ENEE 739C: Advanced Topics in Signal Processing: Coding Theory
Instructor: Alexander Barg

Lecture 9 (**draft;** 11/07/03). Codes defined on graphs. The problem of attaining capacity. Iterative decoding.

http://www.enee.umd.edu/~abarg/ENEE739C/course.html

In this part we analyze code families and their decoding procedures motivated partly by the graph-theoretic representation of product codes of Lecture 8 and partly by decoding of linear block code on a trellis. The ideas explored here shift the general point of view from the study of parameters and structure-motivated decoding algorithms to iterative refinement of the decision. Another motivation is that of studying thresholds of codes versus the code rate instead of first computing the distance and weight distribution. Finally, the attention is shifted from block error rate to bit (symbol) error rate as a measure of quality of transmission.

This view of codes crystallized through the study of turbo decoding and message passing decoding of low-density parity-check codes in the last decade. The material covered here is discussed in books [2, 7, 8] and in more detail in [14, 10]. Codes in this lecture are binary linear.

DECODERS OF BINARY CODES. Our first goal is to motivate and define basic iterative algorithms for codes on graphs. The main idea of these algorithms is to exploit the locally sparse structure of parity checks.

The ML decoder minimizes the error probability if codewords are equiprobable. In general, the error probability is minimum under maximum aposteriori probability (block APP) decoding which outputs $\mathbf{c} = \arg\max_{\mathbf{c}' \in \mathcal{C}} \Pr[\mathbf{c}'|\mathbf{y}]$ where $\mathbf{y}$ is the received word. Since

$$P(\mathbf{c}|\mathbf{y}) = \frac{W^n(\mathbf{y}|\mathbf{c})P(\mathbf{c})}{P(\mathbf{y})},$$

for equiprobable messages these two decoders are the same.

It is possible to take another point of view, that of MAP (*maximum aposteriori probability*) bit decoder. For $a = 0, 1$ it computes

$$P_i(a|\mathbf{y}) = \sum_{\substack{\mathbf{c}\in\mathcal{C} \\ c_i = a}} \Pr[\mathbf{c}|\mathbf{y}] \tag{1}$$

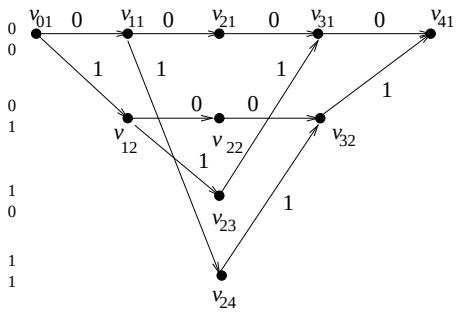
and outputs the bit whose posterior is larger. Note that it may happen that the outcome of this decoding is not a valid codeword.

The motivation behind studying this decoding is as follows. ML and block APP decoding procedures provide very good performance but are in general very difficult to implement. The most successful setting for these algorithms is that of syndrome trellises. ML decoding on a trellis is performed by the Viterbi algorithm which we discuss below. MAP decoding on a trellis can be implemented by the so-called BCJR (Bahl-Cocke-Jelinek-Raviv) procedure (the two-way or backward-forward algorithm). It has been observed that these algorithms can be exported into a more general setting of Tanner graphs like the graph of a product code in the previous lecture. The Viterbi algorithm then becomes "min-sum" algorithm and the BCJR the "sum-product" algorithm. These algorithms are performed in iterations until some stopping condition is satisfied. Codes with a low density of parity checks (in which few coordinates enter each check equation) are particularly suitable for these iterative procedures. Iterative decoding of LDPC codes for the erasure channels has been shown to reach capacity (reliable communication with iterative decoding is possible at any rate below capacity). On other channels theoretical analysis is incomplete.

Recall briefly the Viterbi algorithm and the two-way, or BCJR algorithm.

THE VITERBI ALGORITHM. The trellis of the code has 2^{n-k} levels which correspond to syndrome values and $n + 1$ states. Let us number the levels according to a lexicographic order of the syndromes. An edge between vertices $v_{i,j}$ and $v_{i+1,\ell}$ is drawn if the value of the syndrome i changes (from j) to ℓ upon adding $a\mathbf{h}_i$ (here $\mathbf{h}_i$ is the ith column of $\mathbf{H}$ and $a = 0, 1$). Paths not leading to the state $v_{n,0}$ are purged from the lattice. The obtained (BCJR or Wolf) trellis is minimal.

