Performance of the Sum-Product Decoding Algorithm on Factor Graphs With Short Cycles

Kevin Jacobson

Abstract—**Originally invented by R. G. Gallager in 1962, low-density parity-check (LDPC) codes have reemerged as competition to the celebrated, Shannon-limit-approaching, Turbo codes. The sum-product (SP) decoding algorithm (a version of which was proposed by Gallager) is a key technique for decoding LDPCs. The SP algorithm operates on the factor graph representation of the parity check matrix of an error correcting code, allowing approximate conditional probabilities of the codeword bits to be calculated iteratively. Although the decoding accuracy of the SP algorithm is slightly poorer than the optimum maximum likelihood (ML) decoder, its computational complexity (which grows linearly with code length) for large codes is much lower than that for ML (which grows exponentially with code length). It has been postulated that different factor graph representations of the same parity check code result in different algorithm performance. This report shows, via computer simulation on short codes, that this is indeed true, offers some explanation, and reviews a few researchers' work on factor graph analysis.**

Index Terms—**Sum-product decoding, iterative probability propagation, message passing, belief propagation, low density parity check (LDPC) codes, Gallager codes, error correction, factor graph, Tanner graph, pseudocode.**

I. Introduction

THE development of good error correcting codes with reasonable coding and decoding complexity has been the aim of the coding community for about 50 years. The use of codes with structure that allows decoding ease has almost always meant a sacrifice in error control performance. And conversely, codes achieving good error correction properties have usually been too difficult to decode optimally.

Shannon showed in [1] that if the information transmission rate of a communication system remains below the capacity of a noisy channel, arbitrarily low bit error rates can be achieved using infinitely long codewords. Unfortunately, it is infinitely difficult (impossible) to decode an infinitely long codeword. The optimum decoder uses the maximum likelihood (ML) decoding rule, which compares the received symbol $\mathbf{y} = [\ y_0\ y_1\ \ldots\ y_n\ ] = \mathbf{x} + \mathbf{n} = [\ x_0\ x_1\ \ldots\ x_n\ ] + \mathbf{n}$, where $\mathbf{x}$ is the transmitted codeword and $\mathbf{n}$ is an additive white Gaussian noise vector, with all legitimate codewords in the codebook and selects the most likely transmitted codeword. Even with finite codeword lengths, the ML decoder approaches impossibility as the codeword length increases.

In 1962, Gallager [2],[3] proposed low-density parity-check codes (LDPCs) and an iterative decoding technique for decoding them. Nearly forty years later, other researchers [4], [5], [10] rediscovered this class of codes and algorithm as having the promise to achieve the seemingly opposing aims of good error control and decoding ease. Long LDPCs [4],[6] have performed close to the Shannon limit. The key technique to decoding long length LDPC codewords is the sum-product (SP) algorithm operating on the factor graph (also known as Tanner graph) representation of the parity check matrix of a code.

Although the primary suitability for the SP algorithm is for long codes (short codes can generally be decoded using ML), this report looks at short codes because it is much easier to analyze the inner workings of the algorithm on a short code. Results can be carefully extended to longer codes.

II. Factor Graphs and the Sum Product Algorithm

A. Parity Check Matrix

A parity check code, C, can be described using a parity check matrix **H**. Fig. 1a) shows the systematic parity check matrix for an extended Hamming (8,4) code. Each row of the matrix describes a parity check equation. Creating a new parity check matrix, **H'**, from linear combinations of rows of **H** (parity check equations) creates parity check equations on the same code as long as all rows of **H'** are linearly independent. Figs. 1b), 1c) and 1d) show three parity check matrices equivalent to the systematic matrix A.

B. Factor Graph

A factor graph or Tanner graph [5]-[11] is a bipartite graph representing the relationships between the codeword bits (referred to as *variable nodes*) and the parity checks (*check nodes*). *Edges* between variable nodes and check nodes indicate the participation of variable (bit) i in parity check j. A factor graph is created by drawing an edge between each variable node i and check node j wherever matrix element $h_{ji} = 1$. Fig. 2 shows the factor graph for the systematic parity check matrix in Fig.1a).

Submitted as part of the requirement for course ECE682 Error Control Coding, University of Alberta, April 15, 2004.

$$\underset{a)}{\begin{bmatrix}11101000\\11010100\\10110010\\01110001\end{bmatrix}}\ \underset{b)}{\begin{bmatrix}10100101\\11010100\\00010111\\11101000\end{bmatrix}}$$

$$\underset{c)}{\begin{bmatrix}11101000\\01001101\\10011001\\00101011\end{bmatrix}}\ \underset{d)}{\begin{bmatrix}01011010\\10011001\\11010100\\11111111\end{bmatrix}}$$

Fig. 1. Four equivalent parity check matrices for the extended Hamming (8,4) code a) matrix A (systematic form), b) matrix 1, c) matrix 2, d) matrix 5.

