MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Generating Code Representations Suitable for Belief Propagation Decoding

Jonathan S. Yedidia, Jinghu Chen, and Marc P.C. Fossorier

TR2002-40 September 2002

Abstract

We describe a method for transforming a generalized parity check (GPC) matrix representation of a block linear binary code into another GPC matrix representation of the same code. The output GPC matrix has some attractive features from the point of view of iterative decoding algorithms like belief propagation. In particular, the number of ones in each row is reduced, and there are no cycles of length four in the equivalent graphical representation of the code. We illustrate the method for toy examples including the Golay code, and also for a Euclidean Geometry (n=255,k=127) code. The decoding performance of the belief propagation algorithm using our new GPC matrices improves significantly when they are used on the binary erasure channel, but the results are mixed for the additive white Gaussian noise channel–for the Euclidean Geometry code, the performance still improves, but for the Golay code it deteriorates.

to be published in the Proceedings of the 40th Annual Allerton Conference on Communications, Control, and Computing

Publication History:–

1. First printing, TR-2002-40, September 2002

Generating Code Representations Suitable for Belief Propagation Decoding

Jonathan S. Yedidia
MERL
201 Broadway, 8th Floor
Cambridge, MA 02139
yedidia@merl.com

Jinghu Chen and Marc P. C. Fossorier
Department of Electrical Engineering
University of Hawaii at Manoa
Honolulu, HI 96822
{jinghu,marc}@spectra.eng.hawaii.edu

Abstract

We describe a method for transforming a generalized parity check (GPC) matrix representation of a block linear binary code into another GPC matrix representation of the same code. The output GPC matrix has some attractive features from the point of view of iterative decoding algorithms like belief propagation. In particular, the number of ones in each row is reduced, and there are no cycles of length four in the equivalent graphical representation of the code. We illustrate the method for toy examples including the Golay code, and also for a Euclidean Geometry ($n = 255, k = 127$) code. The decoding performance of the belief propagation algorithm using our new GPC matrices improves significantly when they are used on the binary erasure channel, but the results are mixed for the additive white Gaussian noise channel–for the Euclidean Geometry code, the performance still improves, but for the Golay code it deteriorates.

1 Introduction

The study of error-correcting codes defined by sparse parity check matrices, and message-passing decoding algorithms for such codes, was pioneered nearly four decades ago by Gallager [1]. The practical and theoretical interest in such codes and decoding algorithms has been intense for the last decade, and it is now known that this approach can achieve nearly Shannon-limit error performance in the long block-length limit [2].

For shorter block-lengths, there is still some room for improvement. However, the best codes that have so far been discovered are usually defined in ways that do not immediately give a sparse parity check matrix representation. For this reason, large classes of classical textbook codes, which would give excellent performance under optimal decoding, have been mostly ignored as candidates for message-passing algorithms like the belief propagation (BP) decoding algorithm. A notable exception to this rule are the one-step majority logic decodable codes, which do have sparse parity check matrix representations and have indeed been shown to have excellent error-correcting performance when decoded using BP [3].

The present paper is motivated by the question of whether there are ways to represent other classical codes by sparse parity check matrices, and whether the BP decoding algorithm could be successful using such representations. We report here on an approach

that takes as input any (generalized) parity check matrix representation of a code, and outputs another generalized parity check matrix representation of the same code that will hopefully be more suitable for message-passing decoding algorithms.

The outline of this paper is as follows: in section 2, we review some background material about generalized parity check matrices. In section 3, we give a detailed description of the particular algorithm that we use to re-represent codes. In sections 4 and 5, we describe some empirical results using our algorithm on the Golay code and a multi-step majority logic code based on Euclidean Geometry. The approach seems to work very well for the binary erasure channel (BEC), but the results are more mixed for the additive white Gaussian noise (AWGN) channel. In section 6, we discuss why this may be so and speculate on ways to improve the method.

2 Generalized Parity Check Matrices

In this paper, whenever we refer to "codes," we are referring to binary linear block codes. We recall some elementary facts about such codes. The idea behind codes is to encode messages of k bits using blocks of n bits, where $n > k$. The extra bits are used to help in decoding messages corrupted by noise.

