

Graphical Representations of Error-Correcting Codes

Henry D. Pfister

September 16th, 2017

1 Introduction

Consider the $(7, 4)$ binary Hamming code \mathcal{C} defined by the parity-check matrix

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \quad (1)$$

We can associate this parity-check matrix with a *Tanner graph* G by representing each code bit as a circle (i.e., variable node) and each parity-check equation as a square (i.e., parity node). If a code bit participates in a parity-check, then the graph also includes an edge between those two vertices. In this case, the graph associated with (1) is shown below in Figure 1.

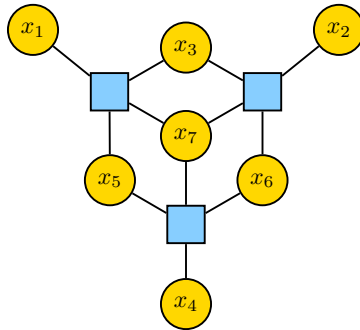


Figure 1: Tanner graph of the $(7, 4)$ Hamming code.

The Tanner graph defines the code precisely and also exposes a number of interesting properties of the code. In particular, the following decomposition property is related to the conditional independence property of general factor graphs. Consider fixing the values of a small set of bits and absorbing the fixed values into the parity nodes. If this causes the graph to separate into disjoint components, then many operations can be implemented using a divide and conquer approach that reduces complexity. In the above graph, consider fixing $x_3 = x_5 = x_7 = 0$ and absorbing the 0's into their adjacent parity nodes. This removes these variables nodes and their adjacent edges. Thus, the graph separates into a component containing x_1 and a component containing x_2, x_4, x_6 . Using this, one can enumerate all valid codewords by iterating over all possible values of x_3, x_5, x_7 and, for each value, listing the valid patterns of the variables in each component.

The Tanner graph also provides an intuitive suboptimal decoding strategy. A parity-check equation asserts that a set of involved variables must sum to 0. This allows one to compute the value of a missing variable if the values of all other variables in that equation are known. Similarly, a parity node in the Tanner graph implies that the value of the adjacent variable nodes must sum to 0. If one of the adjacent variables is unknown and all others are known, then the missing value may be computed. The “peeling decoder” applies this operation iteratively until there is no parity node with exactly one unknown adjacent variable.

Exercise 1.1. Try the peeling decoder on received vector $(0, 1, 0, 1, 0, ?, ?)$. What codeword is recovered?

2 The Tanner Graph and the Peeling Decoder

The simplest example of iterative decoding is the peeling decoder introduced for the binary erasure channel (BEC) by Luby et al. [1] (see also [2, pp. 117–121, 134–136]). This decoder is based on the parity-check matrix of the code and can be applied to an arbitrary linear code¹. Let H be the parity-check matrix of an (n, k) linear code.

Definition 2.1. The *Tanner graph* G of an $m \times n$ parity-check matrix H is a bipartite graph with one vertex for each code symbol and one vertex for each parity check. For each non-zero entry in H , the graph contains an edge that connects the variable node associated with the column to the check node associated with the row. Mathematically, we have $G = (V \cup C, E)$ with

$$V = \{1, 2, \dots, n\} \quad C = \{1, 2, \dots, m\} \quad E = \{(i, j) \in C \times V \mid H_{i,j} \neq 0\}.$$

Algorithm 2.2 (Peeling Decoder). *Iteratively remove known bits from the graph as follows:*

1. Initialize the variables x_1, \dots, x_n to ? and the variables y_1, \dots, y_m to zero.
2. For each non-erased code symbol, let $j \in V$ be its index and set variable x_j to the known value.
3. If there is a degree-1 check node, let $i \in C$ be its index, $j \in V$ be the index of the adjacent variable node, and set $x_j = H_{ij}^{-1}y_i$.
4. If the graph contains a variable node whose value is known (i.e., $x_j \neq ?$), let $j \in V$ be its index and
 - (a) for all i such that $(i, j) \in E$, update $y_i = y_i - H_{ij}x_j$
 - (b) remove bit j and all adjacent edges from the graph (i.e., $V \leftarrow V \setminus j$, $E \leftarrow E \setminus \{(i, j') \in E \mid j = j'\}$)
 - (c) Goto step 3
5. When the algorithm reaches this point, either decoding is successful and $x_j \neq ?$ for all $j \in V$ or the decoder is stuck in a configuration where there are no degree-1 check nodes and the graph contains only variable nodes whose values are unknown.

