Efficient assistance to LDPC code-based erasure recovery in NVM storage

Pulakesh Upadhyaya
Texas A&M University
13 PUBLICATIONS **15** CITATIONS
SEE PROFILE

Ying Wang
Texas A&M University
11 PUBLICATIONS **111** CITATIONS
SEE PROFILE

K.R. Narayanan
Texas A&M University
253 PUBLICATIONS **5,209** CITATIONS
SEE PROFILE

Efficient Assistance to LDPC Code-based Erasure Recovery in NVM Storage

Anxiao (Andrew) Jiang***, Pulakesh Upadhyaya*****, Ying Wang*****, Krishna Narayanan***
Hongchao Zhou†**, Jin Sima**‡**, and Jehoshua Bruck**‡
* Computer Science and Engineering Department, Texas A&M University
* Electrical and Computer Engineering Department, Texas A&M University
† School of Information Science and Engineering, Shandong University
‡ Electrical Engineering Department, California Institute of Technology

***Abstract*—LDPC Codes are an effective way to correct erasures in local and distributed NVM storage systems. When erasures exceed the code's recovery capability (namely, forming a stopping set), additional tools – e.g., retrieving replicated codeword symbols from remote sites or correcting erasures using natural redundancy – can assist. However, such assistance operations often have high costs and should be minimized. This work presents the correspoding Stopping-Set Elimination Problem, namely, given a stopping set, how to remove the fewest erasures so that the remaining erasures can be decoded by belief propagation in k iterations (including $k = \infty$). The NP-hardness of the problem is proven. An approximation algorithm is presented for $k = 1$. And an efficient exact algorithm is presented for general k when the stopping sets form trees.**

I. Introduction

Non-volatile memories (NVMs) have become a main-stream choice for both local and distributed storage. LDPC codes are an important tool for erasure recovery in NVM systems. In this work, we study a basic problem for LDPC codes: when the erasures in a noisy LDPC codeword cannot be corrected by the decoder, how to remove the fewest erasures so that the remaining erasures become decodable? The problem has several applications:

- *Erasure Recovery in Distributed NVM Storage Systems.* Distributed file systems like HDFS have been widely used in big data applications. Typically, they store data in blocks, and ECCs are applied over the blocks (where each block is seen as a codeword symbol of the ECC). Binary LDPC codes are naturally an attractive candidate for distributed storage, as they have excellent code rates, good locality (e.g., a missing block can be recovered by a local SSD from a few neighboring blocks), and excellent computational simplicity (only XOR is used for decoding, since when each block has t bits, the decoding can be seen as t binary LDPC codes being decoded in parallel). Meanwhile, almost all big IT companies store multiple copies of their data at different locations. So when one site loses some blocks in an LDPC code and cannot recover them by itself, it needs to retrieve some lost blocks from other remote sites. Since communication with remote sites is much more costly than accessing local SSDs, it is desirable to minimize the number of blocks retrieved from remote sites as long as the remaining erasures become decodable.
- *Erasure Correction by Natural Redundancy in NVMs.* Data with rich structures, such as languages or images, often have plenty of redundancy left even after being compressed by standard compression algorithms. Such residual redundancy, called *natural redundancy* (NR), has been used to correct erasures (and errors) beyond the decoding threshold of classic decoders [1], [2], [4]. Such decoding operations by NR are most suitable for NVMs, because they usually require fast random-access speed for lookup in dictionaries (such as dictionaries of words, phrases or image patterns) to assist decoding. However, when the erasure rate is high, many lookups are needed, which is costly even for NVMs. So it is desirable to selectively decide which erasures to recover by NR and minimize their numbers, as long as the remaining erasures become decodable by the classic decoder.

When belief-propagation (BP) decoding fails, the remaining erasures form a Stopping Set. The problem to study can now be defined formally as follows. Let $G = (V \cup C, E)$ be a bipartite graph, where V (representing erasures) is a subset of the variable nodes in an LDPC code's Tanner graph, C is a subset of the check nodes in the same Tanner graph such that every node in C is adjacent to at least one node in V, and E is the set of edges in the Tanner graph with one endpoint in V and another endpoint in C. If every node in C has degree two or more, then G is called a *Stopping Graph* and V is called a *Stopping Set*. Now let $k \geq 1$ be an integer parameter. If an iterative BP algorithm that runs on G can decode all the variable nodes in V (where every variable node in V is an erasure) within k iterations, then V is called a *Decodable Set* (or simply *decodable*); otherwise, it is a *Non-Decodable Set* (or simply *non-decodable*). (Here we introduce the parameter k to make the problem more general, and to control not only the *decodability* of erasures but also the *time* for decoding.) Note that a Stopping Set must be a Non-Decodable Set, but not *vice versa*. The problem we study, called *Stopping-Set Elimination (SSE_k) Problem*, is as follows.