An (n, k) code C is defined by the set of 2^k possible code-words $\mathbf{v}$ of length n. Every (n, k) code can be represented by an $(n-k)$ by n parity check matrix H such that every code-word $\mathbf{v}$ satisfies $\mathbf{v}H^T = 0$, where every addition and multiplication is understood to be over GF(2). For example, for the $(7, 4)$ Hamming code, one possible parity check matrix is

$$H_1 = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}. \tag{1}$$

Other parity check matrices for the same code can be obtained by using linear combinations of the rows of H_1. As long as the resulting rows span an $n-k$ dimensional sub-space, they will give another parity check matrix for the same code.

A parity check matrix might have redundant constraints (rows). For example, another parity check matrix for the $(7, 4)$ Hamming code would be

$$H_2 = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}. \tag{2}$$

Although there are seven rows in H_2, only three of them are linearly independent. As an aside, n by n circulant parity check matrices like H_2 can always be constructed for cyclic codes.

If a parity check matrix has more than n columns, we say that it is a *generalized parity check (GPC) matrix* [4]. In general, an M by N GPC matrix for an (n, k) code will have $N \geq n$ columns, and $M \geq N - k$ rows, where the M rows span an $N - k$ dimensional sub-space. We follow the convention of always listing the n columns of the GPC matrix that correspond to the n bits of a code-word first. We will refer to these

bits as *symbol bits*. The additional $N - n$ columns correspond to *auxiliary bits*, which are also sometimes called "hidden bits" or "state variables." These bits help define the code, but they would not be transmitted along with the n bits of a message.

A GPC matrix H satisfies $\mathbf{x}H^T = 0$, where $\mathbf{x}$ is an N-tuple of bits such that the first n bits form a codeword $\mathbf{v}$, and the last $N - n$ auxiliary bits are uniquely determined given $\mathbf{v}$ and H^T.

An example of a GPC matrix for the $(7, 4)$ Hamming code is the matrix

$$H_3 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3}$$

This matrix is obtained by concatenating the n by n identity matrix with an n by k matrix that is the transpose of a generator matrix G for the code. Such a construction is justified by the fact that any code-word $\mathbf{x}$ of a code can be obtained from the generator matrix G and a k bit message $\mathbf{a}$ using $\mathbf{v} = \mathbf{a}G$. If we interpret the bits of $\mathbf{a}$ as auxiliary bits, we obtain this GPC matrix construction [6].

GPC matrices have straightforward graphical representations in terms of *factor graphs* [5, 6], which can be drawn in a number of slightly different ways. One approach would be to denote the n symbol bits with darkened circles, the $N - n$ auxiliary bits with open circles, and connect both kinds of bits with lines to squares representing the parity check constraints they are involved in. Collectively, we refer to the N symbol and auxiliary bits as *variable bits*.

How to implement the BP (or "sum-product") decoding algorithm using GPC matrices, or their equivalent factor graph representations, is by now very widely known [2, 5, 6], and we will not describe the algorithm here. We simply reiterate the well-known fact that the performance of the BP algorithm will depend on the factor graph (or equivalently GPC matrix) used, even for factor graphs that correspond to the same code. It is this degree of freedom that we wish to exploit.

We will describe a particular algorithm to transform GPC matrices that is motivated by the observation that the BP algorithm seems to work best on parity check matrices that have the following characteristics:

1. The number of ones in each row is small.

2. The number of ones in each column is large.

3. For all pairs of rows of the matrix, the number of columns that have a one in both rows is small; ideally zero or one. (The corresponding characteristic of the factor graph for the code would be that it ideally has no cycles of length four.)

Our algorithm produces GPC matrices which are improved in all three of these characteristics. On the other hand, the new GPC matrices also have new auxiliary bits, and because there is no evidence from the channel to determine the value of these bits, their uncertainty might cause performance to deteriorate. Which tendency will dominate is not *a priori* clear, and as we shall see, seems in fact to depend surprisingly on details of the decoding scenario.

3 Description of the Algorithm

We now give a detailed description of the particular algorithm we used to transform GPC matrices. We assume that we are given an M by N input GPC matrix H, and our goal is to construct an M' by N' output GPC matrix H'. To illustrate our algorithm, we will use as the input a parity check matrix for the $(7, 4)$ Hamming code already mentioned previously but which we repeat here:

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}. \tag{4}$$

The basic idea behind the algorithm is to re-write constraints involving large numbers of bits by using auxiliary bits that encode the parity of sets of bits. Using this "divide-and-conquer" approach, we try to minimize the number of bits involved in each constraint by re-writing the constraints in terms of sets of bits that are as large as possible. We also try to use as many redundant constraints as possible.