A given code has multiple equivalent parity check matrices and therefore multiple equivalent factor graphs. The question investigated by this report is: how does the performance of the SP algorithm differ with different factor graphs and why?

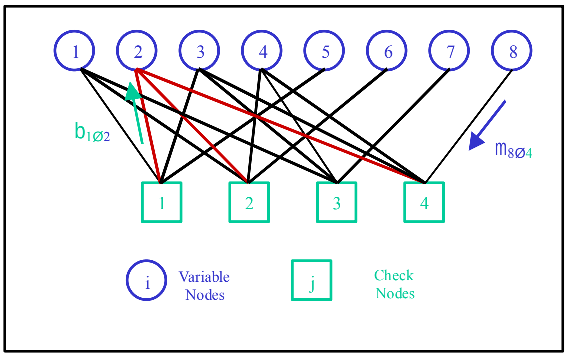


Fig. 2. Factor graph representation of Hamming (8,4) systematic parity check matrix.

C. The SP Algorithm

The SP algorithm[1] is described thoroughly by numerous authors [5],[7]-[12], and will be only briefly described here. Essentially, SP calculates approximations to log-likelihood ratios (LLRs): $\lambda_r = \log[\, p(\, x_r = 1 \mid \mathbf{y}) / p(\, x_r = -1 \mid \mathbf{y})\,]$, $r \in \{1, 2, \ldots, n\}$ for each variable node in order to make decision on each codeword bit, x_r. LLRs are calculated by iteratively passing *extrinsic LLRs* as messages between variable nodes and check nodes. Variable node *i* sends message $\mu_{i\to j}$ to check node *j*, representing the i th node's belief about the corresponding bit being 1 or –1. In turn check node *j* sends a message $\beta_{j\to i}$ representing its belief under the *j* th parity check constraint imposed by the code. Extrinsic LLRs sent to a node contains information gathered from nodes other than the destination node. For example, check node 3 receives messages from variable nodes 1, 3, 4 and 7. The message sent by check node 3 back to variable node 1, $\beta_{3\to 1}$, is combination of messages from received from variable nodes 3, 4 and 7.

Specifically, the extrinsic messages are calculated as follows

$$\beta_{j\to i} = 2\tanh^{-1}\left\{\prod_{l\in C_{j\setminus i}} \tanh\left(\frac{\mu_{l\to j}}{2}\right)\right\}$$

$$\mu_{i\to j} = \sum_{l\in V_{i\setminus j}} \beta_{l\to i}$$

where $C_{j\setminus i}$ is the set of variable nodes connected to check node *j* except variable node *i*, and $V_{i\setminus j}$ is the set of check nodes connected to variable node *i* except check node *j*. The total LLR for node *r* is:

$$\lambda_r = \sum_{l\in V_r} \beta_{l\to r},\, r\in[1,n]$$

Messages are passed back and forth until some stopping criterion is met. This criterion can be a) a preset maximum number of iterations has occurred, b) the λ_r values have converged to some preset accuracy ε, or c) after thresholding the λ_r values, the parity check equations are all satisfied. A combination of these can also be used.

D. Tree Representation of Factor Graph and the Pseudocode

The factor graph described earlier is a graph containing cycles – that is, one can traverse many of the edges between variable and check nodes cyclically, using an infinite number of possible schedules. Normally, the algorithm calculates messages from each variable node in order, and then messages from each check node in order. As discussed in numerous references (for example [10]), conditional bit probabilities calculated by the SP algorithm are exact only for graphs with no cycles. For short codes cycles occur very early, perhaps only after one iteration. For the more useful long LDPC codes, cycles occur much later on due to the low density of edges in the factor graph. However, deviation from optimal decoding does occur even for long LDPC codes as the number of iterations increases.

To visualize how the SP algorithm is executed in a computer program, the factor graph can be redrawn in such a cycle-free tree structure. This tree, on which the SP algorithm operates, is a different code, $\overline{C}$ (termed by some authors as a *pseudocode* and by Wiberg [10] as a *subgraph code*), which is not exactly isomorphic to the original code C. SP performs optimal ML decoding on $\overline{C}$ not on C. Despite this approximation, it turns out that SP performs very well.

The degradation in performance of the SP algorithm on graphs with short cycles has not been well analyzed. However, as described in the section IV, a few researchers have done some work.

Fig. 3 shows the tree representation of the factor graph in Fig. 2, after two iterations using variable node 1 as a root node. Obviously, this tree becomes very complicated very

[1] Other terms used for the SP algorithm and variants are: message passing, belief propagation, and iterative probablility propagation.

quickly. Nevertheless, one can see that each variable or check node appears in the tree code with differing *multiplicity* at each iteration step.

