An Analysis of the Sum-Product Decoding of Analog Compound Codes

Fangning Hu
EECS Department
International University of Bremen
** Jacobs University Bremen as of spring 2007 **
Campus Ring 1, 28759 Bremen, Germany
f.hu@iu-bremen.edu

Werner Henkel
EECS Department
International University of Bremen
** Jacobs University Bremen as of spring 2007 **
Campus Ring 1, 28759 Bremen, Germany
w.henkel@iu-bremen.edu

***Abstract*— We investigate the sum-product decoding on graphs of analog compound codes and show that the iterative decoding can be completely analyzed by tracing the mean vector at each iteration. A novel geometric analysis is proposed to visualize the iterative decoding process in the Euclidean space. Based on this geometric analysis, we propose to decompose the analog compound codes into several orthogonal constituent code spaces to achieve the fastest convergence speed. Simulations are given to verify our conclusions.**

I. Introduction

Turbo and LDPC codes can be regarded as a kind of compound codes [1] which are defined to be codes consisting of two or several interacting constituent codes. Each constituent code can be represented by a cycle-free graph and is decodable on its own. Indeed, Kschishang [2] has shown that various iterative decoding algorithms can be unified by a single framework by applying the sum-product algorithm on their specific factor graphs.

It has been proven in [3] that when the variable nodes of the graph are continuous real values and are initialized with Gaussian densities, all the messages passed on the graph will be preserved to be Gaussian when applying the sum-product algorithm. The mean of the final density of a symbol (variable node) converges to the true MAP solution and the mean vector which is a vector of the means of all the variable nodes converges to the least-squares solution on arbitrary graphs.

A geometric analysis of a Turbo-like decoding on analog block codes has already been proposed by the authors in [4] as a tool to gain an intuitive insight into Turbo-style decoding. In this paper, we study the geometric properties of the mean vector when exactly applying the sum-product algorithm to compound codes with continuous codewords which we defined as analog compound codes. We find that the estimated codeword after decoding in each constituent code exactly lies in the corresponding constituent code spaces and after several iterations, it is exactly a projection onto the corresponding constituent Euclidean code spaces. Based on this geometric analysis, we propose to decompose analog compound codes into several orthogonal constituent code spaces which makes the algorithm achieve the fastest convergence. Simulations are given to verify our conclusions.

Our primary motivation in studying the analog compound codes is to gain an intuitive insight into the behavior of general Turbo-style decoding. The results in this paper may serve as an intermediate step which on the one hand directly enhances the understanding of the sum-product decoding on graphs with Gaussian variable nodes, on the other hand, may also shed light onto the convergence analysis for discrete variable nodes.

This paper is organized as follows: a factor graph representation of an analog compound code as well as its geometric explanation are introduced in Section II. In Section III, the sum-product decoding as well as the mean vector evolution (MVE) of the algorithm are given. Section IV provides the simulation results.

II. A Factor Graph Representation of Analog Compound Codes

We consider a discrete-time communication system consisting of a source with statistically independent real symbols, a block analog encoder, an AWGN channel, and a decoder. The source information $(s_1, s_2, \ldots, s_K)$ is encoded into the codeword $\boldsymbol{x} \in \mathbb{R}^N$ of length N.

As an example, Fig. 1 shows the factor graph representation of a $(9,4)$ analog compound code with a parity-check matrix

$$\boldsymbol{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & h_{24} & h_{25} & h_{26} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & h_{37} & h_{38} & h_{39} \\ h_{41} & 0 & 0 & h_{44} & 0 & 0 & h_{47} & 0 & 0 \\ 0 & h_{52} & 0 & 0 & h_{55} & 0 & 0 & h_{58} & 0 \\ 0 & 0 & h_{63} & 0 & 0 & h_{66} & 0 & 0 & h_{69} \end{bmatrix} \tag{1}$$

with $h_{ij} \in \mathbb{R}$. This compound code can be decomposed into two cycle-free constituent codes with two sub-parity check matrices

$$\boldsymbol{H}^{(1)} = \begin{bmatrix} h_{11}^{(1)} & h_{12}^{(1)} & h_{13}^{(1)} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & h_{24}^{(1)} & h_{25}^{(1)} & h_{26}^{(1)} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & h_{37}^{(1)} & h_{38}^{(1)} & h_{39}^{(1)} \end{bmatrix},$$

$$\boldsymbol{H}^{(2)} = \begin{bmatrix} h_{11}^{(2)} & 0 & 0 & h_{14}^{(2)} & 0 & 0 & h_{17}^{(2)} & 0 & 0 \\ 0 & h_{22}^{(2)} & 0 & 0 & h_{25}^{(2)} & 0 & 0 & h_{28}^{(2)} & 0 \\ 0 & 0 & h_{33}^{(2)} & 0 & 0 & h_{36}^{(2)} & 0 & 0 & h_{39}^{(2)} \end{bmatrix}$$

with the superscript as the index of the constituent codes. In order to make the graph representation of each constituent code cycle-free, we restrict each column of a sub-parity check matrices to contain at most one non-zero entry. This means, one variable node only connects to one check inside one constituent code.