Suppose that the parity-check matrix of a $[4, 2, 2]$ code has the form $H = \begin{bmatrix} 0\,1\,1\,0 \\ 1\,1\,0\,1 \end{bmatrix}$. The trellis has the following form:



Let $\mathbf{y} = (y_1, \ldots, y_n)$ be the received vector. Compute the likelihoods $w_{i,a} = -\log W(y_i|a)$, where W is the probability (distribution) given by a binary-input memoryless channel $\mathscr{W}$. For a given $i \geq 1$ and $1 \leq j \leq S_i$ find

$$r_{i,j} = \min_{a,\ell}(r_{i-1,\ell} + w_{i,a})$$

The optimal decision corresponds to the minimum-weight path in the trellis.

Obviously, the Viterbi algorithm performs ML decoding. A complexity saving is explained by the observation that common parts of different codewords are stored on one and the same edge in the trellis. The probability computation for these parts is performed only once (taken outside the brackets).

The BCJR algorithm does essentially a very similar computation except its object is to perform bit MAP decoding. Our goal is to find which is the two letters 0 or 1 is more likely to have been transmitted in each coordinate. Recall from (1) that we need to compute a collection of marginal distributions. We have

$$P_i(a|\mathbf{y}) = \sum_{\substack{\mathbf{c}\in\mathcal{C}\\ c_i=a}} \Pr[\mathbf{c}|\mathbf{y}] = \sum \frac{P(\mathbf{c},\mathbf{y})}{P(\mathbf{y})} = \sum \frac{W^n(\mathbf{y}|\mathbf{c})P(\mathbf{c})}{P(\mathbf{y})},$$

where $P(\mathbf{y}) = \sum_{\mathbf{c}\in\mathcal{C}} P(\mathbf{y}|\mathbf{c})P(\mathbf{c})$. Assuming that messages are equiprobable[1], we notice that the probability $P_i(a|\mathbf{y})$ is proportional to

$$\sum_{\substack{\mathbf{c}\in\mathcal{C}\\ c_i=a}} W^n(\mathbf{y}|\mathbf{c}) = \sum \prod_{m\neq i} W^n(y_m|c_m).$$

Thus the sought probability is proportional to

$$P_i(a|\mathbf{y}) \propto \sum \prod_{m=1}^{i-1} W(y_m|c_m) \sum \prod_{m=i+1}^{n} W(y_m|c_m) \tag{2}$$

with an obvious breakdown into two parts. The algorithm relies on a decomposition of this form to compute the posterior probabilities of branches (edges) recursively, starting from the state s_0 (forward flow) and s_n (backward flow). Let us associate partial results $\alpha_{i,j}, \beta_{i,j}$ with states and branches of the trellis as follows: the cost $\alpha_{i,j}$ is associated with the branch (i,j) and the vertex to its right; the cost $\beta_{i,j}$ is associated with the branch (i,j) and the vertex to its left. Consider the forward computation. We need to describe the update rule for finding the values $\alpha_{i+1,*}$ given the values $\alpha_{i,*}$. That rule is given by

$$\alpha_{i,j} = \sum_j \alpha_{i-1,j} W(y_i|c),$$

where the sum is over the branches incident to $v_{i,j}$ and for a given branch, the value c is the bit associated with it. The backward computation is organized analogously. Once both flows are complete, every branch will have two numbers associated with it, namely $\beta_{i,j}, \alpha_{i,j}$. These numbers will correspond to the first and the second parts of the decomposition (2), respectively. Thus we can find the (quantity proportional to the) marginal distribution of every branch in the trellis and therefore also find the greater of the marginals of 0 and 1 in each of the n positions of the transmission.

[1]In the context of sum-product, or message passing iterative decoding considered below messages can have different probabilities determined by the output of another decoder. The algorithm can be easily adapted to handle this.

Since the computation involves every path in the trellis (every codeword), the two-way algorithm performs bit MAP decoding. The organization of the computation is efficient for the same reason as above: computation for the common parts of different codewords is performed only once.