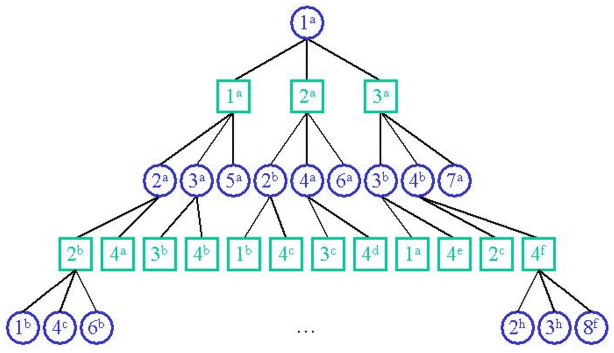


Fig. 3. Tree representation of the factor graph of Fig 2 after two iterations, using variable node 1 as the root node.

Since the local calculations at each multiple node are performed independently, LLRs for different instances of the same bit/check may diverge. Multiple instances of these nodes can be thought of as expansion, iteration step by iteration step, of the original (n, k) code with p = n - k parity checks to a larger (n_{pseudo}, k_{pseudo}) code with $p_{pseudo} = n_{pseudo} - k_{pseudo}$ parity checks. As will be shown in Section V, pseudocode multiplicities increase quickly with the number of iterations, and do not progress equally with each iteration step.

III. Simulation Results

A simulator for the SP algorithm was built in C++, and bit error rate (BER) performance was obtained using a large number of matrices for (8,4) and (16,11) extended Hamming codes.

As mentioned earlier there are three possible iteration-stopping criteria. It was found that the parity check criterion reliably performed better that the LLR convergence check or fixed iteration approaches. The parity check criterion also resulted in more successful decodes in fewer iterations. Fig. 4 shows BER performance, and Fig. 5 shows typical histograms for parity check, LLR convergence check, and parity or LLR convergence check settings. In all cases the decoding algorithm was stopped after 20 iterations and bit decisions made, even if the parity or convergence checks were not satisfied. These decodes were tagged as “unsuccessful decodes” and errors were tabulated separately.

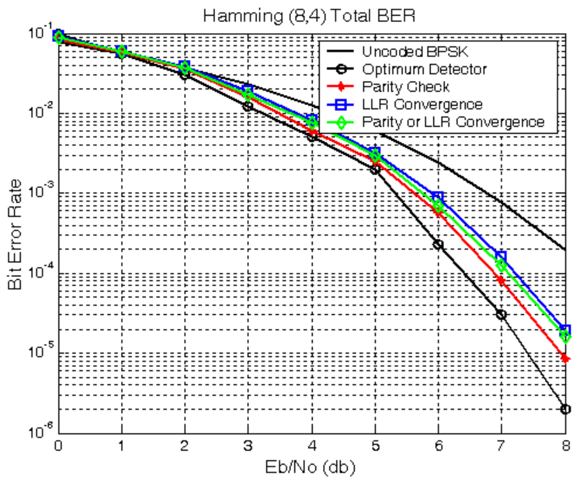


Fig. 4. Algorithm performance with different stopping criteria: parity check only, LLR convergence check, parity or LLR convergence check– Hamming (8,4).

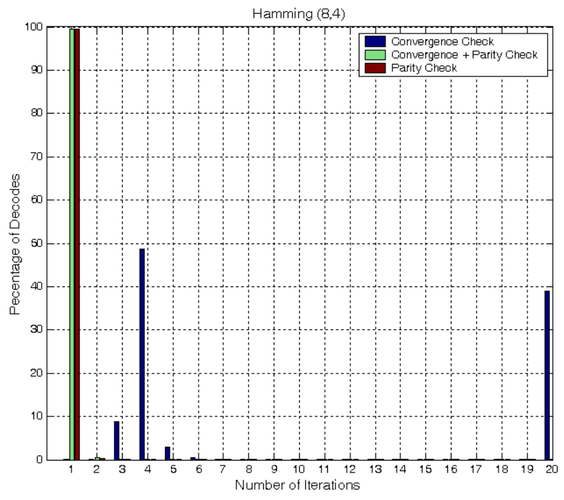


Fig. 5. Algorithm iteration histograms – Hamming (8,4).

Although there exist a large number of possible parity check matrices for each code, many have equivalent properties and thus similar performance. It was useful to categorize them according to column and row weight distributions. The weight of a column or row is the number of non-zero elements in that column or row. The weight of column *i* indicates the number of parity check equations in which the variable node *i* (i.e. bit position *i*) participates. The weight of row *j* indicates the number of variable nodes (bits) that the check node *j* checks. Weight vectors **vnw** and **cnw** describe the variable node and check node weight distributions (number of rows/columns with a given weight) of each parity check matrix. For the (8,4) code, variable node weights range from 1 to 4, and check node weights are either 4 or 8. For the (16,11) code, variable node weights range from 1 to 5, and check node weights are either 8 or 16.