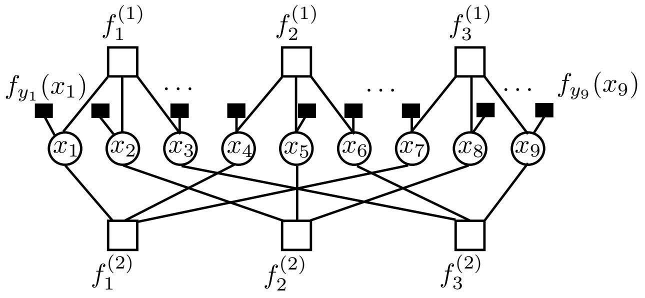

Fig. 1. The factor graph of a $(9,4)$ compound code with two constituent codes.

At the output of the AWGN channel, the received vector can be written as $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{w}$, where $\boldsymbol{w}$ is a random vector with all entries being i.i.d. Gaussian distributed. We follow the notations of the factor graph in [2] for probabilistic modeling where the soft information of each symbol (variable node) is initialized as $p(y_n|x_n)$. Since we only consider the density of x_n, $p(y_n|x_n)$ will be regarded as a function $f_{y_n}(x_n)$ of x_n or a density $p(x_n|y_n)$ of x_n with corresponding received symbol y_n as a parameter. For AWGN, $p(x_n|y_n) \sim p(y_n|x_n) = \mathcal{N}(y_n, \sigma^2)$ with the mean being the received symbol y_n and the variance σ^2 being the channel noise power. The check-node function $f_1^{(1)} = [h_{11}x_1 + h_{12}x_2 + h_{13}x_3 = 0]$ will be 1 if the condition $[h_{11}x_1 + h_{12}x_2 + h_{13}x_3 = 0]$ is satisfied, and 0 otherwise.

Generally, there is no unique decomposition of a compound code into its constituent codes. For example, assumeing all the non-zero entries of $\boldsymbol{H}$ in (1) to be 1, the above code can be seen as a $(9,4)$ analog product code and its sub-parity check matrices become $\boldsymbol{H}^{(1)} = \boldsymbol{I}_3 \otimes \mathbf{1}_3, \boldsymbol{H}^{(2)} = \mathbf{1}_3 \otimes \boldsymbol{I}_3$ with $\mathbf{1}_3$ being an all-1 row vector of length 3, $\boldsymbol{I}_3$ is an identity matrix of size 3 and $\otimes$ is the Kronecker product. Since the last check equation in $\boldsymbol{H}$ is linearly dependent on the other check equations, omitting the last row in $\boldsymbol{H}$ and $\boldsymbol{H}^{(2)}$ will not change the code itself. However, the structure of the second constituent code will be different in its graph representation and hence, there will be an influence on the performance of the graphical decoding. Figure 2 shows two possible decompositions of the $(9,4)$ analog product code.

A further investigation shows that different decompositions of a compound code yield different geometric properties of the resulting constituent code spaces. As an example, we can prove that the first decomposition of the above $(9,4)$ analog product code yields two orthogonal constituent code spaces $\mathcal{G}^{(1)}, \mathcal{G}^{(2)}$ defined by $\mathcal{G}^{(1)} = \{\boldsymbol{H}^{(1)}\boldsymbol{x} = \mathbf{0}\}, \mathcal{G}^{(2)} = \{\boldsymbol{H}^{(2)}\boldsymbol{x} = \mathbf{0}\}$. The intersection of these two constituent code spaces is the original codeword space $\mathcal{G} = \{\boldsymbol{H}\boldsymbol{x} = \mathbf{0}\}$. However, the second decomposition yields two non-orthogonal constituent code spaces. To verify the orthogonality, we arbitrary choose two vectors $\boldsymbol{y}_1, \boldsymbol{y}_2$ lying in the two constituent code spaces

