min}) = Rn + o(n); \\ \gamma_{avg} &\geq \frac{nR}{2} + \alpha_{min} n = \frac{Rn}{2} + o(n). \end{aligned}$$

Therefore, the update complexity in this case is $O(n)$.

(b) $\delta(\alpha) = \alpha_{min} + \kappa\alpha$ for some $\kappa > 0$:
One obtains

$$\alpha_{max} = \frac{1}{\kappa n} \log(\kappa n R e^{-\alpha_{min} n} + e^{\alpha_{min} \kappa n}).$$

Further,

$$\begin{aligned} \gamma_{max} &\geq \frac{1}{\kappa} \log(\kappa n R e^{-\alpha_{min} n} + e^{\alpha_{min} \kappa n}) + o(n) \\ &\approx \frac{1}{\kappa} \log(\kappa k e^{-d_{min}} + e^{\kappa d_{min}}) + o(n) \\ &= d_{min} + k e^{-(1+\kappa) d_{min}} - \frac{\kappa k^2}{2} e^{-2(1+\kappa) d_{min}} \\ &\quad + O(\kappa^2 k^3 e^{-3(1+\kappa) d_{min}}); \\ \gamma_{avg} &\geq \frac{n e^{\alpha_{min} n}}{R} \left(\frac{\alpha_{max} e^{\kappa n \alpha_{max}}}{\kappa n} \right. \\ &\quad \left. - \frac{\alpha_{min} e^{\kappa n \alpha_{min}}}{\kappa n} - \frac{e^{\kappa n \alpha_{max}} - e^{\kappa n \alpha_{min}}}{\kappa^2 n^2} \right). \end{aligned}$$

Considering $\alpha_{min} = 0$ and $d_{min} = a \log n$, which is the case of codes with logarithmic d_{min} (e.g. Gallager cycle codes), one gets

$$\gamma_{max} > c \log n + n^{1 - \frac{1}{c(1+\kappa)}}.$$

So, the lower bound seems to indicate that the code ensembles with polynomial d_{min} have the ensemble update complexity, which is linear in n, and they are not update-efficient. For codes with logarithmic d_{min}, it depends in the value of κ.

IV. Average Complexity γ_{avg}

It would be interesting to find another result on $\gamma_{max/avg}$ for the ensembles with d_{min} growing slower than polynomially. Let us derive a general expression for γ_{avg}.

A. General Expression for γ_{avg}

Let us consider binary codes first. In what follows, we need the following well-known result [3]:

Lemma 1: Consider a binary word of length t and let each entry in the word be 1 with probability p. Then the probability that the word has parity 1 is

$$P(\oplus_{i=1}^{t} b_i = 1) = \frac{1 - (1-2p)^t}{2}.$$

Let us define the following quantity p_t:

Definition 4: Let $V_1, V_2, \ldots, V_t$ be binary vectors of length n. Denote by p_t the probability that the vector $V = \oplus_{i=1}^{t} V_i$ has weight $w_V \leq w_0$.

Now we are ready to state our result:

Theorem 1: Let $A(w)$ be the average weight distribution of a binary linear code ensemble of codelength n and of dimension k. Then the average update complexity $\gamma_{avg} = w_0$, where w_0 verifies

$$\begin{aligned} k = \sum_{i=1}^{w_0} A(i) - \sum_{t=2}^{k-1} \binom{k-1}{t} \\ \cdot \sum_{\lambda_1, \ldots, \lambda_t} \prod_{s=1}^{t} \frac{A(\lambda_s)}{\sum_{i=1}^{w_0} A(i)} \mathbb{1}\left(\frac{1}{t} \sum_{i=1}^{t} \frac{\lambda_i}{n} \leq p_0\right), \end{aligned} \tag{11}$$

with $p_0 \in [0,1]$, being the solution of

$$\frac{1 - (1-2p_0)^t}{2} = \frac{w_0}{n}. \tag{12}$$

Proof: As it has been previously mentioned, the average complexity of a code ensemble is related to the average weight 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 w_0, 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

$$P(w(V_s) = \lambda_s) = \frac{A(\lambda_s)}{\sum_{i=1}^{w_0} A(i)}.$$

Let us focus on $p_t(\{\lambda_s\}_{s=1}^t)$. One has

$$\begin{aligned} p_t(\{\lambda_s\}_{s=1}^t) &= P(w(\oplus_{i=1}^t V_i) \leq w_0 | \{\lambda_s\}_{s=1}^t) \\ &= P\left(\sum_{i=1}^{n} (\oplus_{i=1}^t v_i) \leq w_0 | \{\lambda_s\}_{s=1}^t \right) \end{aligned}$$