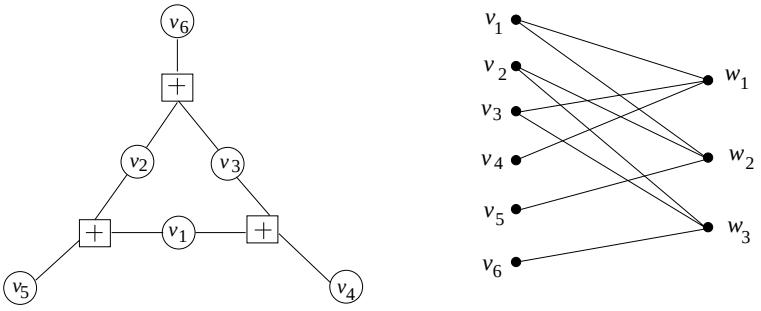
A general view of the BCJR algorithm and the sum-product algorithm is discussed in papers [1, 9, 5].

An important point is to realize that these algorithms can be generalized from trellises to "Tanner graph" representation of codes and from one- or two-pass to iterative decoding. This idea was put forward by [6, 12] implicitly and realized fully in [13].

Given a linear code with an $m \times n$ parity-check matrix $\mathbf{H}$ we represent it by a bipartite graph whose left vertices correspond to coordinates and right vertices to parity-check equations. We do not assume that $\mathbf{H}$ has rank $n - k$ (although it does in our examples); the only assumption is that it fully describes the set of parity checks of the code, i.e., $k \geq n - m$. There are a few possible ways to depict this graph. Consider a code whose p.-c. matrix is

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Its Tanner graph looks as follows:



These representations (the first one somewhat closer to the trellis, the second one to product and related codes) pave way for exporting the Viterbi and BCJR algorithms from trellis decoding to iterative decoding (message passing) algorithms.

THE MIN-SUM ALGORITHM This algorithm and the one after it generalize to the Tanner graph setting the Viterbi and BCJR algorithms defined for the trellis. Decoding is best described in terms of message passing between variable and check nodes.

We will need some notation. As is customary, the coordinates of the transmission are called *variables*. Let $G(V_0 \cup V_1, E)$ denote the Tanner graph, where V_0 is the variable part and V_1 the check part. Individual variable and check nodes are denoted by v_i, w_j respectively. Given a vertex v and an edge $e \in E$ incident to it, we will say that e is in the neighborhood of v and write $e \in \mathcal{N}(v)$. The number of edges incident to a vertex v is called its *degree*, $\deg(v) = |\mathcal{N}(v)|$. An edge can be also defined as a pair of vertices; in that case we write $e = (v, w)$.

Let us initialize the costs of variable nodes to $r_v(x) = -\log W(y_v|x)$ (on a BSC we can use the Hamming distance) and the costs of check nodes to zero. The message sent from a variable node v to a check node w in the ℓth iteration is computed as

$$\mu_{v,w}^{(\ell)}(x) = r_v(x) + \sum_{w' \neq w:\ (v,w') \in E} \mu_{w',v}^{(\ell-1)}(x).$$

By definition $\mu_{\cdot,\cdot}^{(-1)} = 0$. Essentially we are sending the reliability of the symbol x in coordinate v as determined by all the check equations that include it *except* w itself. Note the local nature of the decision

procedure to generate the message. The message sent from a check node w to a variable node v in iteration m is computed as

$$\mu_{w,v}^{(m)}(x) = \min_{\mathbf{x}} \sum_{v' \neq v:\ (w,v') \in E} \mu_{v',w}^{(m-1)}(x_{v'}),$$

where the minimum is computed over all valid assignments of bits to the check equation w which have the variable v equal to x, i.e., the vectors $\mathbf{x}$ such that

(1) $x_v = x$,
(2) $\mathbf{x}$ fulfills the check w.

This time we are sending a local estimate of the reliability of the symbol x in coordinate v as determined by all the other coordinates that enter the check w.

The procedure is terminated either when it converges (further iterations do not alter the local costs) or after a suitable number of steps, say s. The final costs of the variable nodes are computed as follows:

$$q_v(x) = r_v(x) + \sum_{w:\ (v,w) \in E} \mu_{w,v}^{(s)}(x).$$

Properties of this algorithm can be analyzed if G is a tree (does not contain cycles).

Theorem 1. *If the graph G is a tree, then the min-sum algorithm converges in a finite number of steps. The node $v \in V_0$ will be assigned the cost*

$$q_v(x) = \min_{\mathbf{c} \in \mathcal{C}, c_v = x} \Big(-\log W^n(\mathbf{y}|\mathbf{c}) \Big).$$

In other words, for every variable node v we will find the quantity

$$\min_{\substack{\mathbf{c} \in \mathcal{C} \\ c_v = x}} \Big(-\sum_{v \in V_0} \log W(y_{v'}|c_{v'}) \Big).$$

Remarks. 1. It is possible to accommodate different prior probabilities of codewords $P(\mathbf{c})$. For that the initial costs of the check nodes r_w are assigned values in such a way that $\log P(\mathbf{c}) = -\sum_w r_w$.