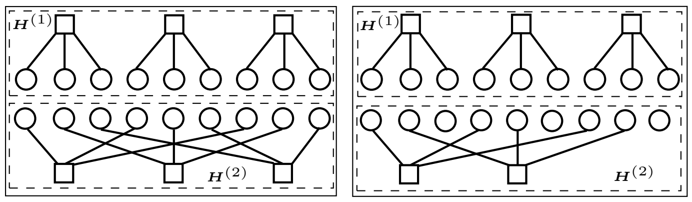

Fig. 2. Two possible ways to decompose the $(9,4)$ analog product code. The left part of the figure corresponds to decomposing the code into two constituent codes with sub-parity check matrices $\boldsymbol{H}^{(1)} = \boldsymbol{I}_3 \otimes \mathbf{1}_3, \boldsymbol{H}^{(2)} = \mathbf{1}_3 \otimes \boldsymbol{I}_3$. In the right part, the last row of $\boldsymbol{H}^{(2)}$ is omitted.

$\mathcal{G}^{(1)}, \mathcal{G}^{(2)}$, respectively, and projecting them onto the codeword space $\mathcal{G}$, resulting in two vectors $\boldsymbol{y}_1^{\perp}, \boldsymbol{y}_2^{\perp}$ which are both orthogonal to the code space $\mathcal{G}$. The angle between the two constituent code spaces is determined by the angle between $\boldsymbol{y}_1^{\perp}$ and $\boldsymbol{y}_2^{\perp}$. The two spaces are orthogonal if the two vectors are orthogonal, i.e., the inner product of these two vectors $\langle \boldsymbol{y}_1^{\perp}, \boldsymbol{y}_2^{\perp} \rangle$ is equal to zero, where $\langle \boldsymbol{y}_1^{\perp}, \boldsymbol{y}_2^{\perp} \rangle = (\boldsymbol{y}_1^{\perp})^T \cdot \boldsymbol{y}_2^{\perp}$ and $(\cdot)^T$ is the transpose. Figure 3 shows the geometric representations of these two decompositions where $\mathcal{G}^{(1)}, \mathcal{G}^{(2)}$ are represented by two intersecting planes with intersecting line as the original code space $\mathcal{G}$. y_1, y_2 are represented by dots and $\boldsymbol{y}_1^{\perp}, \boldsymbol{y}_2^{\perp}$ are represented as arrows in the planes $\mathcal{G}^{(1)}, \mathcal{G}^{(2)}$, respectively.

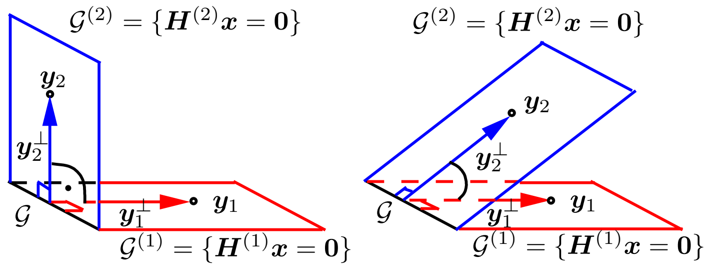
$\mathcal{G}^{(2)} = \{\boldsymbol{H}^{(2)}\boldsymbol{x} = \mathbf{0}\}$
$\boldsymbol{y}_2$
$\boldsymbol{y}_2^{\perp}$
$\mathcal{G}$
$\boldsymbol{y}_1^{\perp}$
$\boldsymbol{y}_1$
$\mathcal{G}^{(1)} = \{\boldsymbol{H}^{(1)}\boldsymbol{x} = \mathbf{0}\}$
$\mathcal{G}^{(2)} = \{\boldsymbol{H}^{(2)}\boldsymbol{x} = \mathbf{0}\}$
$\boldsymbol{y}_2$
$\boldsymbol{y}_2^{\perp}$
$\mathcal{G}$
$\boldsymbol{y}_1^{\perp}$
$\boldsymbol{y}_1$
$\mathcal{G}^{(1)} = \{\boldsymbol{H}^{(1)}\boldsymbol{x} = \mathbf{0}\}$

Fig. 3. The geometric representations of two decomposition of a $(9,4)$ analog product code. The left figure corresponds to the first decomposition which yields two orthogonal constituent code spaces. The right figure corresponds to the second decomposition with two non-orthogonal constituent code spaces.

In the following section, we will propose a criterion based on the geometric properties of the decomposition of the analog compound code to achieve the fastest convergence without deteriorating the performance.