3.1 Forming Sets of Bits

The first step of our algorithm is to form sets of the N input variable bits. To be completely precise, we form three sets of bits, which we call *constraint sets*, *intersection sets*, and *single bit sets*.

The constraint sets and single bit sets are formed trivially. The M constraint sets simply contain the variable bits involved in each constraint (row) of H. In our example, they are $\{1, 2, 3, 5\}$, $\{2, 3, 4, 6\}$, and $\{3, 4, 5, 7\}$. The N single bits sets are sets containing a single one of the N variable bits. In our example, they are $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$, $\{5\}$, $\{6\}$, and $\{7\}$.

We form intersection sets by taking intersections of all possible pairs of the constraint sets. If any of the intersections contain fewer than two variable bits, we discard them. When possible, we then form new intersection sets by taking intersections of the previously created intersection sets, always discarding sets of bits that duplicate previously created intersection sets or contain fewer than two bits. The procedure is precisely the same as that used in forming "regions" in the cluster variation method [7, 8]. In our example, we would obtain the intersection sets $\{2, 3\}$, $\{3, 4\}$, $\{3, 5\}$.

There may be reasons to discard some of the intersection sets and there is certainly no strict need to keep all of them. Any intersection sets that are kept will ultimately correspond to new auxiliary bits for the output GPC matrix.

3.2 Organizing the Sets into a Partially Ordered Set

We next organize all the sets of variable bits retained, whether they be constraint sets, intersection sets, or single bit sets, into a *partially ordered set* [9, 8] of sets. When a set of bits S is a sub-set of another set of bits T, we write $S < T$. If $S < T$, and there is no other set of bits U such that $S < U < T$, we say that S is a *child* of T and T is a *parent* of S. In our example the set $\{1, 2, 3, 5\}$ is a parent of the set $\{2, 3\}$ and of $\{1\}$, but it is not a parent of $\{2\}$, because $\{2\} < \{2, 3\} < \{1, 2, 3, 5\}$.

The partially ordered set of sets that we obtain in our example is illustrated in figure 1. In this diagram, sets of bits that are connected by a line are related by a parent-child relationship.

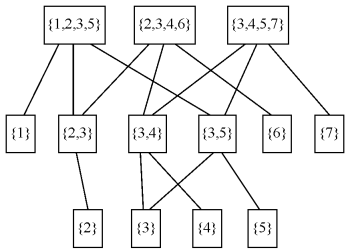

Figure 1: A partially ordered set of sets of variable bits.

3.3 Forming Lists of Sub-sets

For each of the constraint and intersection sets, we next list all of its sub-sets, listing children first, and then all remaining sub-sets. Among the lists of children and sub-sets, we always list the sets containing more bits first, but otherwise the order is arbitrary. In our example, we would obtain the following lists:

$\{1,2,3,5\}$: children $\{2,3\}$, $\{3,5\}$, $\{1\}$, other sub-sets $\{2\}$, $\{3\}$, $\{5\}$.
$\{2,3,4,6\}$: children $\{2,3\}$, $\{3,4\}$, $\{6\}$, other sub-sets $\{2\}$, $\{3\}$, $\{4\}$.
$\{3,4,5,7\}$: children $\{3,4\}$, $\{3,5\}$, $\{7\}$, other sub-sets $\{3\}$, $\{4\}$, $\{5\}$.
$\{2,3\}$: children $\{2\}$, $\{3\}$.
$\{3,4\}$: children $\{3\}$, $\{4\}$.
$\{3,5\}$: children $\{3\}$, $\{5\}$.

3.4 Breaking Down Sets into Unions of Sets

For each constraint set and intersection set, we next find one or more collections of non-overlapping sub-sets that include every bit in the set, using the following procedure. For each set S, we start with the first child on its list, and remove the bits in the child from S. For example for the set $\{1,2,3,5\}$, the first child would be $\{2,3\}$, and the bits remaining after removing the child bits would be $\{1,5\}$. Next, we try to find the first sub-set on the list for S that contains a sub-set of the remaining bits. In our case that would be $\{1\}$. Again we remove those bits, and repeat until all bits are accounted for in non-overlapping sub-sets. In our example, we would obtain $\{1,2,3,5\} = \{2,3\} \cup \{1\} \cup \{5\}$.

