Message Passing Decoding for self-dual F4-additive codes

Constanza Susana Riera
Bergen University College
e-mail: csr@hib.no

Pål Ellingsen
Bergen University College
e-mail: pel@hib.no

Matthew Parker
University of Bergen
e-mail: matthew.parker@uib.no

Hannah Amanda Hansen
Bergen University College
e-mail: hannah@hib.no

Abstract

In this paper, we look at self-dual additive codes over the finite field GF(4). Such codes have a natural graph representation that arises from the code's generator matrix. Working with the graph representations of such codes we investigate a method for doing message-passing decoding.

1 Introduction

The use of the sum-product algorithm has been very successful in the decoding of low density parity check codes [4] [6]. Motivated by the fact that there was no existing fast decoding scheme for GF(4)-additive codes, we have developed an equivalent algorithm for doing message-passing decoding of self-dual additive codes over GF(4) using a natural graph representation of the code [3]. Our algorithm for doing fast decoding of GF(4)-additive codes was first presented in a Master's thesis by Hannah A. Hansen [5], and this paper is based on the results there.

The fields of error-correcting codes and cryptography are linked by the use error-correcting codes for the purpose of cryptography, see e.g. [1], where encrypted messages are treated as messages with noise.

Another application of GF(4)-additive codes is within the field of quantum computing, as it has been shown that such codes can be represented as a self-dual additive code over the field GF(4), see for instance Calderbank et al. [2].

The class of codes we are interested in are the so-called self dual additive codes with respect to the Hermitian inner product over the field GF(4). We shall denote the elements of GF(4) by $\{0, 1, \omega, \omega^2\}$ where $\omega^2 = \omega + 1$. The Hermitian inner product of two vectors u and v over GF(4) is defined as $u \star_4 v = \sum_i u_i v_i^2 \oplus u_i^2 v_i$. Now, it can be proved (see [3]) that the generator matrix G of codes of this type, to within code equivalence, can be written in as $G = \omega I + A$ where A is a symmetric, binary matrix. The matrix A then gives a natural graph representation of the code.

This paper was presented at the NIK-2016 conference; see http://www.nik.no/.

2 Discriminative Decoding

For doing message passing decoding on such codes, we have developed a message passing algorithm that utilizes the graph structure described by the underlying code matrix directly. We call this algorithm *Discriminative Decoding*. The basis of this algorithm is to let the nodes in the graph pass the information they possess about their own state to their neighbors, and repeat this message passing until all nodes agree on their own state, or a maximum number of message passing iterations is reached. The messages sent are simply lenght-4 vectors containing belief values that each bit has about its own state given the word that was received from the channel and the information the node has received from its neighbors.

Normally, the nodes should not need to be aware of the graph structure in such an algorithm, but in our case we differentiate between leaf nodes and internal nodes. A node can easily identify itself as a leaf node by looking at its number of neighbors, and this information is passed to its neighbors. Any node that does not provide this information can then be treated as an internal node. As we shall see in the following, the nodes' status influence how messages from that node is treated.

The basic operation of the algorithm is to send messages indicating each node's belief about its own status to all its neighbors. If the graph has cycles, we iterate over the edges of the graph and pass messages along all edges in the graph. If the graph is a tree, then messages are first passed from the leaves towards the root, and then from the root towards the leaves. In either case, the message passing cycle is repeated until the nodes reach a consensus or until a maximum number of iterations is reached.

Message Calculation

Our algorithm has different strategies for computing the messages to the neighboring nodes depending on the status of each given node. Initially, the nodes in the graph calculates their belief values based on the received values r and the characteristics of the channel. Thus, for a received vector $r \in GF(4)^n$, the initial soft values for a given bit r_i are given by a lenght-4 vector $s_i = (e_i, f_i, g_i, h_i)$ where

$$\begin{aligned} e_i &= P(x_i = 0|r_i) \\ f_i &= P(x_i = 1|r_i) \\ g_i &= P(x_i = \omega|r_i) \\ h_i &= P(x_i = \omega^2|r_i) \end{aligned}$$

Messages are then exchanged between the nodes, and each node combine the received information and return the newly computed belief values to their neighbors. For each neighbor the new message is calculated based on the values received from all neighbors of the originating node, except for the receiving neighbor. This is in order to avoid feedback of erroneous information.

Crucial to the calculations of the messages is a set of four vector operations that are defined below. These operations arise naturally from the structure of the GF(4)-additive codes, particularly in trees.

Definition 2.1 *The pointwise product* $\cdot(u, v) : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$*, of the vectors*

$u = (u_0, u_1, \ldots, u_n)$ *and* $u = (v_0, v_1, \ldots, v_n)$ *is given by:*

$$\cdot(u, v) = \begin{pmatrix} u_0 v_0 \\ u_1 v_1 \\ \ldots \\ u_n v_n \end{pmatrix}$$

Definition 2.2 *The function divided straight-straight is defined by* $dSS : \mathbb{R}^4 \times \mathbb{R}^4 \to \mathbb{R}^4$*, such that for the vectors* $u, v \in \mathbb{R}^4$ *we have:*

$$dSS(u, v) = \begin{pmatrix} v_0 u_0 + v_1 u_1 \\ v_2 u_2 + v_3 u_3 \\ v_0 u_1 + v_1 u_0 \\ v_2 u_3 + v_3 u_2 \end{pmatrix}$$