A. (8,4) Hamming Code

There are four different weight distributions of parity check matrices for this code. All matrix examples for each weight

TABLE I
PARITY MATRIX WEIGHT DISTRIBUTIONS FOR (8,4) HAMMING CODE

Matrix Example	Variable Node Weight Distribution **vnw**	Check Node Weight Distribution **cnw**	Total BER	Successful BER
A	{ 4 0 4 0 }	{ 4 0 }	8.33×10^{-6}	2.81×10^{-6}
1	{ 2 4 2 0 }	{ 4 0 }	7.78×10^{-6}	2.99×10^{-6}
2	{ 3 3 1 1 }	{ 4 0 }	7.31×10^{-5}	3.87×10^{-5}
5	{ 1 3 3 1 }	{ 3 1 }	5.83×10^{-5}	1.74×10^{-5}

BERs are for E_b/N_0=8 dB
Matrix A is the systematic form
vnw = { vnw_1 vnw_2 vnw_3 vnw_4 } means there are vnw_1 weight 1 columns, vnw_2 weight 2 columns, vnw_3 weight 3 columns, vnw_4 weight 4 columns
cnw = { cnw_4 cnw_8 } means there are cnw_4 weight 4 rows, cnw_8 weight 8 rows

distribution exhibited identical BER performance for the signal to noise ratios tested (E_b/N_0 from 0 to 8 dB). Fig. 6 shows the BER curves for the sample matrices with each of these weight distributions. At low E_b/N_0, all matrices have similar performance, but the performance curves begin to separate at higher E_b/N_0.

Table I shows the weight distributions of the sample parity check matrices and their BER performance at E_b/N_0 = 8 dB. Total BER counts all bit errors regardless of them being a result of successful decodes (those for which the parity check equations were satisfied) or unsuccessful decodes (those for which the algorithm hit the iteration limit of 20). Successful BER excludes the unsuccessful decodes from the count. The reason for this separation was to see if the unsuccessful decodes accounted entirely for the shift in performance from optimal decoding (which it didn't).

Note the significant degradation in the performance of matrices 2 and 5 compared to matrices A and 1. Note also that matrices 2 and 5 each have a column of weight 4.

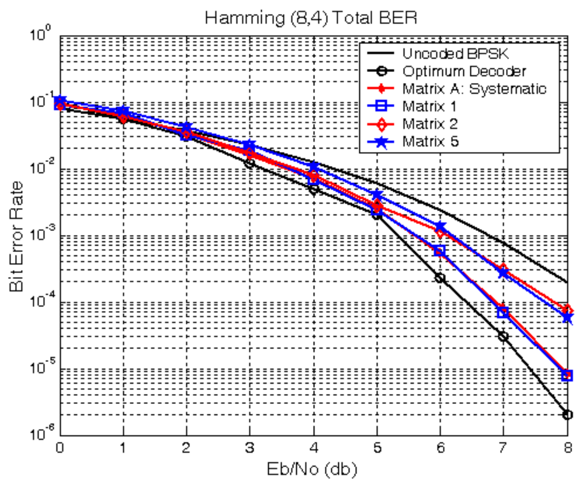


Fig. 6. BER performance of four different parity check matrices – Hamming (8,4).

B. (16,11) Hamming Code

The (16,11) systematic matrix and numerous equivalent matrices were tested for performance. There are a large number of variations of matrices possible, so only a subset with representative characteristics are presented here. BER curves for sample matrices are shown in Fig. 7 and their weight distributions are shown in Table II. Note that matrix 276, with a lower weight spread, has the best performance, and is the only matrix without a weight 5 column.

TABLE II
PARITY MATRIX WEIGHT DISTRIBUTIONS FOR (16,11) HAMMING CODE

Matrix Example	Variable Node Weight Distribution **vnw**	Check Node Weight Distribution **cnw**	Total BER	Successful BER
A	{ 5 0 10 0 1 }	{ 5 0 }	9.24×10^{-6}	7.15×10^{-6}
275	{ 5 0 10 0 1 }	{ 5 0 }	1.13×10^{-5}	9.04×10^{-6}
276	{ 2 6 6 2 0 }	{ 5 0 }	4.94×10^{-6}	1.31×10^{-6}
336	{ 3 6 4 2 1 }	{ 5 0 }	1.13×10^{-5}	6.73×10^{-5}

BERs are for E_b/N_0=8 dB
Matrix A is the systematic form
vnw = { vnw_1 vnw_2 vnw_3 vnw_4 vnw_5 } means there are vnw_1 weight 1 columns, vnw_2 weight 2 columns, vnw_3 weight 3 columns, vnw_4 weight 4 columns, vnw_5 weight 5 columns
cnw = { cnw_8 cnw_{16} } means there are cnw_4 weight 8 rows, cnw_8 weight 16 rows

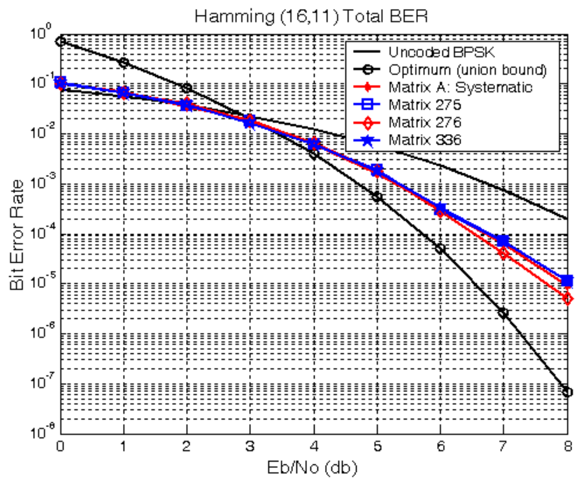


Fig. 7. BER performance of several equivalent parity check matrices – Hamming (16,11). (Note: the union bound is loose at low E_b/N_0.)