III. The Iterative Decoding and the Mean Vector Evolution

The overall decoding procedure essentially updates the densities on the edges of the graph by the sum-product algorithm inside each constituent code one after another and passes the information from one constituent code to another at each iteration through the variable nodes. We call it a *round* when all the constituent codes are processed once, e.g., a round consists of two iterations for a compound code with two constituent codes.

In the analog case, x_n takes values from an infinite field $(-\infty, +\infty)$ and the soft information for each variable node x_n is represented by a density $f(x_n)$ instead of an LLR value as in the discrete binary case. According to the generic updating rules of the sum-product algorithm, it can be proven that the outgoing density from a check node is a convolution of its incoming densities with a sign change for the considered variable. For example, consider a check node of degree three with the check function $[h_1x_1 + h_2x_2 + h_3x_3 = 0]$. In order to simplify the proof, let $x'_n = h_nx_n, n = 1, 2, 3$. The check function becomes $[x'_1 + x'_2 + x'_3 = 0]$. Let $g_2(x'_2), g_3(x'_3)$ be the Gaussian densities arriving at the check, it can be proven that the resulting outgoing density $g_1(x'_1)$ is

$$g_1(x'_1) = (g_2 \circledast g_3)(-x'_1)\ . \tag{2}$$

The proof is provided in the Appendix.

Based on the fact that the convolution of two Gaussians $\mathcal{N}(\mu_i, \sigma_i^2)$ $(i = 1, 2)$ is still a Gaussian $\mathcal{N}(\mu, \sigma^2)$ with $\mu = \mu_1 + \mu_2$ and $\sigma^2 = \sigma_1^2 + \sigma_2^2$, we obtain the outgoing mean and variance of $g_1(x'_1)$ in (2) as

$$\begin{aligned} \mu_{x'_1} &= -(\mu_{x'_2} + \mu_{x'_2})\ , \\ \sigma^2_{x'_1} &= \sigma^2_{x'_2} + \sigma^2_{x'_3}\ . \end{aligned} \tag{3}$$

The outgoing mean and variance of $g_1(x_1)$ for the original variable x_1 can be obtained by substituting $x'_n = h_nx_n, n = 1, 2, 3$ into (3) as

$$\begin{aligned} \mu_{x_1} &= -\tfrac{1}{h_1}(h_2\mu_{x_2} + h_3\mu_{x_3})\ , \\ \sigma^2_{x_1} &= \tfrac{1}{h_1^2}(h_2^2\sigma^2_{x_2} + h_3^2\sigma^2_{x_3})\ . \end{aligned} \tag{4}$$

According to the generic update rule of the sum-product algorithm, the outgoing density from a variable node is a product of its incoming densities. It can be proven that the outgoing mean from a variable node is a weighted sum of its incoming means since the product of two Gaussians is also a Gaussian with

$$\begin{aligned} \sigma^{-2} &= \sigma_1^{-2} + \sigma_2^{-2}\ , \\ \mu &= (\mu_1\sigma_1^{-2} + \mu_2\sigma_2^{-2})/(\sigma_1^{-2} + \sigma_2^{-2}) \\ &= (\mu_1 + w\mu_2)/(1 + w)\ , \end{aligned} \tag{5}$$

where $w = \sigma_1^2/\sigma_2^2$.

Since all the densities are preserved to be Gaussian and simulation results show that the variance converges to a fixed value after several iterations, the iterative decoding can be completely analyzed by tracing all the means. A final density of a symbol is computed as the product of all the incoming densities to the corresponding variable node. The symbol is estimated based on the MAP rule using the final density. In case of a Gaussian density, the mean achieves the highest probability and thus is the estimate of the symbol. The estimated codeword is exactly the vector consisting of the means of all the variable nodes. The procedure stops when the estimated codeword $\hat{\boldsymbol{x}}$ lies in the code space, i.e., $\boldsymbol{H}\hat{\boldsymbol{x}} = \boldsymbol{0}$.

Note that inside each constituent code, one node is connected at most by one edge. Thus, we can use the same subscript of a variable node to uniquely identify the mean and the variance arriving at or outgoing from this variable node inside one constituent code. Let $U_n^{(i)}(t), u_n^{(i)}(t)$ be the mean and the variance arriving at the nth variable node from a check in the ith constituent code at iteration step t and $V_n^{(i)}(t), v_n^{(i)}(t)$ be the mean and the variance arriving to the ith constituent code computed by other constituent codes through the nth variable node, we can derive the update rule of the mean and the variance at each iteration step. The update rule of one decoding round for a compound code with two constituent codes can be summarized as depicted in Fig. 4. We call the evolution of the mean vector at each iteration the *mean vector evolution* (MVE).