We then attempt to find additional redundant ways to break down each constraint set or intersection set S, by using the first child in the list for S that was not previously involved in a union, and repeating the above procedure. In our example we would thereby obtain $\{1,2,3,5\} = \{3,5\} \cup \{1\} \cup \{2\}$. We continue until every child is involved in at least one union, and repeat for every constraint and intersection set.

3.5 Converting to a GPC matrix

We denote the values of the N variable bits of the input GPC matrix by x_i, where i is an index running from 1 to N. For any set of bits S, we use the notation x_S to denote the modulo-two sum of the bits in S. For example, if $x_1 = 1$, $x_2 = 0$, and $x_3 = 1$, then $x_{\{1,2,3\}} = 1 + 0 + 1 = 0$.

We can convert each of the above expressions for a set in terms of a union of subsets into a parity check equation. For example, the equation $\{1,2,3,5\} = \{2,3\} \cup \{1\} \cup \{5\}$

is equivalent to $x_{\{1,2,3,5\}} = x_{\{2,3\}} + x_{\{1\}} + x_{\{5\}}$. We also know that for every constraint set S, $x_S = 0$.

We can thus re-write all the equations for sets as a union of sub-sets as an equivalent GPC matrix, which for our example would be

$$H' = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{5}$$

The first seven columns of H' correspond to the original seven symbol bits, while the last three columns correspond to the parity of the intersection sets $\{2,3\}$, $\{3,4\}$, and $\{3,5\}$. The first six rows are constraints derived using the three constraint sets, while the last three are derived using the three intersection sets.

Notice that the average number of ones per row has been reduced, the average number of ones per column has been increased, and there are no longer any four-cycles in the corresponding factor graph.

In general, however, the procedure described so far does not guarantee that one produces a GPC matrix with no four-cycles in the corresponding factor graph. In particular, it is possible that the same pair of sub-sets is used in the union equations for two different sets of variable bits. We will not prove it here, but if eliminating four-cycles in the factor graph is considered a priority, it can always be arranged by pruning rows from the output GPC matrix. One merely needs to ensure that there remains at least one parity check equation corresponding to each constraint set and to each intersection set, which ensures that the output GPC matrix will still represent the same code as the original GPC matrix.

Note that the complexity of the BP algorithm scales with the number of iterations needed and the number of ones in the GPC matrix, so although using H' is more costly than using H, the increase in computational expense is much less than one might naively guess by simply looking at the sizes of the GPC matrices.

4 Empirical Results for the Golay Code

We now present some empirical results obtained applying this procedure to the $(23, 12)$ Golay code, which should be considered a "toy" example. We began with a 23 by 23 circulant input parity check matrix H with eight ones per row and column obtained using the parity check polynomial $h(x) = 1 + X + X^2 + X^3 + X^4 + X^7 + X^{10} + X^{12}$ [10]. Applying the procedure described in the previous section, we obtained an output GPC matrix H' with 495 rows and 230 columns. Each row had between three and five ones in it.

Using standard BP decoding (all BP decoding algorithms described here ran until a code-word was found or a fixed number of iterations were reached; the fixed number was 300 iterations unless stated otherwise) on the BEC yielded the results shown on the left hand side of figure 2. Using BP on the output GPC matrix worked considerably