IV. RESEARCH ON THE PERFORMANCE OF ITERATIVE ALGORITHMS ON FACTOR GRAPHS

Performance of the SP algorithm on factor graphs with short cycles is not yet very well researched. A few researchers, such as Wiberg [10], Frey, Koetter, and Vardy [14],[15] and Koetter and Vontobel [16], are beginning to develop analysis techniques. This work focuses on analysis of the pseudocode, $\overline{C}$, generated by the computation tree – the cycle-free tree representation of the original code, C.

Wiberg [10] looked at weights of *deviations* $e \in E$ (analogous to detours on a trellis) in the pseudocode (his terminology was subgraph code). He obtains a bound on error probability as a function of these weights:

$$P(\text{decoding error}) \leq \sum_{e \in E} Q\left(\frac{\sqrt{\omega(e)}}{\sigma}\right)$$

where $\omega(e)$ is the weight of deviation e, and E is the set of deviations. In order to calculate this it is necessary to obtain the weight enumeration function for the deviations, which becomes more difficult with the number of iterations. He observes that $\omega(e)$ is small when some nodes occur more frequently that others, which in turn increases the error probability bound.

In order to develop some idea of weights in a factor graph, Koetter and Vontobel [16] analyze the behavior of iterative algorithms using *finite graph covers*. A degree *m* graph cover contains *m* replications of the original factor graph with edges drawn maintaining the node adjacencies of the factor graph. In this representation they are able to show that there exist pseudocodewords with *pseudoweights* less than the minimum codeword weight of the original code. Since the error performance of any code is directly related to distances between codewords, this lower weight, which arises from the iterative decoding process, causes the pseudocode to perform slightly poorer than the original code.

Frey, Koetter, and Vardy [15] present a recursion formula for calculating node multiplicities (the number of times copies of a variable or check node occurs in the tree code) in the computation tree. For the i^{th} node, the length n+p multiplicity column vector $\mathbf{m}^{(l)}$ at half-iteration l, is given by:

$$\mathbf{m}^{(l)} = \mathbf{A}\mathbf{m}^{(l-1)} - (\mathbf{D} - \mathbf{I})\mathbf{m}^{(l-2)}$$

Where $\mathbf{A}$ is the (n+p)x(n+p) node adjacency matrix, $\mathbf{D}$ is a (n+p)x(n+p) diagonal matrix containing the original code node degrees, $\mathbf{I}$ is the (n+p)x(n+p) identity matrix, $\mathbf{m}^{(0)}$ contains a 1 in the i^{th} position and 0s elsewhere, and $\mathbf{m}^{(1)} = \mathbf{A}\mathbf{m}^{(0)} + \mathbf{m}^{(0)}$. An example of this computation is shown in the next section.

Using multiplicities, these authors describe the appearance of pseudosignals – pseudowords that correspond to codewords in C, and spurious pseudocodewords – pseudowords that satisfy the parity checks in $\overline{C}$ but do not have a corresponding codeword. They show that pseudosignals may have positive or negative correlation with codewords. Negatively correlated pseudosignals give rise to decoding errors.

Also described in [15] is skewness – unequal scaling of signal dimensions. This results in a tilting of the decoding boundaries in pseudocode space. A simplfied illustration of this is given in the next section.

V. Analysis of Simulation Results

A. Tree Code Expansion for (3,0) Trivial Code

In [16], Koetter and Vontobel use a trivial zero-rate code as an illustrative example of their analysis. Since the tree codes for the (8,4) and (16,11) codes are far too complicated to draw, it was decided to use their example in order to visualize the appearance of pseudocodewords. Although the (3,0) is clearly a useless code (a codeword is made up entirely of parity bits and no information bits), it is useful to illustrate the appearance of pseudocodewords because there exists only one valid codeword, [0 0 0].