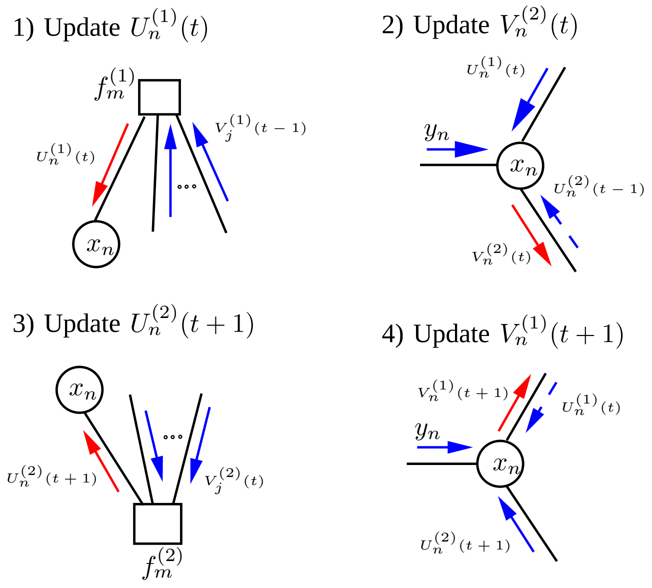

Fig. 4. The mean vector evolution (MVE) of one decoding round of a compound code with two constituent codes. The upper part of the figure illustrates the MVE in the first constituent code, the lower part illustrates the MVE in the second constituent code

In Fig. 4, the mean and the variance are initialized as $V_n^{(1)}(0) = V_n^{(2)}(0) = y_n, v_n^{(1)}(0) = v_n^{(2)}(0) = \sigma^2$ at iteration step $t = 0$. At iteration step t, the mean (extrinsic information) $U_n^{(1)}(t)$ and the variance $u_n^{(1)}(t)$ from the mth check in the first constituent code to the nth variable node is computed according to (4) as

$$U_n^{(1)}(t) = -\frac{1}{h_{mn}^{(1)}}\sum_{j\neq n}^{N} h_{mj}^{(1)}V_j^{(2)}(t-1)\ , \tag{6}$$

$$u_n^{(1)}(t) = \frac{1}{(h_{mn}^{(1)})^2}\sum_{j\neq n}^{N}(h^{(1)})^2_{mj}v_j^{(2)}(t-1)\ . \tag{7}$$

The mean (a-priori information) $V_n^{(1)}(t)$ and the variance $v_n^{(1)}(t)$ which is passed to the second constituent code through the nth variable node are computed according to (5) as

$$V_n^{(2)}(t) = \frac{y_n + w_n^{(1)}(t)U_n^{(1)}(t)}{1 + w_n^{(1)}(t)}\ , \tag{8}$$

$$v_n^{(2)}(t) = \frac{1}{\sigma^{-2} + (u_n^{(1)}(t))^{-1}}\ . \tag{9}$$

with $w_n^{(1)}(t) = \sigma^2/u_n^{(1)}(t)$.

An estimate of the nth symbol can be obtained after the decoding of the first constituent code as a weighted sum of all the means currently entering the node:

$$\hat{x}_n(t) = \frac{y_n + w_n^{(1)}(t)U_n^{(1)}(t) + w_n^{(2)}(t-1)U_n^{(2)}(t-1)}{1 + w_n^{(1)}(t) + w_n^{(2)}(t-1)} . \tag{10}$$

Similarly, $U_n^{(2)}(t+1), V_n^{(1)}(t+1)$ will be updated at iteration step $t+1$ in the second constituent code and an estimate results from

$$\hat{x}_n(t+1) = \frac{y_n + w_n^{(1)}(t)U_n^{(1)}(t) + w_n^{(2)}(t+1)U_n^{(2)}(t+1)}{1 + w_n^{(1)}(t) + w_n^{(2)}(t+1)} . \tag{11}$$

This will finish one decoding round.

