Introduzione ai Low-Density Parity-Check codes (LDPC)

L. Giugno, M. Luise – 23/05/2003

1. Ceni Storici

I Low-Density Parity-Check codes (LDPC) furono presentati per la prima volta alla comunità scientifica da R. Gallager durante la sua dissertazione finale di Ph.D. al M.I.T. nel 1960. Tuttavia ad essi non è stata data molta importanza fino all'invenzione dei Turbo codici e del principio della decodifica iterativa da parte dei francesi Berrou e Glavieux nel 1993. Ciò ha permesso, difatti, la decodifica di sequenze di bit lunghe anche qualche migliaio di unità con complessità ragionevole senza ricorrere ad algoritmi di decodifica a massima verosimiglianza (Maximum Likelihood Sequence Estimator - MLSE) e con prestazioni molto vicine al limite di Shannon.

2. Codici a blocco

Un codice a blocco di tipo (*n,k*) è ottenuto aggiungendo *n-k* simboli binari di codice, ad un blocco di *k* simboli binari di sorgente. Il tasso di codice è pari a *k/n*.

3. Low-Density Parity-Check Codes

A questo punto è possibile introdurre i *Low-Density Parity-Check codes no*tando che faremo riferimento esclusivamente a sorgenti numeriche binarie, per cui le cifre generate potranno essere solo $\{0, 1\}$.

Consideriamo, quindi, i codici a blocco lineari (*n,k*) definiti all'interno del *campo* $\Omega_2 \triangleq \left(\{0,1\},+,\cdot\right)$, cioè l'insieme dei numeri binari in cui si definisce una operazione di somma ed una operazione di prodotto, entrambe modulo 2. Definiamo, poi, con Ω_2^n lo spazio vettoriale *n*-dimensionale su Ω_2. Gli elementi di Ω_2^n sono le 2^n *n*-uple $\mathbf{v} = [v_0, v_1, ..., v_{n-1}]$ che tratteremo come vettori riga. Un codice a blocco lineare (*n,k*) Γ risulta un *sottospazio vettoriale k*-dimensionale (*k*<*n*) di Ω_2^n. Indicando con $\mathbf{u} = [u_0, u_1, ..., u_{k-1}]$ una parola di sorgente di lunghezza *k*, è possibile associare ad ognuna di esse una corrispondente parola di codice, $\mathbf{c} = [c_0, c_1, ..., c_{n-1}]$ appartenente al sottospazio vettoriale Γ. Quindi alle 2^k parole di sorgente $\mathbf{u}$ di lunghezza *k* vengono associate 2^k parole di codice $\mathbf{c}$ di lunghezza *n*.

Essendo Γ un sottospazio vettoriale di dimensione *k,* esistono *k* vettori linearmente indipendenti, indicati con $\mathbf{g_0}$, $\mathbf{g_1}$,..., $\mathbf{g_{k-1}}$, che costituiscono una base di Γ. Quindi il passaggio da una parola di sorgente $\mathbf{u} = [u_0, u_1, \ldots, u_{k-1}]$ ad una di codice $\mathbf{c} = [c_0, c_1, \ldots, c_{n-1}]$ si realizza nel modo seguente:

$$\mathbf{c} = u_0 \cdot \mathbf{g_0} + u_1 \cdot \mathbf{g_1} + ... + u_{k-1} \cdot \mathbf{g_{k-1}} \tag{3.1}$$

cioè combinando linearmente i vettori della base di Γ tramite $\mathbf{u}$. Naturalmente, a due parole di sorgente diverse corrispondono due parole di codice diverse, condizione questa fondamentale per poter risalire in modo univoco, in fase di ricezione, al vettore di sorgente trasmesso. In forma matriciale ciò si esprime con:

$$\mathbf{c} = \mathbf{u} \cdot \mathbf{G} \tag{3.2}$$

dove la matrice:

$$\mathbf{G} = \begin{bmatrix} \mathbf{g_0} \\ \mathbf{g_1} \\ \cdots \\ \mathbf{g_{k-1}} \end{bmatrix}_{k \times n} \tag{3.3}$$