Fig. 8 shows two equivalent matrices for this code, and Fig. 9 shows their tree codes after one full iteration.

$$\underset{A}{\begin{bmatrix}110\\111\\011\end{bmatrix}} \qquad \underset{B}{\begin{bmatrix}110\\111\\100\end{bmatrix}}$$

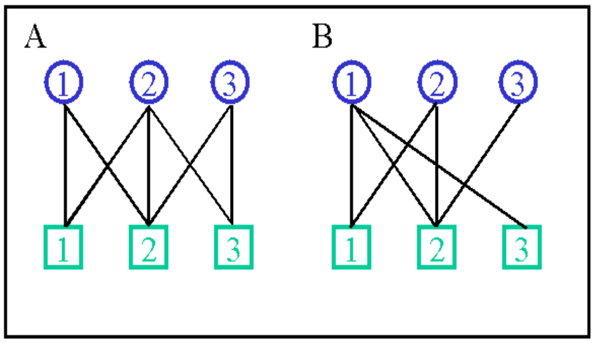
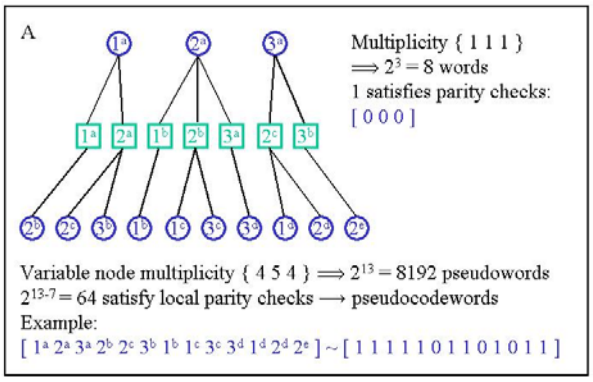
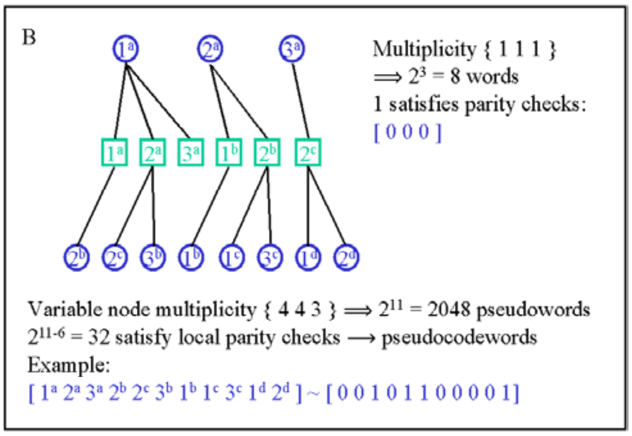


Fig. 8. Two equivalent (3,0) parity matrices and their factor graphs.


A
1a 2a 3a
1a 2a 1b 2b 3a 2c 3b
2b 2c 3b 1b 1c 3c 3d 1d 2d 2e
Multiplicity { 1 1 1 }
⟹ 2³ = 8 words
1 satisfies parity checks:
[0 0 0]
Variable node multiplicity { 4 5 4 } ⟹ 2¹³ = 8192 pseudowords
2¹³⁻⁷ = 64 satisfy local parity checks ⟶ pseudocodewords
Example:
[1a 2a 3a 2b 2c 3b 1b 1c 3c 3d 1d 2d 2e] ~ [1 1 1 1 1 0 1 1 0 1 0 1 1]
Fig. 9a). Tree code for the (3,0) matrix A.


B
1a 2a 3a
1a 2a 3a 1b 2b 2c
2b 2c 3b 1b 1c 3c 1d 2d
Multiplicity { 1 1 1 }
⟹ 2³ = 8 words
1 satisfies parity checks:
[0 0 0]
Variable node multiplicity { 4 4 3 } ⟹ 2¹¹ = 2048 pseudowords
2¹¹⁻⁶ = 32 satisfy local parity checks ⟶ pseudocodewords
Example:
[1a 2a 3a 2b 2c 3b 1b 1c 3c 1d 2d] ~ [0 0 1 0 1 1 0 0 0 0 1]
Fig. 9b). Tree code for the (3,0) matrix B.

Observe that the first matrix has many short cycles, while the second has lost some cycles. Thus the node multiplicities and growth of the pseudocode are lower in the second matrix.

Each matrix has a number of tree pseudocodewords that satisfy *local* parity checks, so with the addition of noise via the transmission channel, LLR messages from multiple checks may converge to an erroneous value. Each variable node collapses the multiple LLRs by summation, and then a decision is made on the variable's bit value. For example, for node 2 of the first matrix, the LLR value for the decision will be

$$\lambda_2 = \beta_{1\to 2} + \beta_{2\to 2} + \beta_{3\to 2}$$

A bit error is made if the individual LLR messages $\beta_{j\to 2}$, which may be converging independently in response to local checks, cause the total LLR λ_2, to sum to the wrong value.

B. Analysis of (8,4) Pseudocode

The ideas of node multiplicity and skewness from [15] were used to analyze the (8,4) code and explain the performance characteristics of the four representative matrices.