Exercise 2.3. Implement the peeling decoder in some programming language (e.g., Matlab) and test it using the $(7, 4)$ Hamming code with cyclic parity-check matrix

$$H = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}. \quad (2)$$

Use Monte Carlo simulation to estimate the bit erasure probability (i.e., the expected fraction of erasures) after decoding for a binary erasure channel with erasure probability $\epsilon = 0.15$.

This Hamming code can also be defined using an overcomplete parity-check matrix

$$H' = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}. \quad (3)$$

whose rows have linear dependencies. Although the code is the same, the peeling decoder based on this matrix is more powerful. Use Monte Carlo simulation to estimate the bit erasure probability (i.e., the expected fraction of erasures) after decoding for this decoder for the erasure probabilities $\epsilon \in \{0.1, 0.2, 0.3\}$. Compare the performance of the peeling decoder for H' with with the performance for H .

¹While our primary interest is binary codes, this description is valid for codes defined over any field.

For channels such as the binary-input AWGN channel, the received value contains reliability information for the received symbol. Peeling decoders are not well-suited to these types of channels, however, because they work best when each step is correct with high probability. But, often there is no symbol that can be locally determined with low error probability. In this case, iterative belief-propagation decoding is the favored solution. See the Appendix A for more details.

3 Static Analysis of the Peeling Decoder

For any parity-check matrix, the performance of the peeling decoder is completely determined by the stopping sets of its Tanner graph.

Definition 3.1 (Stopping Set). A *stopping set* (ss) S is a subset of variable nodes that, when initially erased, prevents the peeling decoder from recovering the value of any variable node in S . Mathematically, S is a subset of variable nodes whose induced subgraph has no check nodes with degree 1 (i.e., all neighbors of S must be connected to S at least twice). For example, to satisfy the parity-check equations, the subgraph induced by the support set of any codeword cannot have any degree-1 check nodes and therefore must be a stopping set. By convention, the empty set is considered a stopping set.

Lemma 3.2 (Lemma 3.140 in [2]). *Stopping sets have the following properties:*

1. If S_1 and S_2 are stopping sets, then $S_1 \cup S_2$ is a ss.
2. Each subset W of V contains a unique maximum ss.
3. If $W \subseteq V$ is the set of initially erased variable nodes, then the peeling decoder will recover all variables except those in the unique maximum ss contained in W .

Proof. For the first claim, we observe that, if a check node c is a neighbor of $S_1 \cup S_2$, then it must be a neighbor of either S_1 or S_2 . Since S_1 and S_2 are both stopping sets, either S_1 or S_2 must be connected to c at least twice. Therefore, $S_1 \cup S_2$ is connected to c at least twice.

For the second claim, we define U to be the union of all stopping sets contained in W . The first claim implies that U is indeed a ss. It is maximal because any ss S contained in W (i.e., $S \subseteq W$) is also contained in U (i.e., $S \subseteq U$).

For the third claim, we first observe that, if W contains a stopping set S (i.e., $S \subseteq W$), then the peeling decoder will not recover any variable in S . This follows from the definition of a ss and is based on the fact that even revealing all variables $V \setminus S$ does not allow the peeling decoder to recover any variable in S . This also implies that the peeling decoder will not recover any variables in U , the unique maximum ss contained in W . Since U is the maximum ss in W , for all non-empty $T \subseteq W \setminus U$, the set $T \cup U$ is not a stopping set and therefore will be reduced by one step of the peeling decoder. This implies that all variables in $W \setminus U$ will be recovered by the peeling decoder. \square

Definition 3.3. A ss is *minimal* if the only stopping set it contains is the empty set.