By averaging over the code ensemble, one gets that

$$\begin{aligned} p_t(\{\lambda_s\}_{s=1}^t) &= P\left(P(\oplus_{i=1}^t v_i) \leq \frac{w_0}{n} | \{\lambda_s\}_{s=1}^t \right) \\ &= P\left(\frac{1}{t} \sum_{i=1}^{t} \frac{\lambda_i}{n} \leq p_0 \right), \end{aligned}$$

where p_0 is the solution of (14), obtained by Proposition 1.

Putting everything together, one obtains

$$\begin{aligned} p_t &= \sum_{\lambda_1,\ldots,\lambda_t} \prod_{s=1}^{t} \frac{A(\lambda_s)}{\sum_{i=1}^{w_0} A(i)} P\left(\frac{1}{t}\sum_{i=1}^{t}\frac{\lambda_i}{n} \le p_0\right) \\ &= \sum_{\lambda_1,\ldots,\lambda_t} \prod_{s=1}^{t} \frac{A(\lambda_s)}{\sum_{i=1}^{w_0} A(i)} \mathbb{1}(\frac{1}{t}\sum_{i=1}^{t}\frac{\lambda_i}{n} \le p_0). \quad (13) \end{aligned}$$

Hence,

$$N_{ld} = \sum_{t=2}^{k-1} \begin{pmatrix} k-1 \\ t \end{pmatrix} \sum_{\lambda_1,\ldots,\lambda_t} \prod_{s=1}^{t} \frac{A(\lambda_s)}{\sum_{i=1}^{w_0} A(i)} \cdot \mathbb{1}(\frac{1}{t}\sum_{i=1}^{t}\frac{\lambda_i}{n} \le p_0),$$

and the number of linearly independent lines is simply the difference of the number of codewords of weight w,where $1 \le w \le w_0$, 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 $\mathbb{F}_q$ 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

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

B. An Example

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

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 κ, 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.

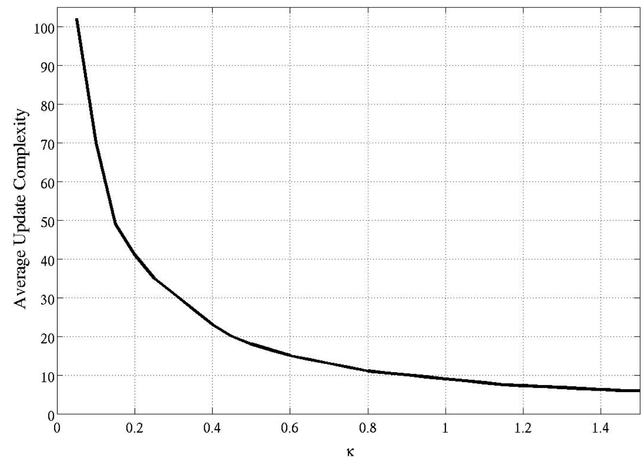

Fig. 1. w_0 vs α for code ensembles with $A(w) = e^{\kappa w}$, $n = 10000$.

A. Particular Case of LDPC Codes

Note that the bipartite graph of a code from the LDPC ensemble contains Rn variable nodes, related to information positions in the codeword, and $(1-R)n$ of nodes, related to redundancy positions (R is the ensemble design rate). Consider the bipartite graph of an LDPC code and pick at random an information variable node as a root node. Construct its computation tree, *only containing redundancy variable nodes.* If one puts the message of the root node to 1 and computes the number of variable nodes, changing their messages, in the computation tree, one gets an estimation of the average update complexity. Unfortunately, this quantity is difficult to compute, due to the presence of cycles in the computation tree. Let us soften the constraint, by only taking into account the cycles which have the larger probability to appear in the computation tree. Note that in this case one gets an upper bound on the average update complexity γ_{avg}. Also note that it is not interesting to remove the constraint of the presence of cycles completely, as in this case one will get $\gamma_{avg} \le n-k$, which is a trivial result.