è chiamata *matrice generatrice* (generator matrix). Poiché $\mathbf{g_0}$, $\mathbf{g_1}$,..., $\mathbf{g_{k-1}}$ sono vettori linearmente indipendenti, il rango della matrice $\mathbf{G}$ è proprio *k* (ricordiamo che il rango di una matrice è il massimo numero di righe linearmente indipendenti) e per tale motivo, combinando opportunamente tra loro le righe mediante il metodo di *Gauss-Jordan*, può essere ricondotta alla forma seguente:

$$\mathbf{G} = \left[\begin{array}{ccccc|c} 1 & 0 & 0 & \cdots & 0 & * * * * * * \\ 0 & 1 & 0 & \cdots & 0 & * * * * * * \\ 0 & 0 & \ddots & \ddots & \vdots & * * * * * * \\ \vdots & \vdots & \ddots & \ddots & 0 & * * * * * * \\ 0 & 0 & \cdots & 0 & 1 & * * * * * * \end{array}\right]_{k \times n} = \left[\mathbf{I}_{k \times k} \,\middle|\, \mathbf{P}_{k \times (n-k)}\right]_{k \times n} \tag{3.4}$$

Potrebbe essere necessario, preventivamente, cambiare l'ordine di alcune colonne in modo che le prime *k* siano linearmente indipendenti tra loro e si possa ottenere la matrice identità semplicemente combinando le righe di $\mathbf{G}$. Naturalmente cambiando l'ordine di alcune colonne varia l'ordine dei corrispondenti bit nella parola di codice, ma le proprietà 'globali' del codice non vengono alterate.

Indichiamo con $\Gamma^{\perp}$ il *sottospazio ortogonale* di Γ che avrà necessariamente dimensione *n-k*. Chiamiamo $\mathbf{h_0}, \mathbf{h_1}, ..., \mathbf{h_{n-k-1}}$ gli *n-k* vettori della base di $\Gamma^{\perp}$. Essendo $\Gamma^{\perp}$ il sottospazio ortogonale di Γ, si verifica che:

$$\mathbf{c}\mathbf{h}_\mathbf{i}^\mathbf{T} = 0 \quad \forall\, \mathbf{h_i} \in \Gamma^{\perp}\,,\ \forall\, \mathbf{c} \in \Gamma \tag{3.5}$$

e che

$$\mathbf{x}\mathbf{h}_\mathbf{i}^\mathbf{T} \neq 0 \quad \forall\, \mathbf{h_i} \in \Gamma^{\perp}\,,\ \forall\, \mathbf{x} \notin \Gamma \tag{3.6}$$

cioè le parole di codice verificano l'equazione (3.5) che chiamiamo *controllo di parità* (parity-check). Ricorrendo anche in questo caso alla forma matriciale, possiamo definire la *matrice per il controllo della parità* (parity-check matrix) $\mathbf{H}$ come segue:

$$\mathbf{H} \triangleq \begin{bmatrix} \mathbf{h_0} \\ \mathbf{h_1} \\ \cdots \\ \mathbf{h_{n-k-1}} \end{bmatrix}_{(n-k) \times n} \tag{3.7}$$

Per quanto detto sopra, deve valere la seguente relazione:

$$\mathbf{c}\mathbf{H}^\mathbf{T} = \mathbf{0} \tag{3.8}$$

se e solo se $\mathbf{c} \in \Gamma$. Per un particolare codice lineare Γ, conoscendo la sua matrice $\mathbf{H}$, è possibile, in fase di ricezione, verificare la corretta operazione di decodifica mediante

$$\mathbf{c_{DEC}}\mathbf{H}^\mathbf{T} = \mathbf{0} \tag{3.9}$$

dove $\mathbf{c}_{\mathbf{DEC}}$ è la stima che il decodificatore compie sulla base della sequenza al suo ingresso $\mathbf{c}$. Se le equazioni di controllo di parità non sono verificate, possiamo rivelare la presenza di un errore di decodifica.