For a particular parity-check matrix, let $A_{ss}(h)$ be the number of stopping sets of weight h and $\hat{A}_{ss}(h)$ be the number of minimal stopping sets of weight h . For the BEC, one can use $\hat{A}_{ss}(h)$ to bound the probability that the peeling decoder does not successfully recover all symbols. Since a decoding failure occurs only if the set of channel erasures contain some minimal ss, one can show (see Appendix B) that

$$P_B(\epsilon) \leq \sum_{h=1}^n \hat{A}_{ss}(h) \epsilon^h \quad (4)$$

$$P_b(\epsilon) \leq \sum_{h=1}^n \hat{A}_{ss}(h) \frac{h}{n} \epsilon^h. \quad (5)$$

Question 3.4. *Can the exact decoding failure probability be computed solely in terms of the minimal stopping-weight enumerator $\hat{A}_{ss}(h)$? [Hint: Try finding two codes with the same $\hat{A}_{ss}(h)$ and different decoding failure probabilities.]*

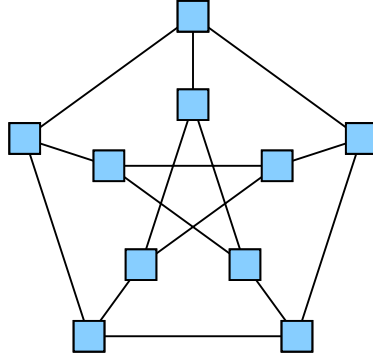


Figure 2: The Petersen graph defines a $(15, 6, 5)$ cycle code.

Exercise 3.5. Use your program from Exercise 2.3 to find all stopping sets for the peeling decoder based on (2) and list the minimal stopping sets. Simulate this code and decoder on the BEC for erasure probabilities $\epsilon \in \{0.1, 0.2, 0.3\}$ and compare the resulting block erasure rate to the expression given by (4).

Remark 3.6. The idea of peeling also applies to more general constraint satisfaction problems on graphs (e.g., Sudoku). The idea is to fix a variable if it has a unique satisfying value determined by its local constraints. This process continues until there is no such variable. A partially determined solution is a stopping set precisely if there is no such variable.

4 Important Generalizations

4.1 Cycle Codes

A different graphical representation starts with an arbitrary graph $G = (V, E)$ and uses it to define a binary linear code. The idea is to associate code bits with the edges of the graph and parity-checks with the vertices. The codewords are edge indicator vectors (i.e., vectors $\{0, 1\}^{|E|}$) for “special” subsets of edges. We say that a subset of edges is special if its induced subgraph (i.e., the edges along with all adjacent vertices) has no vertices of odd degree. Such a code is called the *cycle code* of the graph [3, pp. 136-138]. It is easy to verify that the weight of the non-zero codeword with minimum weight is equal to the girth of the graph.

The primary reason to consider this construction is that there are many well-known graph constructions that also lead to good codes. For example, one can try small extremal graphs (such as the Petersen graph) or infinite families of graphs (such as deterministic constructions of Ramanujan expander graphs).

To construct a Tanner graph for a cycle code, one associates the edges of the original graph with variable nodes in the Tanner graph and vertices of the original graph with parity nodes in the Tanner graph. If an edge e in the original graph (associated with variable-node v_b in the Tanner graph) is attached to vertex v in the original graph (associated with parity-node v_p in the Tanner graph), then there is an edge in the Tanner graph between a variable-node v_b and parity-node v_p .

To see that this works, we note that any codeword in the cycle code is associated with a subset of edges in the original graph whose induced subgraph U has no vertices of odd degree. In the code defined by the Tanner graph, this subset of edges is associated with a subset of variable nodes (i.e., the set of variable nodes equal to 1). The number of 1’s adjacent to a parity-node in the Tanner graph is equal to the degree of its associated vertex in U . Thus, all parity checks are satisfied precisely if U has no vertices of odd degree. From this equivalence, it follows that the cycle code is linear.