We will need the following known fact:

Lemma 2: Let q be the probability that a message, coming out from the check node, equals to 1. If $\hat{\rho}(x)$ is the degree distribution at the check node side and p is the probability of the entering message to be 1, then

$$q(p) = \sum_{j \text{ even}} \sum_{i} \frac{\hat{\rho}_i}{j} \begin{pmatrix} i-1 \\ j \end{pmatrix} p^j (1-p)^{i-j-1}.$$

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 γ_{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[2] and check nodes with

[2]except the depth 0

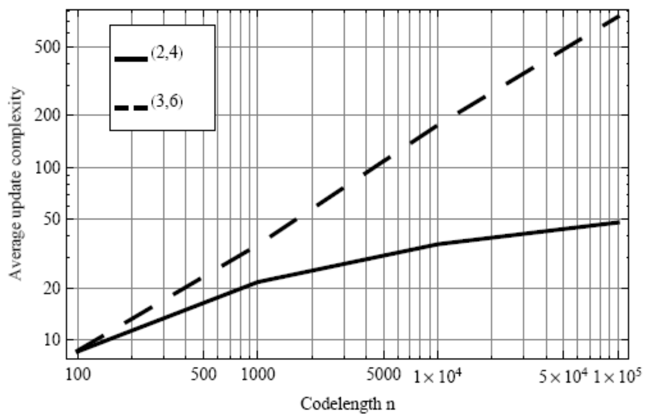

Fig. 2. Upper bound on γ_{avg} vs codelength n for $(2,4)$ LDPC codes (black line) and $(3,6)$ LDPC codes (dashed line).

at least two redundancy connections, the corresponding degree distribution $\hat{\rho}(x)$ is given by the set of $\hat{\rho}_i$'s with $2 \leq i \leq d_{max}$:

$$\hat{\rho}_i = \frac{1}{Z}\sum_{j\in C}\rho_j(1-R)^i R^{j-i},$$

where $C = (3,\ldots,d_{max})$ is the set of check nodes degrees; Z is the normalization constant. Also, at depth i, $i > 1$, we have

$$p_i \leq \frac{N_{i-1}}{n(1-R) - \sum_{j=1}^{i-2}\hat{\rho}'(1)^j N_j}. \tag{15}$$

As for the depth 1,

$$\hat{\rho}_i = \frac{1}{Z}\sum_{j\in C}\rho_j(1-R)^i R^{j-i}$$

with $1 \leq i \leq d_{max} - 1$, and $p_1 = 1/n$. Therefore, $N_1 = \Lambda'(1)q(p_1)$, where $\Lambda(x) = \sum_i \Lambda_i x^i$ denotes the degree distribution of variable nodes from the node perspective. At depth $i > 1$, the number of redundancy variable nodes is

$$N_i \leq \lambda'(1)q\left(p_i\right). \tag{16}$$

Note that from some depth L the computation tree does not contain any additional nodes. One therefore has

$$\gamma_{avg} \leq \sum_{t=1}^{L-1} N_t.$$

Note that this expression is difficult to evaluate analythically. Unfortunately, no simple linear bound on $q(p)$ gives any non-trivial bound on γ_{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 compexity.

Fig.2 presents the upper bound on γ_{avg} for $(2,4)$ and $(3,6)$ ensembles with respect to codelength 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 dymanical 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 γ_{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 γ_{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, than 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

[1] Anthapadmanabhan P., Soljanin E., Vishwanath S.: Update-Efficient Codes for Erasure Correction. In: 48th Annual Allerton Conference, Allerton, Monticello IL (Sep 2010)

[2] Richardson, T., Urbanke, U.: Modern Coding Theory. Cambridge University Press (2008)

[3] Gallager, R.: Low-Density Parity-Check Codes. PhD Thesis. MIT Press (1963)

[4] Ashikhmin, A., Barg, A.: Minimal vectors in linear codes. Trans. Inform. Theory, v.44, no.5, 2010–2017 (1998)