2. For a tree the min-sum algorithm computes the max-likelihood decision (if the codewords are equiprobable) or in general the max APP decision.

THE SUM-PRODUCT ALGORITHM. Essentially the only difference between the min-sum and the sum-product algorithms is in the fact that where the first takes a minimum over feasible bit assignments, the second computes a sum of probabilities for all the possible feasible bit assignments (also because of this sum we switch from logarithms to probabilities themselves).

Initialize the costs of the variable nodes to $r_v(x) = W(y_v|x), x = 0, 1$ and the check nodes to $r_c(x) = 1$. Messages from the variable side to the check side have the form

$$\mu_{v,w}^{(\ell)}(x) = r_v(x) \prod_{w' \neq w:\ (v,w') \in E} \mu_{w',v}^{(\ell-1)}(x).$$

where $\mu_{\cdot,\cdot}^{(-1)} = 1$ by definition.

Messages in the reverse direction look as follows:

$$\mu_{w,v}^{(m)}(x) = \sum_{\mathbf{x}} \prod_{v' \neq v:\ (w,v') \in E} \mu_{v',w}^{(m-1)}(x_{v'}),$$

where the sum is computed over all the vectors $\mathbf{x}$ such that

(1) $x_v = x$,
(2) $\mathbf{x}$ fulfills the check w

Termination conditions are the same as of the min-sum algorithm. The final costs of the variable nodes are computed as follows:

$$q_v(x) = r_v(x) \prod_{w:\ (v,w)\in E} \mu^{(s)}_{w,v}(x). q_v(x) = \prod_{w:\ (v,w)\in E} \mu^{(s)}_{w,v}(x).$$

Theorem 2. *If the graph G is a tree then the sum-product algorithm converges in a finite number of steps. The node $v \in V_0$ will be assigned the cost*

$$q_v(x) = \sum_{\mathbf{c}\in\mathcal{C}, c_v = x} W^n(\mathbf{y}|\mathbf{c}).$$

Thus on a cycle-free graph the sum-product algorithm performs bit MAP decoding. Indeed, if the codewords are equiprobable then the final cost of a vertex v is proportional to the bit MAP decision $P_v(x|\mathbf{y})$ from (1) since then for every codeword

$$P_v(x|\mathbf{c}) = \frac{\sum_{\substack{\mathbf{c}\in\mathcal{C}\\ c_v = x}} W^n(\mathbf{y}|\mathbf{c})}{\sum_{\mathbf{c}\in\mathcal{C}} W^n(\mathbf{y}|\mathbf{c})} \propto \sum_{\substack{\mathbf{c}\in\mathcal{C}\\ c_v = x}} W^n(\mathbf{y}|\mathbf{c}).$$

If the codewords are not equiprobable, then the initial values of the check costs $r_c(x)$ are set according to the prior distribution on the code.

The sum-product algorithm is also called the *belief propagation* algorithm.

Example: Message passing decoding for the Binary Erasure Channel. The importance of this example stems from the fact that for most part, rigorous analysis of message passing decoding of linear codes, and in particular, of LDPC codes is possible only on a BEC(p).

The algorithm proceeds as follows:

(1) variable node values are set to the values of the received symbols y_i
(2) the message sent on the edge (w, v) from the check node w to the variable node v is erasure if at least one of the edges $(v', w), v' \neq v$ is an erasure, or equals

$$\sum_{v'\neq v: v', w\in E} \mu^{(m-1)}_{v',w}$$

(sum mod 2 of the messages sent to w except the one from v).
(3) the message sent on the edge (v, w) from the variable node v to the check node w is erasure if both $y_v = *$ and all the messages $\mu^{(m-1)}_{v,w}$ are erasures, or it is the value of the variable node.

The reference to the value of the variable node in the last part may seem ambiguous because if y_v is erased then it might be that different check nodes send different guesses about the bit transmitted in position v. This is however not the case because the algorithm only makes such a guess if the check is completely sure of the value of the variable.

To make this algorithm look close to the sum-product we can use the isomorphism $\mathbb{Z}_2^+ \cong \{1, -1\}$ (the additive group of $\mathbb{Z}_2$ and the multiplicative group of 2nd degree roots of unity). It must be possible to set this decoding entirely in terms of sum-product.