Definition 1. *Given a Stopping Graph $G = (V \cup C, E)$, how to remove the minimum number of variable nodes from V*

such that the remaining variable nodes can be decoded by BP decoding within k iterations? (If the constraint on "k iterations" does not exist, we can see k as being ∞.)

This work was presented at the Allerton Conference of 2017 [3]. In the following, we introduce its main results.

II. Main Results

A. NP-hardness of SSE_k Problems

Theorem 2. *The SSE_k Problem is NP-hard. In particular, even if $k = 1$ or $k = \infty$, the problem is still NP-hard.*

B. Approximation Algorithm for SSE_1 Problem

Theorem 3. *Let d_v and d_c denote the maximum degrees of variable nodes and check nodes, respectively, in the Stopping Graph $G = (V \cup C, E)$. Then there exists an algorithm of time complexity $O(d_v^2 d_c^2 |V|)$ with an approximation ratio of $d_v(d_c - 1)$.*

The proof to the theorem is constructive. An algorithm that achieves the above performance is presented in [3].

C. Optimal Algorithms for Stopping Trees

The Stopping Graph $G = (V \cup C, E)$ can be a tree, especially when the erasure rate is low. In this case, we call G a *Stopping Tree*. Given a Stopping Tree $G = (V \cup C, E)$, we can pick an arbitrary variable node $v \in V$ as the root, run Breadth-First Search (BFS) on G starting with v, and label the nodes of G by $v_1, v_2, \cdots, v_{|V|+|C|}$ based on their order of discovery in the BFS. We denote the resulting BFS tree by G_{BFS}. For any non-root node u in G_{BFS}, let $\pi(u)$ denote its parent. Let G_{sub} denote the subtree of G_{BFS} obtained this way: if we remove the subtree rooted at $\pi(v_{|V|+|C|})$ from G_{BFS}, the remaining subgraph is G_{sub}. We now define a generalization of the SSE_k problem for a stopping tree when k is finite.

Definition 4. *[$gSSE_k$ Problem] Let $G = (V \cup C, E)$ be a Stopping Graph. and let k be a non-negative integer. Every variable node $v \in V$ is associated with two parameters $\delta(v) \in \{1, 2, \cdots, k, \infty\}$ and $\omega(v) \in \{0, 1, \cdots, k, \infty\}$ satisfying the condition that either $\delta(v) = \infty$ or $\omega(v) = \infty$, but not both; and when the BP decoder runs on G, v's value can be recovered (namely, v can become a non-erasure) by the end of the $\delta(v)$-th iteration automatically (namely, without any help from neighboring check nodes). Then, how to remove the minimum number of variable nodes from V such that for every remaining variable node v with $\omega(v) \le k$, it can be corrected by the BP decoder in no more than $\omega(v)$ iterations? (By default, if $\omega(v) = 0$, v has to be removed from V because the BP decoder starts with the 1st iteration.)*

A solution to the $gSSE_k$ Problem (namely, the set of removed nodes) is called a *g-Elimination Set*. We see that if $\delta(v) = \infty$ and $\omega(v) = k$ for every $v \in V$, then the $gSSE_k$ Problem is identical to the SSE_k Problem.

In G_{BFS}, let $\tau \in \{1, 2, \cdots, |V|+|C|\}$ denote the minimum integer such that v_τ either is a sibling of $v_{|V|+|C|}$ or is $v_{|V|+|C|}$ itself. (So $v_\tau, v_{\tau+1}, \cdots, v_{|V|+|C|}$ are siblings.) Define $\mathcal{P} \triangleq \{i \mid \tau \le i \le |V| + |C|, \omega(v_i) \le k\}$ and $\mathcal{Q} \triangleq \{i \mid \tau \le i \le |V| + |C|, \delta(v_i) \le k\}$.