For a general compound code with more than two constituent codes, the a-priori information $V_n^{(m)}(t)$ for the mth constituent code is computed by weighted sum of y_n and the extrinsic information $U_n^{(i)}(t), i \in \mathcal{L}/\{m\}$ coming from all the other constituent codes except the mth constituent code

Simulation shows that the estimated codeword $\hat{\boldsymbol{x}}$ after processing the ith constituent code will exactly lie in the ith constituent code space, i.e., $\boldsymbol{H}^{(i)}\hat{\boldsymbol{x}}(t) = \boldsymbol{0}$. This process can be geometrically illustrated as shown in Fig. 5.

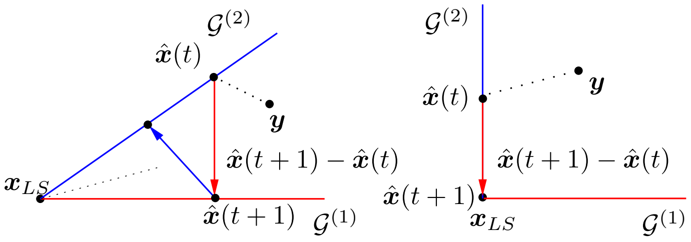
$\mathcal{G}^{(2)}$
$\hat{\boldsymbol{x}}(t)$
$\boldsymbol{y}$
$\hat{\boldsymbol{x}}(t+1) - \hat{\boldsymbol{x}}(t)$
$\boldsymbol{x}_{LS}$
$\hat{\boldsymbol{x}}(t+1)$
$\mathcal{G}^{(1)}$
$\mathcal{G}^{(2)}$
$\hat{\boldsymbol{x}}(t)$
$\boldsymbol{y}$
$\hat{\boldsymbol{x}}(t+1) - \hat{\boldsymbol{x}}(t)$
$\hat{\boldsymbol{x}}(t+1)$
$\boldsymbol{x}_{LS}$
$\mathcal{G}^{(1)}$

Fig. 5. The geometric illustration of the iterative decoding process in non-orthogonal (left) and orthogonal (right) constituent code spaces.

The constituent code spaces $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(2)}$ are illustrated by two intersecting lines in the figure with the intersecting point as the least-squares solution. The received vector $\boldsymbol{y}$ as well as the estimated codewords $\hat{\boldsymbol{x}}(t), \hat{\boldsymbol{x}}(t+1)$ are represented by dots in the figure. Simulation results show that after some iterations, $\hat{\boldsymbol{x}}(t+1) - \hat{\boldsymbol{x}}(t)$ will be orthogonal to the constituent code space $\mathcal{G}^{(i)}$ with i being 1 and 2 in turn at each iteration step. It is obvious from the geometric illustration that it converges fastest when the constituent code spaces are orthogonal to each other.

For a general $M \times N$ parity-check matrix $\boldsymbol{H} = [\boldsymbol{h}_1; \boldsymbol{h}_2; \ldots; \boldsymbol{h}_M]$ with $\boldsymbol{h}_m$ as the mth row, let $\mathcal{G}_m = \{\boldsymbol{x} | \boldsymbol{h}_m \boldsymbol{x} = \boldsymbol{0}\}$ be the mth constituent code space, we can make all the spaces $\mathcal{G}_m, \forall m \in 1, \ldots, M$ orthogonal to each other by converting $\boldsymbol{h}_m, \forall m \in 1, \ldots, M$ to be a set of orthogonal vectors using the Gram-Schmidt process.

IV. Simulation Results and the Geometric Analysis

We decompose a $(25, 16)$ analog product code in two different ways. The first decomposition has two orthogonal constituent code spaces with parity-check matrices $\boldsymbol{H}^{(1)} = \boldsymbol{I}_5 \otimes \boldsymbol{1}_5, \boldsymbol{H}^{(2)} = \boldsymbol{1}_5 \otimes \boldsymbol{I}_5$, the second decomposition has two non-orthogonal constituent code spaces with almost the same parity-check matrices except for omitting the last row in $\boldsymbol{H}^{(2)}$. Assuming the all-zero codeword to be transmitted and the channel variance $\sigma^2 = 1$, the iteration stops when $\boldsymbol{H}\hat{\boldsymbol{x}} = \boldsymbol{0}$. In all the following figures, the dashed line shows the performance of non-orthogonal decomposition and the solid line shows the performance of orthogonal decomposition.

Figure 6 compares the convergence speed. It shows the estimation error $||\hat{\boldsymbol{x}}(t) - \hat{\boldsymbol{x}}_{LS}||$ versus the iteration step t. Our results show that, in both cases, it converges to the least-squares solution. However, it only needs two iterations to converge to the least-squares solution if the constituent code spaces are orthogonal to each other.