In the original graph, each edge connects exactly two vertices and, thus, the degree of any variable node in the Tanner graph is 2. In fact, any Tanner graph where all variable nodes have degree 2 defines a cycle code. This graph associated with the implied cycle code can be constructed by removing each degree-2 vertex from the Tanner graph and connecting the two dangling edges.

Exercise 4.1. Write down the parity-check matrix of the cycle code defined by the Petersen graph in Figure 2 and use linear algebra (either by hand or via computer) to compute the code dimension.

4.2 Generalized Code Constraints

A parity node in a standard Tanner graph represents the (linear) constraint that the values of the adjacent variable nodes must sum to 0. A natural generalization is to allow a parity node to represent a more complicated linear constraint. For example, one could use generalized constraint nodes of degree-7 that require the adjacent variable nodes to equal a codeword in the (7, 4) Hamming code. In this case, the local code protecting adjacent variables is much stronger (e.g., it can correct all single errors).

One complication with this approach is that the code typically depends on the ordering of the edges attached to each generalized constraint. This can be handled by assigning colors to edges and using these colors to identify which socket of the constraint node should be attached to that edge.

The idea of a peeling decoder can also be extended to generalized constraints. For an erasure channel, a generalized constraint may be peeled off if there is a unique satisfying assignment that matches the adjacent variable nodes that are not erased. Once again, such a decoder will continue until it becomes stuck in a stopping set from which no progress can be made.

For an error channel (e.g., the binary symmetric channel), the peeling decoder strategy is often used when the generalized constraints are t -error correcting linear codes. In this case, one can locally decode each generalized constraint in a sequential fashion. If there are t or fewer errors in the values of the adjacent variable nodes, then the decoding is successful and the correct values are found. If not, the decoder may fail to decode (e.g., there is no valid codeword within distance t of the current pattern) or it may miscorrect (e.g., there are $\geq t + 1$ errors and the decoder returns an incorrect codeword within distance t). Due to the possibility of miscorrection, the decoding dynamics for errors are more complicated than in the erasure case. For errors, a stopping set can be defined as an error pattern where each generalized constraint is either satisfied or the local decoding operation fails (e.g., leaving the current pattern unchanged). However, the decoder is not guaranteed to terminate in a stopping set. Instead, it may continue in a periodic cycle of miscorrections.

5 Trellis Diagrams and Shortest Path Algorithms

A *trellis diagram* is a labeled directed graph where the vertices are grouped into *layers* (e.g., labeled by $\{0, 1, \dots, n\}$) and the edges are only allowed to connect vertices in adjacent layers. Such a graph can represent a code by providing a bijective mapping between codewords and directed paths from layer 0 to layer n . In this case, each path contains n edges and the edge labels denote the n bits in the codeword. The set of edges connecting two adjacent layers is called a *trellis section*.

For an (n, k) linear code defined by a canonical generator matrix $G = [I \ P]$, there is a natural trellis with 2^k states. For an (n, k) linear code defined by a canonical parity-check matrix $H = [P^T \ I]$, there is a natural trellis with 2^{n-k} states. For the (7, 4) Hamming code defined by

$$P = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix},$$

the trellis diagram associated with the parity-check matrix is shown in Figure 3.