LDPC codes. The next element in the picture is a particular code structure that supports the iterative procedures of the type discussed. It is suggested by the fact that decisions at vertices are taken locally based on the checks involving the coordinate (check) in question. Therefore it is plausible that we will keep decoding complexity under control and guarantee convergence if there are only few checks that involve a variable and few variables that figure in each check. In this part we follow [10].

A linear binary code $\mathcal{C}$ is called *low-density parity-check* or LDPC code if the number of ones in both each row and each column of some parity-check matrix $\mathbf{H}$ of $\mathcal{C}$ is bounded above by a constant. If $\mathbf{H}$ contains r ones per row and l ones per column the code is called a *regular* (r, l) code[2].

[2]A bipartite graph $G = (V_0 \cup V_1, E)$ such that $\deg v = r$ for every $v \in V_0$ and $\deg w = l$ for every $w \in V_1$ is called (l, r) regular. If $r = l = \Delta$, the graph is called Δ-regular.

Proposition 3. *The rate of an (r, l) LDPC code $R \geq 1 - \frac{l}{r}$.*

Proof : Let m be the number of rows in $\mathbf{H}$. Count the ones in $\mathbf{H}$ (the edges in the Tanner graph) in two ways:

$$(n-k)r \leq mr = nl.$$

∎

The main questions addressed in this setting are

(1) convergence and error probability of iterative decoding
(2) threshold probability of particular LDPC ensembles
(3) design of capacity approaching LDPC codes on binary-input channels (primary examples include the BEC, BSC, binary-input Gaussian channel)

We need certain notation. Let Λ_i be the number of variable nodes of degree i, P_i the number of check nodes of degree i. Consider the node degree generating functions

$$\Lambda(x) = \sum_{i \geq 1} \Lambda_i x^i, \qquad C(x) = \sum_{i \geq 1} C_i x^i.$$

where Λ_i, for instance, is the number of variable nodes of degree i. Consider also generating functions of "edge degrees"

$$\lambda(x) = \frac{\Lambda'(x)}{\Lambda'(1)} = \sum_i \lambda_i x^{i-1}, \qquad \rho(x) = \frac{C'(x)}{C'(1)} = \sum_i \rho_i x^{i-1},$$

where λ_i, for instance, is the fraction of edges that are incident on a variable node of degree $i-1$. We have

$$\Lambda(1) = n, \qquad C(1) \geq n(1-R), \qquad \Lambda'(1) = C'(1) = |E|$$

The normalized versions of these functions are

$$L(x) = \frac{\Lambda(x)}{\Lambda(1)}, \qquad \Gamma(x) = \frac{C(x)}{C(1)}.$$

For the code rate we have

$$R(\mathcal{C}) \geq 1 - \frac{C(1)}{\Lambda(1)}.$$

From now on we will disregard the difference between the true and designed code rate.

The Standard Ensemble of LDPC (n, λ, ρ) codes can be defined by choosing n variable nodes, $n(1-R)$ check nodes, and assigning edges according to a random permutation on the set of $|E|$ elements.

An important result, stated in the next theorem, shows that with high probability a code from the (n, λ, ρ) ensemble under message passing decoding has the bit error rate close to the expected bit error rate on the ensemble.

Theorem 4. *Suppose a code $\mathcal{C}$ from the (n, λ, ρ) ensemble is used on a BEC(p). For any $\epsilon > 0$ there exists a $\delta = \delta(\lambda, \rho, \epsilon, m)$ such that upon m iterations of message passing decoding,*

$$\Pr\left\{|P_b(\mathcal{C}, p, m) - \mathsf{E}\left(P_b(\mathcal{C}, p, m)\right)| > \epsilon\right\} \leq e^{-\delta n}.$$

This is a particular case of a more general result valid for a variety of communication channels under sum-product decoding. Another result, also stated without proof says that the expected bit error rate over the ensemble, converges to the optimal (MAP) bit error rate.

Theorem 5. *Given the degree distributions, consider a sequence of LDPC ensembles of growing length. Then*

$$\lim_{n \to \infty} \mathsf{E}\left(P_b, p, m\right) = P^{\mathrm{MAP}}_{m,\lambda,\rho}(p).$$

The quantity on the right-hand side is the bit error rate under m rounds of message-passing decoding of an ensemble of tree LDPC codes with edge degree distributions λ, ρ.