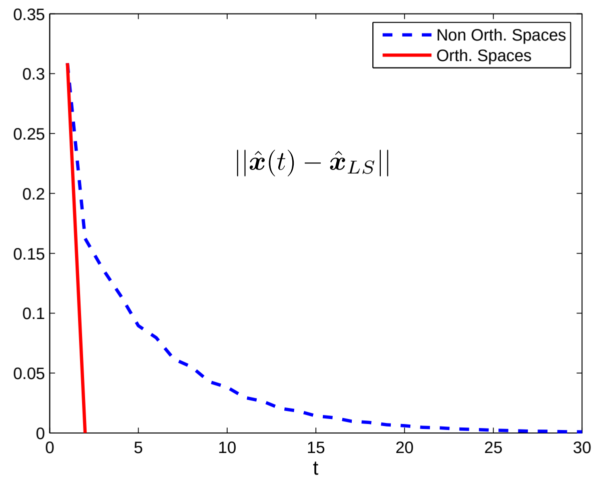

Fig. 6. Comparing the convergence speed between the orthogonal decomposition and the non-orthogonal decomposition of a $(25, 16)$ analog product code when applying the sum-product algorithm.

Figure 7 compares the convergence speed between the orthogonal decomposition and the non-orthogonal decomposition of a $(7, 4)$ analog compound code with a parity-check matrix

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0.3 & 0.9 & 0 & 0.9 & 0 \\ 0 & 0.3 & 0 & 0 & 0.5 & 0 & 0.7 \\ 0.2 & 0.9 & 0.5 & 0.4 & 0.3 & 0.2 & 0.1 \end{bmatrix} ,$$

where each row of $\boldsymbol{H}$ is denoted as $\boldsymbol{h}_m, m = 1, 2, 3$. Here, we first decompose it into two non-orthogonal constituent code spaces with the sub-parity check matrices $\boldsymbol{H}^{(1)} = [\boldsymbol{h}_1; \boldsymbol{h}_2]$ and $\boldsymbol{H}^{(2)} = \boldsymbol{h}_3$. Then, we make the second constituent code space orthogonal to the first one by converting $\boldsymbol{h}_3$ to $(\boldsymbol{h}_3)_{orth}$ which is orthogonal to $\boldsymbol{h}_1, \boldsymbol{h}_2$ using the Gram-Schmidt process, i.e.,

$$(\boldsymbol{h}_3)_{orth} \quad = \quad \boldsymbol{h}_3 - \sum_{m=1}^{2} \frac{\langle \boldsymbol{h}_m, \boldsymbol{h}_3 \rangle}{\langle \boldsymbol{h}_m, \boldsymbol{h}_m \rangle} \boldsymbol{h}_m .$$

Figure 7 shows $\log(||\hat{\boldsymbol{x}}(t) - \hat{\boldsymbol{x}}_{LS}||)$ versus the iteration step t. The iteration stops when the estimation error $||\hat{\boldsymbol{x}}(t) - \hat{\boldsymbol{x}}_{LS}||$

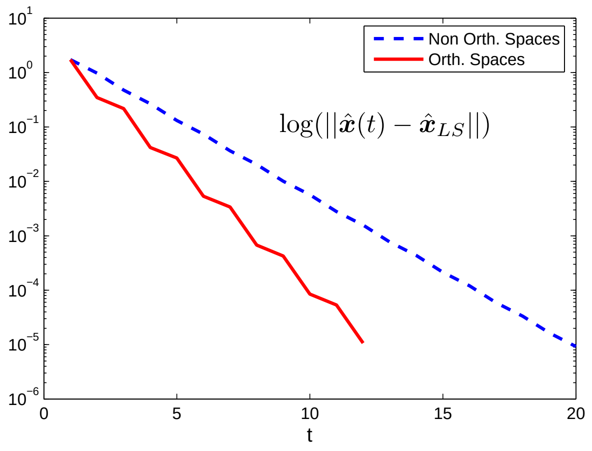

Fig. 7. Comparing the convergence speed of the orthogonal decomposition and the non-orthogonal decomposition of a $(7, 4)$ analog compound code when applying the sum-product algorithm.