Assume that a random codeword vector \underline{X} is transmitted through a memoryless channel and \underline{Y} is the random output vector. If the realization \underline{y} is observed, then the optimal decoder chooses the codeword $\hat{\underline{x}}$ that maximizes $\Pr(\underline{X} = \underline{x} | \underline{Y} = \underline{y})$ over $\underline{x} \in \mathcal{C}$. If the codeword vector is chosen uniformly from the code, then one finds that

$$\Pr(\underline{X} = \underline{x} | \underline{Y} = \underline{y}) = \frac{\Pr(\underline{X} = \underline{x}, \underline{Y} = \underline{y})}{\Pr(\underline{Y} = \underline{y})} = \frac{\Pr(\underline{X} = \underline{x}) \Pr(\underline{Y} = \underline{y} | \underline{X} = \underline{x})}{\Pr(\underline{Y} = \underline{y})} = a \Pr(\underline{Y} = \underline{y} | \underline{X} = \underline{x}),$$

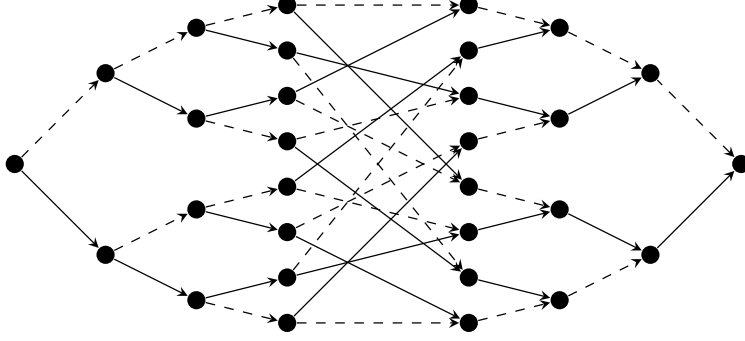


Figure 3: Trellis diagram for (7,4) Hamming code where dashed edges represent 1's.

where a is a constant that is independent of \underline{x} . For a memoryless channel, each channel use is independent and we find further that

$$\Pr(\underline{Y} = \underline{y} | \underline{X} = \underline{x}) = \prod_{i=1}^n \Pr(Y_i = y_i | X_i = x_i).$$

The key obstacle to implementing an optimal decoder is the complexity of maximizing this quantity over all possible codewords. This decoding problem can be setup as a shortest path problem on a weighted trellis diagram. To do this, we assign a non-negative weight (or length) to each edge in the trellis. For an edge in the i -th section labeled by bit x , the non-negative weight is chosen to be

$$w_i(x) \triangleq -\ln \Pr(Y_i = y_i | X_i = x).$$

The weight (or length) of a directed path $\underline{x} = (x_1, x_2, \dots, x_n)$ is also assumed to be additive so that

$$w(\underline{x}) \triangleq \sum_{i=1}^n w_i(x_i).$$

It follows that

$$w(\underline{x}) = -\sum_{i=1}^n \ln \Pr(Y_i = y_i | X_i = x_i) = -\ln \Pr(\underline{Y} = \underline{y} | \underline{X} = \underline{x}).$$

Thus, minimizing $w(\underline{x})$ over all codewords is equivalent to the optimal decoding problem of maximizing $\Pr(\underline{Y} = \underline{y} | \underline{X} = \underline{x})$ over all codewords. This is also equivalent to finding the shortest path from layer 0 to layer n . There are a variety of ways to find shortest paths in graphs. For weighted trellis diagrams, the Viterbi algorithm is quite popular and provides a nice example of dynamic programming [4].

Now, we discuss two simple constructions of trellis diagrams for block codes [5]. In many cases, the complexity can be reduced by optimizing the trellis [6].

For the generator matrix construction, the trellis consists of $n + 1$ layers of vertices indexed by $\{0, 1, \dots, n\}$. In layer 0, there is a single state that is unlabeled. For $1 \leq i \leq k$, layer i contains 2^i vertices indexed by length i binary strings. These binary strings represent the first i bits in the codeword. For $\underline{\ell} \in \{0, 1\}^i$ and $\underline{\ell}' \in \{0, 1\}^{i+1}$, there is a directed edge from state $\underline{\ell}$ in layer i to state $\underline{\ell}'$ in layer $i + 1$ if $\ell_j = \ell'_j$ for $j = 1, 2, \dots, i$. Such an edge is labeled by the bit ℓ'_{i+1} . For $i > k$, layer i contains 2^k vertices indexed by length k binary strings. These binary strings represent the first k bits in the codeword. For $\underline{\ell} \in \{0, 1\}^k$, there is a single directed edge from state $\underline{\ell}$ in layer $i - 1$ to state $\underline{\ell}$ in layer i that is labeled by the i -th bit of the codeword whose first k bits equal $(\ell_1, \ell_2, \dots, \ell_k)$. Using this construction, there is a bijective mapping between the codewords and directed paths from layer 0 to layer n .