These two theorems show that a code chosen from the standard LDPC ensemble with high probability will have bit error rate equal to that of MAP decoding *under a simple iterative decoding algorithm*. The existence of such an algorithm constitutes the difference between these results and the corresponding results for the ensemble of random linear codes.

A related question is whether the threshold probability for the standard ensemble (under a suitable choice of the degree distributions) approaches capacity of the BEC $\mathscr{C} = 1 - p$. In brief, the answer is yes (this is the only interesting example where such a result is known).

We will study density evolution equations for the tree ensemble on an erasure channel.

Iterative decoding on the Tanner graph can be viewed as exchanging information between the decoders of a repetition code on the variable side and a single parity-check code on the check side. The algorithms can be generalized to other channels [11], message passing between decoders of other codes to cover iterative decoding of serially concatenated codes, parallel concatenations (turbo codes, whose decoding has been realized to be an instance of the belief propagation algorithm), repeat-accumulate codes and other schemes. Less is known about the analysis of such schemes, although a number of useful approximations are available [4].

In practical terms, many good LDPC and turbo codes have been constructed not just for the BEC, but also for the BSC and the Gaussian channel. Does this mean that the previous knowledge should now be discarded and all the effort spent on the study of iterative decoding? I think the answer is a firm "no": experience teaches us that good ideas reemerge in different contexts, and it is never clear which new connections and problems may arise. In particular, it seems unreasonable that we should ignore the code structure altogether in the decoding problem. Iterative schemes will form a solid part of coding theory but never displace the rest of it.

We conclude with a collection of sundry results on LDPC codes and their relatives.

Flipping algorithm. We discuss one simple iterative decoding algorithm of regular LDPC codes on a BSC which stands somewhat aside in the context of this lecture.

Let $\mathcal{C}$ be an (r, l) regular LDPC code with the parity-check matrix $\mathbf{H}$ and let $\mathbf{y}$ be the vector received from a BSC. Denote by $\mathbf{s}(\mathbf{y}) = \mathbf{H}\mathbf{y}^T$ be the syndrome of $\mathbf{y}$. Let $\mathbf{c}_i = (c_1 \dots c_{i-1} \bar{c}_i c_{i+1} \dots c_n)$ denote the vector $\mathbf{c}$ in which the ith bit has been inverted.

Flipping algorithm $(\mathcal{C}, \mathbf{y})$

- Put $\hat{\mathbf{c}} = \mathbf{y}$.
- Find a coordinate i such that $\mathrm{wt}(\mathbf{s}(\hat{\mathbf{c}}_i)) \le \mathrm{wt}(\mathbf{s}(\hat{\mathbf{c}}))$, assign $\hat{\mathbf{c}} \leftarrow \hat{\mathbf{c}}_i$.
- Repeat until no further replacements can be made, output $\hat{\mathbf{c}}$.

Let us connect convergence of the flipping algorithm to one property of the underlying Tanner graph, its expansion. Let $\mathbf{y}$ be the error vector and $\mathbf{s} = \mathbf{H}\mathbf{y}^T$ be its syndrome. Let us assume that there exists a function $w_*(e)$ such that for every $\mathbf{y}, \mathrm{wt}(\mathbf{y}) = e$, the weight of the syndrome $\mathrm{wt}(\mathbf{s}) \ge w_*(e)$. Thus,

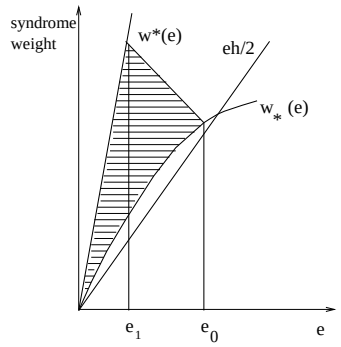
$$w_*(e) \le \mathrm{wt}(\mathbf{y}) \le w^*(e) := el.$$

Lemma 6. *If* $\mathrm{wt}(\mathbf{s}) > el/2$, *there exists a coordinate* i *such that* $\mathrm{wt}(\mathbf{s} + \mathbf{h}_i) < \mathrm{wt}(\mathbf{s})$.