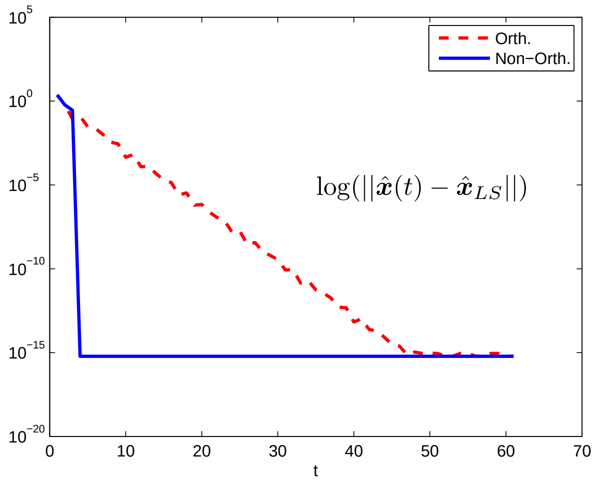

Fig. 8. Comparing the convergence speed of the trade off of the sparseness and the orthogonal decomposition of a $(7, 4)$ analog compound code when applying the sum-product algorithm.

is less than 10^{-5}. It only needs 12 iteration steps for an orthogonal decomposition while 20 iteration steps are needed for a non-orthogonal decomposition for achieving the same estimation error.

We can verify whether the estimated codeword $\hat{\boldsymbol{x}}$ lies in the ith constituent space $\boldsymbol{H}^{(i)}$ at each iteration t by verifying the equality of $\boldsymbol{H}^{(i)}\hat{\boldsymbol{x}}(t) = \boldsymbol{0}$ with i being 1 and 2 in turn at each iteration step. Simulation shows that $\boldsymbol{H}^{(i)}\hat{\boldsymbol{x}}(t)$ is always zero.

We also compare the tradeoff between the sparseness and the orthogonality. We found that sacrificing the sparseness in order to achieve the orthogonality may lead to a worse performance. It is shown in Fig. 8, where the original parity check matrix is:

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0.3 & 0.9 & 0 & 0.9 & 0 \\ 0 & 0.3 & 0 & 0 & 0.5 & 0 & 0.7 \\ 0 & 0.2 & 0 & 0 & 0 & 0.3 & 0 \end{bmatrix}$$

while a orthogonal decomposition yields the third row to be $[-0.0996, 0.1783, -0.0299, -0.0897, -0.0361, 0.2103, -0.0506]$ which leads to a denser matrix than the original one. Thus, our conclusion applies under the condition that the parity check matrices before and after orthogonality have almost the same degree of sparseness.

V. Conclusion

We investigate the sum-product decoding of compound codes with continuous codewords and provide a criterion to decompose the analog compound codes with respect to the convergence speed based on the geometric properties of the constituent codes, i.e., to make the constituent code spaces orthogonal.

Appendix

We provide the proof for the Eqn. (2) of Section III.

Let the density $g_2(x_2'), g_3(x_3')$ be the messages arriving at a check node of degree three with the check function $f(x_1', x_2', x_3') = [x_1' + x_2' + x_3' = 0]$. The resulting outgoing message to variable x_1 will then be $g_1(x_1') = (g_2 \circledast g_3)(-x_1')$.

Proof: According to the generic rule of the sum-product algorithm, the check-to-variable function $g_1(x_1')$ is computed as follows:

$$\begin{aligned} g_1(x_1') &= \sum_{\sim x_1'} [x_1' + x_2' + x_3' = 0] g_2(x_2') g_3(x_3') \\ &= \sum_{x_2'} \sum_{x_3' = -x_1' - x_2'} g_2(x_2') g_3(x_3') \\ &= \sum_{x_2'} g_2(x_2') g_3(-x_1' - x_2') \\ &= (g_2 \circledast g_3)(-x_1') \,. \end{aligned}$$

The notation "$\sim x_1'$" has been adopted from [2] and means that x_1' is a fixed parameter in the sum. The case with three variables can easily be extended to the case of multiple variables.

References

[1] F.R. Kschishang and B.J. Frey, "Iterative Decoding of Compound Codes by Probability Propagation in Graphical Models," *IEEE JSAC*, vol.16, no.2, Feb. 1998.

[2] F.R. Kschishang, B.J. Frey, and H. Loeliger, "Factor Graphs and the Sum-Product Algorithm," *IEEE Trans. on Information Theory*, vol. 47, no. 2, Feb. 2001.

[3] H.A. Loeliger, "An introduction to factor graphs," *IEEE Signal Proc. Mag.*, pp. 28-41, Jan. 2004.

[4] F. Hu, W. Henkel, "A Geometric Description of the Iterative Least-Squares Decoding of Analog Block Codes", 4th International Symposium on Turbo Codes and Related Topics, Munich, April 3-7, 2006.