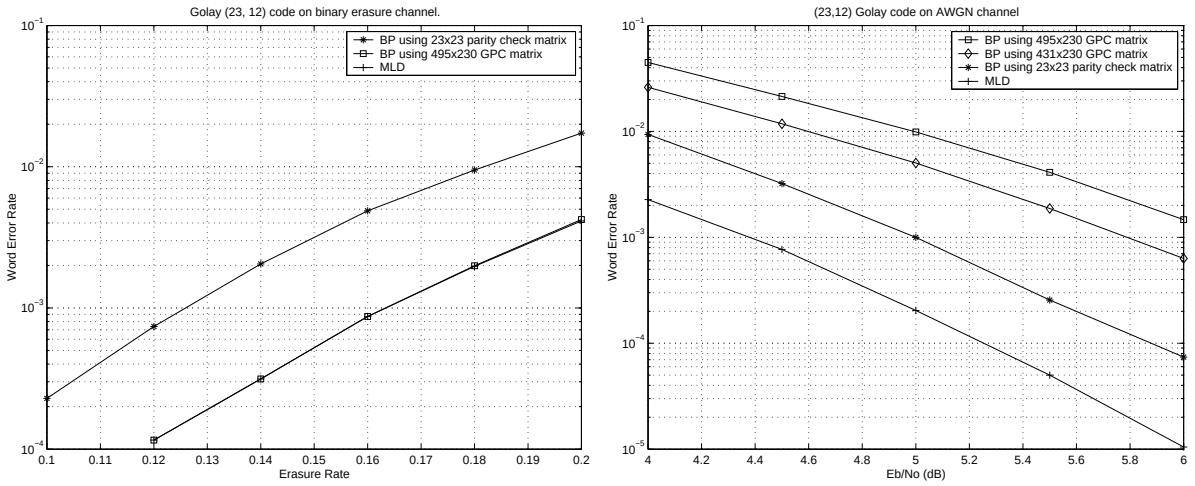

Figure 2: $(23, 12)$ Golay code on binary erasure channel (left) and AWGN channel (right).

better than it did on the input GPC matrix, and was nearly indistinguishable from the performance of maximum likelihood (ML) decoding.

These results were very encouraging, but unfortunately, using H' actually gave worse performance than using H for the AWGN channel, as shown on the right hand side of figure 2. We also tried using a 431 by 230 GPC matrix that was obtained by modifying the previously described algorithm in section 3.4 so that children were not selected to participate in a redundant parity check if they had already been selected to participate previously. This reduced the number of 4-cycles, and somewhat improved the performance, but it was still worse than that obtained by simply using BP with the original GPC matrix.

5 Empirical Results for a Euclidean Geometry Code

We also applied our method to an $(n = 255, k = 127)$, two-step majority logic decodable code based on Euclidean geometry (EG). An excellent reference to these codes is the book by Lin and Costello [10]; in their notation, the code we studied has parameters, $m = 4$, $s = 2$, and $\mu = 1$. In the geometric interpretation of this code, each symbol bit corresponds to a point, there are $21 * 255 = 5355$ lines each consisting of four points, and there are also 5355 planes consisting of four parallel lines.

This code can be represented by a redundant parity check matrix H with $M = 5355$ rows, and $N = 255$ columns, where each row has sixteen ones corresponding to the sixteen points in each plane. The factor graph corresponding to this parity check matrix is infested with four-cycles because many pairs of planes intersect on lines.

When we apply our algorithm using this input GPC matrix H, we find that the constraint sets correspond to the planes, the intersection sets correspond to the lines, and the single bit sets correspond to the points. Each constraint set has 20 children, which can be divided into five sets of four parallel (non-overlapping) lines.

We therefore obtain an output GPC matrix H' which has 255 symbol bits and 5355 auxiliary bits for a total $N' = 5610$. There are also five parity check equations obtained from each constraint set and one from each intersection set for a total $M' = 6 * 5355 = 32130$.

The 5355 rows of H' derived from the intersection sets have five ones each, while

the other 26775 rows derived from constraint sets each have four ones. Each of the 255 columns of H' corresponding to a symbol bit has 84 ones in it, while each of the 5355 columns of H' corresponding to an auxiliary bit has 21 ones in it. There are no four-cycles in the factor graph corresponding to H'.