Definition 2.3 *The function divided straight-cross is defined by* $dSX : \mathbb{R}^4 \times \mathbb{R}^4 \to \mathbb{R}^4$*, such that for the vectors* $u, v \in \mathbb{R}^4$ *we have:*

$$dSX(u, v) = \begin{pmatrix} v_0 u_0 + v_1 u_1 \\ v_0 u_1 + v_1 u_0 \\ v_2 u_2 + v_3 u_3 \\ v_2 u_3 + v_3 u_2 \end{pmatrix}$$

Definition 2.4 *The function twisted straight-cross is defined by* $tSX : \mathbb{R}^4 \times \mathbb{R}^4 \to \mathbb{R}^4$*, such that for the vectors* $u, v \in \mathbb{R}^4$ *we have:*

$$tSX(u, v) = \begin{pmatrix} v_0 u_0 + v_2 u_1 \\ v_0 u_1 + v_2 u_0 \\ v_1 u_2 + v_3 u_3 \\ v_1 u_3 + v_3 u_2 \end{pmatrix}$$

The calculation and exchange of messages then follows depending on the status of the node in question.

For leaves, there is only one possible operation: to send the soft information to its parent. In every iteration of message passing, the leaves will receive updated belief values from their parents and their marginals can be computed taking the pointwise product of the soft values of the leaf and the received belief values.

For internal nodes, the computation of the messages consist of computing the product of the beliefs received from leafs, the *leaf-product*, and the product of the beliefs received from internal nodes, the *internal-product*. The leaf-product and internal-product are computed by using the vector operations defined above on the corresponding set of messages.

The leaf-product is the result of repeatedly applying the tSX-product to all messages that are received from any neighboring leaves, excluding the message from the receiving node.

The internal-product is similarly the result of repeatedly applying the dSX-product to all messages received from any neighboring internal nodes, again excluding the receiving node.

Then, there are two options for the ensuing message calculation:

1. If the node has only leaf nodes as neighbors, or the receiving node is the only internal neighbor, then the message is computed as the dSS-product of the the node's soft information and the leaf-product.

2. If the node has more than one internal node as neighbor, then the message is produced by taking the dSX-product of the leaf-product and the soft information of the node, and then take the dSS-product of this result and the internal-product minus the belief that was previously sent by the receiving node.

The computation of the marginal probabilities for leaves is done in a straightforward way as described above. For internal nodes, the situation is more involved. First, the node computes the leaf-product and the internal-product including all messages that was received from the neighbors. After that, there are three alternatives depending on the geometry of the nodes neighborhood.
Case 1: The node is the center of a star graph. In this case the marginal is given by the pointwise product of its soft information and the leaf-product.
Case 2: The node has only internal nodes as neighbors. In this case the marginal is given by the pointwise product of its soft information and the internal-product.
Case 3: The node has both internal nodes and leaf node as neighbors. In this case the marginal is given by the pointwise product of its soft information and the dSX-product of the leaf-product and the internal-product.

Correctness of the Algorithm

In order to verify the algorithm, one must prove that the calculation of the marginal probabilities for the nodes in the graph using the new decoding algorithm, gives results equal to the global marginals. If this is the case, we also know that our algorithm is an instance of the sum-product algorithm, which is what we want to achieve. The equivalence of the two methods of calculation is proved in [5] by using induction on the tree height, and thus we conclude that the marginals computed using our message passing algorithm are equivalent to the marginals computed by the global function.

3 Performance

Using simulations we have studied the performance of the algorithm described in the previous chapter. We have used two basic cases in the simulations: tree graphs and graphs with cycles of different sizes. The simulations show that the decoding algorithm is correct for the tree graphs used in the simulations, as expected from the underlying theory. For the graphs containing cycles, we made two main observations. When the cycle size of the graph is kept constant, we observed the effect of the number of decoding iterations which clearly shows how the bit- and word error rate decreases with the number of iterations. Then, for graphs with a varying number of 3-cycles and under a constant number of iterations, we observe how the error rate increases significantly with the number of short cycles.

4 Conclusion

In this short paper we have explained the fundamentals of a new algorithm for doing message passing decoding on self-dual GF(4)-additive codes with respect to

the Hermitian inner product, that was first developed in [5].

References

[1] Eli Ben-Sasson, Iddo Ben-Tov, Ivan Damgård, Yuval Ishai, and Noga Ron-Zewi. On public key encryption from noisy codewords. In *IACR International Workshop on Public Key Cryptography*, pages 417–446. Springer, 2016.

[2] A Robert Calderbank, Eric M. Rains, Peter W. Shor, and Neil JA Sloane. Quantum error correction via codes over gf (4). *arXiv preprint quant-ph/9608006*, 1996.

[3] Lars Eirik Danielsen and Matthew G. Parker. On the classification of all self-dual additive codes over gf (4) of length up to 12. *Journal of Combinatorial Theory, Series A*, 113(7):1351–1367, 2006.

[4] Robert G. Gallager. Low density parity check codes. *IRE Trans. Information Theory*, 8:21–28, 1962.

[5] Hannah A Hansen. Message-passing decoding on self-dual $\mathbb{F}_4$-additive codes. Master's thesis, Høgskolen i Bergen, 2016. `http://home.hib.no/ansatte/pel/message-passing-decoding-f4.pdf`.

[6] R. M. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27:533–547, 1981.