We can reduce the $gSSE$ Problem from G_{BFS} to its subtree G_{sub}. The following lemma considers the case $\max_{i\in\mathcal{P}} \omega(v_i) \le \max_{i\in\mathcal{Q}} \delta(v_i)$.

Lemma 5. *Suppose $\max_{i\in\mathcal{P}} \omega(v_i) \le \max_{i\in\mathcal{Q}} \delta(v_i)$. Consider five cases:*

1) *Case 1: If $|\mathcal{Q}| > 0$ and $\max_{i\in\mathcal{Q}} \delta(v_i) = k$, let S be a minimum-sized g-Elimination Set for G_{sub}.*
2) *Case 2: If $|\mathcal{Q}| > 0$, $\max_{i\in\mathcal{Q}} \delta(v_i) < k$ and $\delta(\pi(\pi(v_{|V|+|C|}))) \le k$, let S be a minimum-sized g-Elimination Set for G_{sub} where $\delta(\pi(\pi(v_{|V|+|C|})))$ is changed to $\min\{\delta(\pi(\pi(v_{|V|+|C|}))), \max_{i\in\mathcal{Q}} \delta(v_i) + 1\}$.*
3) *Case 3: If $|\mathcal{Q}| > 0$ and $\omega(\pi(\pi(v_{|V|+|C|}))) \le \max_{i\in\mathcal{Q}} \delta(v_i) < k$, let S be a minimum-sized g-Elimination Set for G_{sub}.*
4) *Case 4: If $|\mathcal{Q}| > 0$ and $\max_{i\in\mathcal{Q}} \delta(v_i) < \omega(\pi(\pi(v_{|V|+|C|}))) \le k$, let S be a minimum-sized g-Elimination Set for G_{sub} where $\delta(\pi(\pi(v_{|V|+|C|})))$ is changed to $\max_{i\in\mathcal{Q}} \delta(v_i) + 1$ and $\omega(\pi(\pi(v_{|V|+|C|})))$ is changed to ∞.*
5) *Case 5: If $|\mathcal{Q}| = 0$, there are two sub-cases: (1) if $\omega(\pi(\pi(v_{|V|+|C|}))) = 0$, let S be a minimum-sized g-Elimination Set for G_{sub}; (2) otherwise, let S be a minimum-sized g-Elimination Set for G_{sub} where $\delta(\pi(\pi(v_{|V|+|C|})))$ is changed to 1 and $\omega(\pi(\pi(v_{|V|+|C|})))$ is changed to ∞.*

Then $S \cup \{v_i | i \in \mathcal{P}\}$ is a minimum-sized g-Elimination Set for G_{BFS}.

The case for $\max_{i\in\mathcal{P}} \omega(v_i) > \max_{i\in\mathcal{Q}} \delta(v_i)$ can be analyzed similarly. Due to the page limit, we skip its details.

An efficient algorithm that constructs an optimal solution to the SSE_k problem (for finite k) can be built based on the above reductions, and be extended to $k = \infty$. It first runs BFS on G to get the tree G_{BFS} that labels nodes by $v_1, v_2, \cdots, v_{|V|+|C|}$. Then it processes the nodes in the reverse order of their labels, and keeps reducing the $gSSE_k$ Problem to smaller and smaller subtrees, while finding more and more nodes in the solution. The algorithm has linear time complexity $O(|V| + |C|)$. For its detailed pseudo code, please see [3].

Acknowledgment: This work was supported in part by NSF Grant CCF-1718886.

References

[1] A. Jiang, Y. Li, and J. Bruck, "Enhanced Error Correction via Language Processing," in *Proc. Non-Volatile Memories Workshop*, 2015.

[2] A. Jiang, P. Upadhyaya, E. F. Haratsch and J. Bruck, "Error Correction by Natural Redundancy for Long Term Storage," in *Proc. NVMW*, 2017.

[3] A. Jiang, P. Upadhyaya, Y. Wang, K. R. Narayanan, H. Zhou, J. Sima and J. Bruck, "Stopping Set Elimination for LDPC Codes," in *Proc. 55th Annual Allerton Conference on Communication, Control and Computing (Allerton)*, Monticello, IL, October 2017. Available at *http : //faculty.cse.tamu.edu/ajiang/Publications/*2017/ *StoppingSetElimination_Allerton.pdf*.

[4] Y. Wang, K. R. Narayanan and A. Jiang, "Exploiting Source Redundancy to Improve the Rate of Polar Codes," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, pp. 864–868, 2017.