For the parity-check matrix construction, the trellis consists of $n + 1$ layers of vertices indexed by $\{0, 1, \dots, n\}$. For simplicity, we describe a trellis that, in each layer, has 2^{n-k} states labeled by the binary vectors in $\{0, 1\}^{n-k}$. Many of these states are unnecessary and can be removed without affecting the decoder. For $i \in \{1, 2, \dots, n\}$ and $\underline{\ell}, \underline{\ell}' \in \{0, 1\}^{n-k}$, there is a directed edge from state $\underline{\ell}$ in layer $i - 1$

to state $\underline{\ell}'$ in layer i if $\underline{\ell}' = \underline{\ell}$ or if $\underline{\ell}' = \underline{\ell} + \underline{h}_i$ where \underline{h}_i is the i -th column of the parity-check matrix H . In the first case, the edge is labeled by 0 and, in the second case, the edge is labeled by 1. The state in layer i represents the partial syndrome vector

$$s_i(\underline{x}) = \sum_{j=1}^i x_j \underline{h}_j$$

of all paths that enter that state. Since we know that the total syndrome $\sum_{i=1}^n x_i \underline{h}_i$ of any codeword must be $\underline{0}$, there is a bijective mapping between the set of codewords and the set of directed paths that start at state $\underline{0}$ in layer 0 and end at state $\underline{0}$ in layer n . Any state that does not lie on one of these paths may be removed from the trellis without affecting the bijective property or the decoder.

6 Historical Notes

Low-density parity-check codes and iterative decoding were introduced in 1960 by Gallager [7]. The idea of carefully designed graphs and generalized code constraints was introduced in 1981 by Tanner [8]. The importance of these ideas was not widely understood until their rediscovery in 1995 by MacKay [9] after the introduction of Turbo Codes [10]. Based on these results, a unified approach to inference problems on factor graphs was introduced in 2001 [11]. Around same time, a class of capacity-achieving codes was introduced for the erasure channel [1] and similar codes were optimized for general channels [12].

References

- [1] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Efficient erasure correcting codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 569–584, Feb. 2001.
- [2] T. J. Richardson and R. L. Urbanke, *Modern Coding Theory*. New York, NY: Cambridge University Press, 2008.
- [3] W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*. Cambridge, MA, USA: The M.I.T. Press, 2nd ed., 1972.
- [4] G. D. Forney, Jr., "The Viterbi algorithm," *Proc. of the IEEE*, vol. 61, pp. 268–277, March 1973.
- [5] J. K. Wolf, "Efficient maximum likelihood decoding of linear block codes using a trellis," *IEEE Trans. Inform. Theory*, vol. 24, pp. 76–80, Jan. 1978.
- [6] D. J. Muder, "Minimal trellises for block codes," *IEEE Trans. Inform. Theory*, vol. 34, no. 5, pp. 1049–1053, 1988.
- [7] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA, USA: The M.I.T. Press, 1963.
- [8] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. 27, pp. 533–547, Sept. 1981.
- [9] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, pp. 399–431, March 1999.
- [10] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *Proc. IEEE Int. Conf. Commun.*, vol. 2, (Geneva, Switzerland), pp. 1064–1070, IEEE, May 1993.
- [11] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, vol. 47, pp. 498–519, Feb. 2001.
- [12] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 619–637, Feb. 2001.