Proof : Indeed, the average number of unsatisfied checks per coordinate is $\mathrm{wt}(\mathbf{s})/e > l/2$. Hence at least one coordinate contributes more than $l/2$ ones to $\mathbf{s}$, so flipping it reduces the weight of the syndrome. ∎

The assumption of the lemma will be satisfied if $w_*(e) > el/2$. Let e_0 be the maximum value of e such that this inequality holds true. Clearly $e_0 > 0$ for large n and constant l. For any error vector of weight $e \le e_0$ the algorithm will find a coordinate to invert. Inverting it, we reduce the weight of the syndrome and either add or remove one error. In the worst case, the number of errors becomes one greater and the weight

of syndrome one less. To visualize the decoding process, let us represent it as a trajectory in the "phase plane" showing the weight of syndrome vs. the multiplicity of errors. Starting from an (integer) point in the plane, in the worst case we advance to its south-east neighbor. Thus, for successful decoding, it is sufficient that moving along the worst-case trajectory, we do not leave the region $\mathrm{wt}(\mathbf{e}) < e_0$. We conclude that for the algorithm to converge, it is sufficient that the initial point is below the line drawn from the point $(e_0, e_0 l/2)$ with slope -1. Denote by e_1 the value of e for which this line and the upper bound $w^*(e)$ intersect. This is a lower bound on the number of correctable errors. A quick calculation shows that $e_1 > e_0/2$. Hence the algorithm corrects any $e_0/2$ or fewer errors.



It can be shown that if the algorithm converges, then in every step the weight of the syndrome reduces by a constant factor, so the complexity of the algorithm is $O(n \log n)$.

Lemma 7. *Suppose that every subset $T \subset V_1$ of $e \le \alpha n$ variable nodes have at least $(\frac{3}{4} + \epsilon)le$ neighbors in V_0, $\epsilon > 0$. The flipping algorithm for this code corrects an $\alpha/2$ proportion of errors.*

Proof : The total number of check nodes for which at least one of the edges is in error equals the weight $\mathrm{wt}(\mathbf{s})$ plus the number, x, of checks containing $2, 4, \ldots$ errors. By assumption, the number of neighbors of variables in error is

$$\mathrm{wt}(\mathbf{s}) + x > \frac{3}{4} le.$$

Every unsatisfied check contains at least one error, and every satisfied one at least two. Hence

$$\mathrm{wt}(\mathbf{s}) + 2x \le le.$$

These two inequalities imply that $\mathrm{wt}(\mathbf{s}) > el/2$, hence $e < e_0$. ∎

In general, the property of the graph under which the number of neighbors of every (not too large) subset of vertices is bounded from below, is called expansion, and graphs satisfying this are called *expander graphs*. In the next lecture we will see another application of expansion to error correction.

Theorem 8. (Gallager bound [6]) *Let $\mathcal{C}$ be an $[n, nR]$ linear code whose parity-check matrix has r ones in each row. Suppose that $\mathcal{C}$ is used for transmission on a BSC(p). If the error probability under ML decoding $P_e(\mathcal{C}) \to 0$ as $n \to \infty$ then*

$$R \le 1 - \frac{h_2(p)}{h\Big(\frac{1}{2}(1 - (1-2p)^r)\Big)}.$$

Proof : Let $\mathbf{c}$ be the transmitted and $\mathbf{y}$ the received.vectors.

$$I(\mathbf{c}, \mathbf{y}) = H(\mathbf{c}) - H(\mathbf{c}|\mathbf{y}) = H(\mathbf{y}) - H(\mathbf{y}|\mathbf{c}). \tag{3}$$

We have

$$H(\mathbf{c}) = nR, \quad H(\mathbf{y}|\mathbf{c}) = nh_2(p).$$

Let us estimate $H(\mathbf{y})$. Let $\mathbf{s}$ be the received syndrome. The trick is to realize that knowing $\mathbf{s}$ and the values of $\mathbf{y}$ on some information set of $\mathcal{C}$ is equivalent to knowing $\mathbf{y}$. This is true because the parity-check matrix can be

written as $[\mathbf{I}|\mathbf{A}]$, where the columns of $\mathbf{A}$ correspond to the information set E. Then $\mathbf{s} = \mathbf{y}(\bar{E}) + \mathbf{A}(\mathbf{y}(E))^T$. This implies the identity

$$H(\mathbf{y}) = H(\mathbf{s}) + H(\mathbf{y}|\mathbf{s}).$$

A parity check is satisfied if it contains an even number of errors; thus, the probability of this equals

$$\sum_{\text{even } i} \binom{r}{i} p^i (1-p)^{r-i} = \frac{1}{2}(1 + (1-2p)^r)$$