The recursive equations given above were used in this analysis. Using matrix A as an example, with variable node 1 as the root node, we have $\mathbf{m}^{(0)} = [\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ ]^T$, $\mathbf{D} = \mathrm{diag}(\ 3\ 3\ 3\ 3\ 1\ 1\ 1\ 1\ 4\ 4\ 4\ 4\ )$, and $\mathbf{A}$ is:

$$\mathbf{A} = \begin{bmatrix} \mathbf{000000001110} \\ \mathbf{000000001101} \\ \mathbf{000000001011} \\ \mathbf{000000000111} \\ \mathbf{000000001000} \\ \mathbf{000000000100} \\ \mathbf{000000000010} \\ \mathbf{000000000001} \\ \mathbf{111010000000} \\ \mathbf{110101000000} \\ \mathbf{101100100000} \\ \mathbf{011100010000} \end{bmatrix}$$

Table III shows the total variable node multiplicity numbers (the variable node portion of multiplicity vector $\mathbf{m}^{(l)}$) for the four tested matrices. Two things are notable from this table. First, replication of original codeword bits does not progress evenly in the pseudocode, and this progression differs for each matrix. Thus it is expected that different matrices process beliefs concerning each bit (as LLRs) differently. For accurate decoding, there should remain positive correlation between pseudocodewords and the original codewords. Second, the number of bits in the pseudocode increase exponentially, meaning that the number of pseudocodewords increase more than exponentially.

TABLE III
VARIABLE NODE MULTIPLICITIES FOR (8,4) HAMMING CODE

V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8
Matrix	**A**						
10	10	10	10	4	4	4	4
46	46	46	46	22	22	22	22
190	190	190	190	94	94	94	94
766	766	766	766	382	382	382	382
Matrix	**1**						
10	7	7	7	4	10	4	7
40	31	31	31	16	40	16	40
142	115	115	115	70	142	70	115
520	403	403	403	232	520	232	403
Matrix	**2**						
7	7	7	4	13	4	4	10
37	37	37	22	49	22	22	46
145	145	145	88	211	88	88	172
571	571	571	352	787	352	352	712
Matrix	**5**						
14	14	8	17	14	11	11	11
104	104	44	125	104	77	77	77
752	752	404	863	752	605	605	605
5282	5282	2510	6335	5282	4043	4043	4043

Each row in the table corresponds to an iteration step: row 1 is after the first iteration, row 2 after the second iteration, etc. Four iterations are shown for each matrix.

Fig. 10 was created using the multiplicity vectors, summing up the appropriate elements to obtain n_{pseudo}, p_{pseudo} and then $k_{pseudo} = n_{pseudo} - p_{pseudo}$. It shows how fast the pseudocode grows with each iteration step. The number of pseudocodewords is $2^{\mathbf{kpseudo}}$.

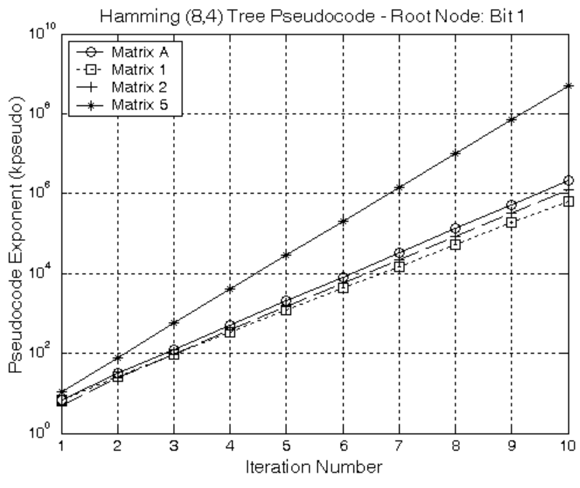


Fig. 10. Expansion of pseudocode size with number of full iterations for the tested matrices – Hamming (8,4).

Fig. 11 was calculated as illustrated by the following example. For matrix A, using node 1 as the root, the variable node portion of multiplicity vector $\mathbf{m}^{(4)}$ (after 2 iterations) is [7 4 4 6 3 6 4 6]. Thus codeword $\mathbf{C}_1 = [\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ ]$ becomes pseudosignal $\overline{\mathbf{C}_1} = [\ 7\ 4\ 4\ 6\ 3\ 6\ 4\ 6\ ]$. The angle between these vectors is

$$\phi = \cos^{-1}\left\{ \frac{\langle \mathbf{C}_1 \overline{\mathbf{C}}_1 \rangle}{\|\mathbf{C}_1\| \|\overline{\mathbf{C}}_1\|} \right\}$$

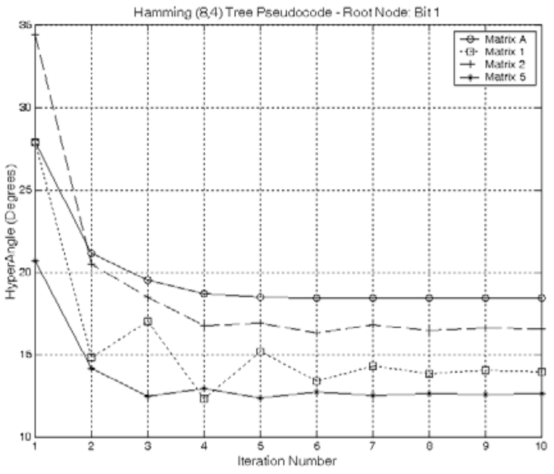


Fig. 11. Angle in 8 dimensional space between codeword [1 1 1 1 1 1 1 1] and its corresponding pseudocodeword – Hamming (8,4).