A questo punto resta da capire:

a) cosa significa in dettaglio l'operazione espressa dalla (3.8);

b) come è possibile ricavare, dato un particolare codice Γ, la sua matrice $\mathbf{H}$ di parity-check.

Per quanto riguarda il punto a), supponiamo che la parola di codice in esame $\mathbf{c}$ abbia *w* componenti uguali ad 1 (ovvero che la distanza di Hamming della parola in esame rispetto al vettore nullo sia *w*) e che questi 1 siano nelle posizioni indicate con $l_1, l_2, ..., l_w$. Ad esempio, se $n=8$ e $w=3$, con $\mathbf{c}=[0,1,0,1,1,0,0,0]$ si ha $l_1=1$, $l_2=3$ e $l_3=4$. L'operazione (3.8) corrisponde quindi a sommare tra loro gli elementi di *w* righe di $\mathbf{H}^{\mathbf{T}}$ per ottenere il vettore nullo. Queste righe sommate sono proprio quelle con indici $l_1, l_2, ..., l_w$.

Poiché la matrice $\mathbf{H}^{\mathbf{T}}$ ha dimensione *n* x (*n-k*), con l'equazione (3.8) vengono eseguiti contemporaneamente *m*=*n-k* controlli di parità per verificare che la parola di codice (da intendersi all'uscita del ricevitore digitale) sia effettivamente compatibile con quelle generabili nel sottospazio vettoriale Γ (e quindi sia stata correttamente decodificata). Un'importante conseguenza di questa osservazione è che la distanza minima d_{min}, del codice Γ è esattamente il numero minimo di righe di $\mathbf{H}^{\mathbf{T}}$ che devono essere sommate tra loro per avere il vettore nullo. La *distanza minima* di un codice è un parametro importante, perché conta il numero di posizioni nel quale le due parole di codice più vicine (nel senso di Hamming) differiscono. Quindi il codice è tanto migliore quanto maggiore è d_{min}.

Come esempio riportiamo un codice a blocco (7,4) definito tramite la seguente matrice $\mathbf{H}^{\mathbf{T}}$ (ovviamente *n*=7, *k*=4 e quindi *m*=*n-k*=3):

$$\mathbf{H}^{\mathbf{T}}_{n\times(n-k)} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}_{7\times 3} \tag{3.10}$$

In esso, il numero minimo di righe da sommare per ottenere il vettore nullo è 3, infatti:

$$\mathbf{riga_0} + \mathbf{riga_1} + \mathbf{riga_6} = [1,1,1]+[1,1,0]+[0,0,1] = \mathbf{0}$$

di conseguenza la distanza minima d_{min} del codice è 3.

Indichiamo ora con $\mathbf{w}$ il vettore degli errori di canale, cioè il vettore di disturbo (anch'esso con elementi $w_i \in \{0,1\}$) che si somma (modulo 2) alla parola di codice $\mathbf{c}$. Il vettore ricevuto è allora $\mathbf{y}=\mathbf{c}+\mathbf{w}$ e, in base alla (3.8), il decodificatore eseguirà *m*=*n-k*=7-4=3 controlli di parità:

$$\mathbf{y}\mathbf{H}^{\mathrm{T}}_{7\times 3} = [y_0,\ldots,y_6]\cdot\begin{bmatrix}1&1&1\\1&1&0\\1&0&1\\0&1&1\\1&0&0\\0&1&0\\0&0&1\end{bmatrix} = \mathbf{0} \;\Rightarrow\; \begin{cases} y_0+y_1+y_2+y_4=0\\ y_0+y_1+y_3+y_5=0\\ y_0+y_2+y_3+y_6=0\end{cases}$$

Rimandiamo alla sezione 6 l'esame del punto b), cioè come ricavare la matrice di controllo **H** da quella di generazione **G**.