A General Channels and Belief-Propagation Decoding

For channels such as the binary-input AWGN channel, the received value contains reliability information for the received symbol. Peeling decoders are not well-suited to these types of channels, however, because

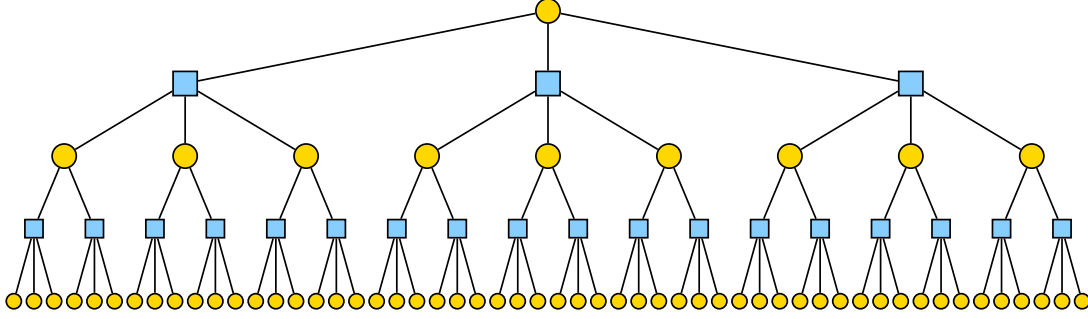


Figure 4: Consider the progression of messages directed upwards in this Tanner graph. Since the graph is a tree, one can think of these messages as encapsulating all the information in the subtree directed downwards from the edge. This picture naturally motivates the “leave one out” rule of belief-propagation decoding. In fact, this rule is also well-motivated in general graphs but for more complicated reasons.

they work best when each step is correct with high probability. But, often there is no symbol that can be locally determined with low error probability.

In this case, iterative belief-propagation decoding is the favored solution. The idea is to have variable and parity nodes pass messages to each other along edges in the Tanner graph. These messages communicate the conditional distribution of certain subsets of local observations given the value of the variable. For binary variables, the message is typically represented by the log-likelihood ratio (LLR) $L = \ln \frac{q}{p}$, where q (resp. p) is the observation probabilities given that the adjacent variable takes the value 0 (resp. 1). The message updates are based on the assumption that all input messages to a node are accurate statistically-independent estimates of this probability.

For a binary variable node of degree- d , the variable X takes the value 0 or 1 with equal probability and the input messages represent independent observations Y_1, \dots, Y_d given that value. We also assume there is a local observation Y_0 associated with the direct observation of X through the channel. Since the output message on the j -th edge does not use the j -th observation, it follows that

$$\begin{aligned}
 L_j &= \ln \frac{\Pr(Y_0, \dots, Y_{j-1}, Y_{j+1}, \dots, Y_d | X = 0)}{\Pr(Y_0, \dots, Y_{j-1}, Y_{j+1}, \dots, Y_d | X = 1)} \\
 &= \prod_{i=0, i \neq j}^d \ln \frac{\Pr(Y_i | X = 0)}{\Pr(Y_i | X = 1)} \\
 &= \sum_{i=0, i \neq j}^d L'_i.
 \end{aligned}$$

For a binary code, a parity node represents the constraint that the values of all adjacent variable nodes sum to 0. After choosing one edge to be the output edge, the output message can be computed as the soft-XOR of the other input messages. This represents the distribution of the XOR of $d - 1$ random bits whose distributions are defined the input messages LLRs. The formula is given in Figure 5 and we do not derive it here.

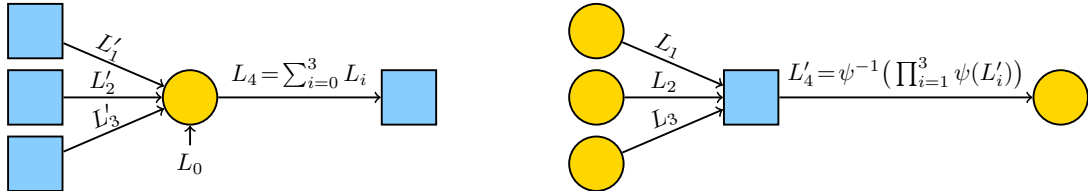


Figure 5: Local update rules for the belief-propagation decoding of a binary linear code in the LLR domain, where $\psi(x) = \tanh\left(\frac{x}{2}\right)$.