(this is verified by expanding $(1-p+p)^r + (1-p-p)^r$ by the binomial formula). Let us denote the right-hand side of the last equation by p_r. We have

$$H(\mathbf{s}) \le n(1-R)h_2(p_r)$$

since dependent rows can only decrease the value of entropy. Finally (with a bit of handwaving) $H(\mathbf{y}|\mathbf{s}) = H(\mathbf{y}) = nR$. Then

$$H(\mathbf{y}) \le n((1-R)h_2(p_r) + R).$$

Substituting into (3), we obtain

$$H(\mathbf{c}|\mathbf{y}) \ge n(h_2(p) - (1-R)h_2(p_r)).$$

Suppose that the right-hand side of this inequality is positive, or

$$R > 1 - \frac{h_2(p)}{h_2(p_r)}.$$

Then $H(\mathbf{c}|\mathbf{y}) \ge \epsilon > 0$. But then the error probability is bounded away from zero by Fano's inequality. Hence the bound claimed in the statement of the theorem is necessary for reliable communication. ∎

This theorem implies that a code (family) whose parity checks each involve a constant number of coordinates does not reach capacity of the BSC under ML decoding. Indeed, let r be large (but fixed), then the gap to capacity (the difference between the capacity and the threshold rate of a regular (r, ℓ) code) is

$$(1 - h_2(p)) - \left[1 - \frac{h_2(p)}{h_2(p_r)}\right] \approx \frac{h_2(p)}{2\ln 2}(1-2p)^{2r}$$

(recall the behavior of $h_2(\theta)$ in the neighborhood of $\theta = \frac{1}{2}$ from Lecture 2). It is still possible that the code comes arbitrarily close to capacity for large but fixed r (for $r \to \infty$ the upper bound on the threshold approaches $\mathscr{C}$) Gallager's bound can be improved and generalized to binary-input output-symmetric channels and irregular LDPC codes [3].

Asymptotic properties of LDPC ensembles. Consider the ensemble of regular (r, l) codes. As $r \to \infty$ and $l = r(1-R)$, the average relative distance in the ensemble approaches $\delta_{\mathrm{GV}}(R)$. Their weight profile is not far from the weight profile of a random linear code.

The bulk of ideas about LDPC codes (including the sum-product algorithm) were suggested by Gallager in [6].

References

1. S. M. Aji and R. J. McEliece, *The generalized distributive law*, IEEE Trans. Inform. Theory **46** (2000), no. 2, 325–343. MR 1 748 973
2. R. E. Blahut, *Algebraic codes for data transmission*, Cambridge University Press, 2003.
3. D. Burshtein, M. Krivelevich, S. Litsyn, and G. Miller, *Upper bounds on the rate of LDPC codes*, IEEE Trans. Inform. Theory **48** (2002), no. 9, 2437–2449. MR **2003j:**94121
4. S.-Y. Chung, T. J. Richardson, and R. L. Urbanke, *Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation*, IEEE Trans. Inform. Theory **47** (2001), no. 2, 657–670.
5. G. D. Forney, Jr., *Codes on graphs: normal realizations*, IEEE Trans. Inform. Theory **47** (2001), no. 2, 520–548. MR **2002f:**94055
6. R. G. Gallager, *Low-density parity-check codes*, MIT Press, Cambridge, MA, 1963.
7. C. Heegard and S. B. Wicker, *Turbo coding*, Kluwer Academic Publishers, Boston e.a., 1999.
8. W. C. Huffman and V. Pless, *Fundamentals of error-correcting codes*, New York: Cambridge University Press, 2003.
9. F. Kschischang, B. J. Frey, and H.-A. Loeliger, *Factor graphs and the sum-product algorithm*, IEEE Trans. Inform. Theory **47** (2001), no. 2, 498–519.

10. T. Richardson and R. Urbanke, *Modern coding theory*, Lecture Notes (2001–), EPFL, available on-line at `http://lthcwww.epfl.ch/teaching/coding2003.php`.
11. T. J. Richardson and R. L. Urbanke, *The capacity of low-density parity-check codes under message-passing decoding*, IEEE Trans. Inform. Theory **47** (2001), no. 2, 599–618.
12. M. Tanner, *A recursive approach to low-complexity codes*, IEEE Trans. Inform. Theory **27** (1981), no. 5, 1710–1722.
13. N. Wiberg, H.-A. Loeliger, and R Kötter, *Codes and iterative decoding on general graphs*, Eur. Trans. Telecomm. (1995), 513–525.
14. S. B. Wicker and S. Kim, *Fundamentals of codes, graphs, and iterative decoding*, Boston : Kluwer Academic Publishers, 2003.