From these two figures, one can make some conjecture to explain the performance of the different matrices. Recall that matrices A (systematic) and 1 had similar performance, and matrices 2 and 5 were similarly poorer than A and 1. From Fig. 10 we see that matrices A, 1, and 2 have similar pseudocode growth, all slower than matrix 5. From Fig. 11 it is apparent that matrix 2 has the worst skew after only one iteration, while matrices A and 1 have identical skew. The poor performance of matrix 2 can be explained by the slippage of pseudocodewords to the wrong side of the decision boundaries by this great skew, causing negative correlation between pseudocodewords and codewords. Although matrix 5 has lower skew angle, its massive growth in number of pseudocodewords may mean that even small skew can cause more pseudocodewords to correlate to the wrong codewords.

VI. Conclusion

The primary conclusion from this project is that the choice of parity matrix does have an effect on the performance of the SP iterative decoding algorithm. The existence of short cycles in the factor graph degrades performance. In general, it appears that the existence of weight 1 columns, which effectively terminates cycles, is beneficial. Also, a low spread of column weights, resulting in an even progression of node multiplicity, improves performance. More research is still required to fully quantify the analysis of short cycles on factor graphs and determine more solid criteria for factor graph selection for best performance. There is a research opportunity to quantify the pseudoweight distributions of different parity matrices in order to find a bound on error performance.

References

[1] C.E. Shannon, “A mathematical theory of communication,” *Bell Sys. Tech. J.*, vol. 27, 1948, pp. 379-623.

[2] R.G. Gallager, “Low-density parity-check codes,” *IRE Trans. Info. Theory*, vol. IT-8, January 1962, pp. 21-28.

[3] R.G. Gallager, *Low-Density Parity-Check Codes,* MIT Press, Cambridge, MA. 1963.

[4] D.J.C. MacKay, R.M. Neal, “Near Shannon limit performance of low density parity check codes ,” *Elect. Letters*, vol. 33, No. 6, 13th March 1997, pp. 457,458.

[5] R.M. Tanner, “A recursive approach to low complexity codes,” *IEEE Trans. Info. Theory*, vol. IT-27, No. 5, September 1981, pp. 533-547.

[6] S. Chung, et al. “On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit,” *IEEE Commun. Lett.*, vol. 5, February 2001, pp. 58-60.

[7] C. B. Schlegel, L.C. Pérez, *Trellis and Turbo Coding,* IEEE Press, 2003, ch. 8 & 9.

[8] D.J.C. MacKay*, Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, 2003.

[9] F.R. Kschischang, B.J. Frey, H.-A. Loeliger, “Factor graphs and the sum-product algorithm,” *IEEE Trans. Info. Theory*, vol. 47, No. 2, February 2001, pp. 498-519.

[10] N. Wiberg, “Codes and decoding on general graphs,” Ph.D. dissertation 440, Dept. Elect. Eng., Linköping Univ., Linköping, Sweden, 1996.

[11] D.J.C. MacKay, “Good error-correcting codes based on very sparse matrices,” *IEEE Trans. Info. Theory*, vol. 45, No. 2, March 1999, pp. 399-431.

[12] B.J. Frey, D.J.C. MacKay, “A revolution: belief propagation in graphs with cycles,” *Advances in Neural Information Processing Systems 10,* MIT Press, Cambridge MA, presented at Neural Inf. Processing Systems Conf., Denver, CO, December 1997.

[13] J.B. Anderson, S.M. Hladik, “An optimal circular Viterbi decoder for the bounded distance criterion,” *IEEE Trans. Comm.*, vol. 50, No. 11, November 2002, pp. 1736-1742.

[14] B.J. Frey, R. Koetter, A. Vardy, “Skewness and pseudocodewords in iterative decoding,” in *Proc. IEEE Int. Symp. Information Theory*, Cambridge, MA, August 1998, pg. 148.

[15] B.J. Frey, R. Koetter, A. Vardy, “Signal space characterization of iterative decoding,” *IEEE Trans. Info. Theory*, vol. 47, No. 2, February 2001, pp. 766-781.

[16] R. Koetter, P.O. Vontobel, “Graph-covers and iterative decoding on finite length codes,” *Proc. of Turbo Conference,* Brest, 2003.

[17] B. Vucetic, J. Yuan, *Turbo Codes – Principles and Applications*, Kluwer Academic Publishers, 2000.