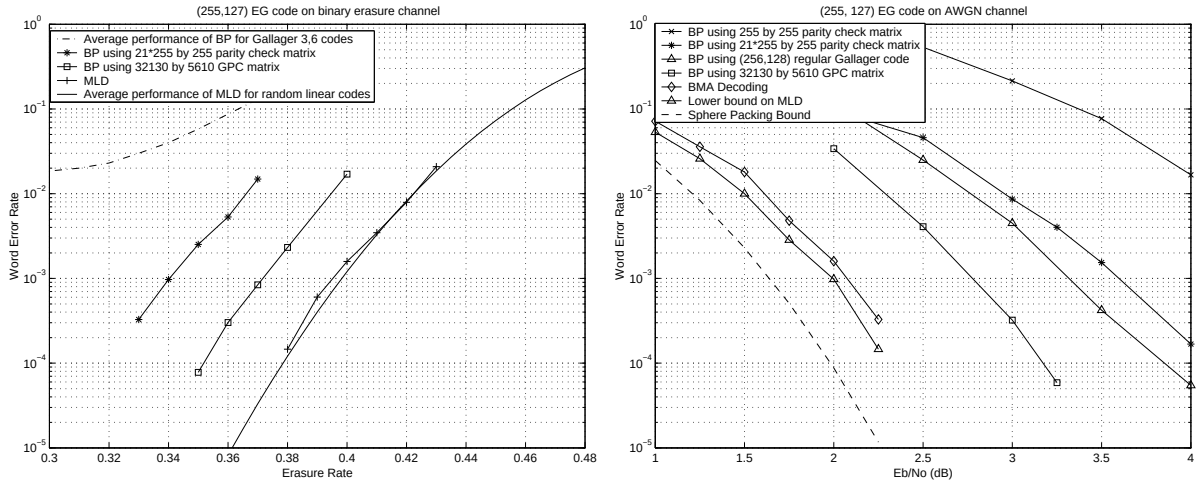

Figure 3: Performance for Euclidean Geometry ($n = 255, k = 127$) code on the BEC (left) and AWGN channel (right).

Our results for the BEC are shown in the left hand side of figure 3. To help situate the results, we have also plotted analytical results for the average performance of BP decoding of ($n = 256, k = 128$) regular Gallager LDPC codes with 6 ones per row and 3 ones per column, as well as the average performance of ML decoding of the random ensemble of ($n = 256, k = 128$) linear codes [11]. The random ensemble is obtained by taking 128 by 256 parity check matrices where each element of the parity check matrix is chosen to be 0 or 1 with equal probability [11].

With regard to the Gallager code, we note that we are comparing to a Gallager code that uses the channel a similar number of times, rather than one that has a similar decoding complexity. Although we are using a 32130 by 5610 GPC matrix for our EG code, each block transmitted still only requires 255 bits to be sent over the channel.

We see that the results using BP decoding with H are already quite good–much better than BP decoding of Gallager codes. This is a consequence of using highly redundant parity check matrices with 5355 rows instead of just 128 rows. We confirm this by comparing with BP decoding using a 255 by 255 parity check matrix–the performance (not shown) is worse than the performance for the Gallager codes.

The performance using H' was even better, but in contrast with the results for the Golay code, BP decoding using H' was still clearly distinguishable from ML decoding. ML decoding results were obtained by applying Gaussian elimination on any bits that were still un-decoded after BP decoding converged. The performance of ML decoding of this code was about the same as the average performance of ML decoding of a random linear code, which indicates that this code is quite good.

In contrast with the Golay code, for the EG code the performance of H' is much better than the performance of H on the AWGN channel. On the right hand side of figure 3, we compare the results using three different parity check matrices for the EG code; the first is a 255 by 255 matrix, the second is H (a $21 * 255$ by 255 parity check matrix) and the third is H' (a 32130 by 5610 GPC matrix). Clearly, our method does give

an improvement in this case, and only by using H' can we out-perform regular Gallager codes. (In these simulations, we used 50 BP iterations for the EG code, and 500 BP iterations for the Gallager code.)

We have also plotted the results of decoding the EG code using a "box and match algorithm" (BMA) [12], as well as a lower bound on ML decoding derived using BMA. BMA is considerably more complex than BP decoding, and is only currently feasible for rate $\frac{1}{2}$ codes with $n \leq 256$. We see that we cannot hope for further improvement in decoding performance using BP greater than about 0.8 dB at a word error rate near 10^{-4}. We have also plotted Shannon's sphere packing bound [13], which gives a lower bound on the probability of word errors for *any* code, to give an idea of how close the EG code is to optimal.

6 Discussion

6.1 Connection to Generalized Belief Propagation

The partially ordered sets obtained in our method are reminiscent of those used in generalized belief propagation (GBP) decoding [7, 8]. For the EG code that we discussed, our BP algorithm is in fact equivalent to a GBP algorithm with a region graph constructed as follows (using a factor graph corresponding to a (21*255) by 255 parity check matrix). For each of the single bit sets (points) or intersection sets (lines) form a region with the corresponding variable bits in it, and connect each line region to all the corresponding point regions. For each constraint set (plane), make five duplicate regions (each containing variable bits corresponding to the points in the plane and a factor node corresponding to the parity check) for the five ways that the plane can be divided into four parallel lines, and connect each such region to the corresponding line regions. This unusual duplication of regions is permitted because duplicating parity check constraints does not change the problem. Although this particular region graph and GBP algorithm is particularly simple and efficient, other GBP algorithms may potentially give better results, because they allow for more complicated beliefs at the intermediate regions.