A key element of the update rule is the “leave one out” property whereby the output message on edge e is computed using the input messages on all other edges except edge e . Thus, the input message on edge e does not affect the output message on edge e . Using this rule, the belief-propagation decoder is optimal if the Tanner graph is a tree. The reason for this is illustrated in Figure 4.

B Union Bound for Erasure Channels

Consider a length- n code on the binary erasure channel with erasure probability ϵ . Assume the decoder performance is independent of the transmitted codeword. For example, this occurs for linear codes with both peeling decoding and optimal decoding. In this case, the block erasure rate is given by

$$P_B = \sum_{T \subseteq \{1,2,\dots,n\}} \epsilon^{|T|} (1-\epsilon)^{|T^c|} \mathbb{I}(\text{decoder fails for } T),$$

where T denotes the set of erasures and $\mathbb{I}(\text{statement})=1$ if the statement is true and 0 if it is false. From the previous sections, we know that the peeling decoder fails if and only if the erasure pattern contains a minimal stopping set. Thus, we let \mathcal{S} be the set of minimal stopping sets and observe that

$$\mathbb{I}(\text{peeling fails for } T) = \mathbb{I}(T \in \mathcal{S}) \leq \sum_{S \in \mathcal{S}} \mathbb{I}(T \subseteq S).$$

From this, we see that

$$\begin{aligned} P_B &\leq \sum_{T \subseteq \{1,2,\dots,n\}} \epsilon^{|T|} (1-\epsilon)^{|T^c|} \sum_{S \in \mathcal{S}} \mathbb{I}(T \subseteq S) \\ &= \sum_{S \in \mathcal{S}} \sum_{T \subseteq \{1,2,\dots,n\}} \mathbb{I}(T \subseteq S) \epsilon^{|T|} (1-\epsilon)^{n-|T|} \\ &= \sum_{S \in \mathcal{S}} \epsilon^{|S|} \sum_{R \subseteq \{1,2,\dots,n\} \setminus S} \epsilon^{|R|} (1-\epsilon)^{(n-|S|)-|R|} \\ &= \sum_{S \in \mathcal{S}} \epsilon^{|S|} = \sum_{h=1}^n \epsilon^h |\{S \in \mathcal{S} \mid |S| = h\}| \\ &= \sum_{h=1}^n \hat{A}_{ss}(h) \epsilon^h. \end{aligned}$$

To upper bound the bit erasure rate, we proceed similarly with

$$P_b = \sum_{T \subseteq \{1,2,\dots,n\}} \epsilon^{|T|} (1-\epsilon)^{|T^c|} \frac{1}{n} \sum_{i=1}^n \mathbb{I}(\text{peeling fails for bit } i \text{ with } T).$$

The peeling decoder fails to recover a bit if and only if that bit lies in the union of minimal stopping sets covered by the erasure pattern T . Using this fact, we see that

$$\begin{aligned} \frac{1}{n} \sum_{i=1}^n \mathbb{I}(\text{peeling fails for bit } i \text{ with } T) &= \frac{1}{n} \sum_{i=1}^n \mathbb{I}(i \in T \text{ and } T \in \mathcal{S}) \\ &= \frac{|T|}{n} \mathbb{I}(T \in \mathcal{S}). \\ &\leq \frac{|T|}{n} \sum_{S \in \mathcal{S}} \mathbb{I}(T \subseteq S). \end{aligned}$$

Thus, using the same arguments as above, one finds that

$$P_b \leq \sum_{h=1}^n \hat{A}_{ss}(h) \frac{h}{n} \epsilon^h.$$