6.2 Outlook

We hope that we have convinced the reader that even given a fixed code, there is an interesting problem of finding a GPC matrix representation suitable for message-passing decoding algorithms. We believe that classical codes of medium block-lengths are particularly interesting codes to consider, both because they are often known to be intrinsically good codes, and because they can have usefully redundant parity check matrix representations.

Although we have presented a particular method for transforming GPC matrices to fix ideas, we do not want to give the impression that we believe it is optimal. In fact, there are some questionable features of our method, that may cause some of the problems that were found in decoding the Golay code on the AWGN channel.

In particular, in the GPC matrices that we construct, auxiliary bits (which have no evidence from the channel) usually participate in parity checks with other auxiliary bits, rather than with symbol bits. This may induce the BP algorithm to assign incorrect beliefs for auxiliary bits in early iterations. On the BEC, this would not be a problem, because an auxiliary bit can never be assigned the wrong value; it can only stay erased.

We have also restricted ourselves to the standard BP algorithm, but other message-passing algorithms may be more appropriate for the kind of GPC matrices we are considering. For example, for multi-step majority logic decodable codes, it may be that GPC matrices and message-passing algorithms that more directly mimic multi-step majority logic decoding algorithms [10] would out-perform the approach that we have taken.

References

[1] R.G. Gallager, *Low-density parity check codes*, MIT Press, 1963.

[2] See the articles in the Special Issue on Codes and Graphs and Iterative Algorithms, *IEEE Trans. on Information Theory*, vol. 47, no. 2, February 2001.

[3] R. Lucas, M. Fossorier, Y. Kou, and S. Lin, "Iterative Decoding of One-Step Majority Logic Decodable Codes Based on Belief Propagation," *IEEE Trans. Commun.*, vol. 48, pp. 931-937, June 2000.

[4] D.J.C. MacKay, "Relationships Between Sparse Graph Codes," *(unpublished) proceedings of IBIS 2000*, Japan, available online at http://www.inference.phy.cam.ac.uk/mackay/abstracts/ibis.html.

[5] F.R. Kschischang, B.J. Frey, and H.-A. Loeliger, "Factor Graphs and the Sum-Product Algorithm," *IEEE Trans. on Information Theory*, vol. 47, no. 2, pp. 585-598, February 2001.

[6] G.D. Forney, Jr., "Codes on Graphs: Normal Realizations," *IEEE Trans. on Information Theory*, vol. 47, no. 2, pp. 520-548, February 2001.

[7] J.S. Yedidia, W.T. Freeman, and Y. Weiss, "Constructing Free Energy Approximations and Generalized Belief Propagation Algorithms," MERL Technical Report TR2002-35, 2002, available online at htp://www.merl.com/papers/TR2002-35/.

[8] R.J. McEliece and M. Yildirim, "Belief Propagation on Partially Ordered Sets," to appear in the IMA volume *Mathematical Systems Theory in Biology, Communications, Computation, and Finance*, D. Gilliam and J. Rosenthal, eds. 2002, available online at http://www.ee.caltech.edu/EE/Faculty/rjm/.

[9] R.P. Stanley, *Enumerative Combinatorics, vol. 1*, Cambridge University Press, 1983.

[10] S. Lin and D.J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, 1983.

[11] C. Di, D. Proietti, I.E. Teletar, T.J. Richardson, and R.L. Urbanke, "Finite-Length Analysis of Low-Density Parity-Check Codes on the Binary Erasure Channel," *IEEE Trans. on Information Theory*, vol. 48, no. 6, pp. 1570-1579, June 2002.

[12] A. Valembois and M. Fossorier, "Box and Match Techniques Applied to Soft-Decision Decoding," *Proc. IEEE Symp. on Inform. Theory*, Lausanne, Switzerland, p. 143, 2002.

[13] C. Shannon, "Probability of Error for Optimal Codes in a Gaussian Channel," *Bell Syst. Tech. J.*, vol. 38, pp. 622-656, 1959.