4. Tipi di LDPC

Fino ad ora abbiamo parlato delle equazioni di "parity-check" eseguite dal decodificatore per stabilire la corretta rivelazione della sequenza binaria ricevuta, ma dalla discussione non è ancora emerso il perché questi codici sono chiamati "low-density".

Un codice LDPC code è dunque un codice a blocco lineare e sistematico per il quale la matrice di parity-check ha una bassa densità di 1 ("low-density"), cioè la maggior parte dei suoi elementi è uguale a zero. Un codice LDPC *regolare* di tipo (*n*,*k*) *(regular LDPC code)* è un codice a blocco lineare la cui matrice **H** contiene esattamente un numero $w_c \geq 3$ di 1 per ogni colonna e un numero $w_r = w_c \cdot n/(n-k)$ di 1 per ogni riga, con $w_c \ll m = n-k$. Il tasso del codice è quindi:

$$r_{REG} = \frac{k}{n} = 1 - \frac{w_c}{w_r} < 1 \tag{4.1}$$

Un codice LDPC *iregolare* di tipo (*n*,*k*) *(irregular LDPC code)* è un codice a blocco lineare la cui matrice **H** è sempre una matrice *sparsa* (ovvero a bassa densità di 1, "low density"), ma nella quale il numero di 1 per colonna o per riga non è costante. Il tasso del codice è quindi:

$$r_{IRREG} = \frac{k}{n} < 1 \tag{4.2}$$

In fig. 4.1 è riportato un esempio di matrice di controllo di parità 25 x 50 relativa ad un codice LDPC regolare di tipo (50,25).

Definiamo come *fattore di sovrapposizione* (overlap) il risultato del prodotto scalare tra due colonne qualsiasi all'interno della matrice. Dal nostro esempio, si nota che comunque prendiamo due colonne di **H** queste presentano al più un solo 1 sulla stessa riga, per cui il fattore di sovrapposizione risulta ≤ 1. In generale, più piccolo è il fattore di sovrapposizione di **H**, più uniforme è la distribuzione degli elementi diversi da 0 al suo interno.

5. Grafo di Tanner

Una rappresentazione alternativa a quella matriciale, per i codici LDPC, è stata introdotta da Tanner. Tale modalità, una volta fissato il particolare codice LDPC da usare, permette di visualizzare, tramite un *grafo bipartito*, il legame esistente tra i bit di una qualsiasi parola di codice e le equazioni per il controllo della parità. Tale schema è appunto il *Grafo di Tanner*.

Un *grafo bipartito* è un grafico costituito da due classi di nodi, dove le connessioni che si possono realizzare sono solo quelle che legano due nodi appartenenti a classi diverse.

Nel caso specifico del Grafo di Tanner le due classi sono:

a) *nodi di bit (bit nodes)* associati ai bit di codice;

b) *nodi di controllo (check nodes)* associati alle equazioni per il controllo della parità.

Il Grafo di Tanner di un codice va realizzato secondo la seguente regola:

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

Fig. 4.1 – Esempio di matrice di parity-check

"il nodo di controllo *j* è connesso al nodo di bit *i* $\Leftrightarrow$ l'elemento h_{ji} di $\mathbf{H}$ è uguale ad 1"

Evidentemente il numero di nodi di controllo è *m*=*n*-*k* e il numero dei nodi di bit è *n*. Per capire meglio come sono fatti tali grafici facciamo un esempio con una matrice piccola. Prendiamo un codice a blocco (10,5), con $w_c = 2$ e $w_r = w_c \cdot n/(n-k) = 4$. La matrice di controllo $\mathbf{H}$ è la seguente:

$$\mathbf{H}_{(n-k)\times n} = \begin{bmatrix}
1&1&1&1&0&0&0&0&0&0\\
1&0&0&0&1&1&1&0&0&0\\
0&1&0&0&1&0&0&1&1&0\\
0&0&1&0&0&1&0&1&0&1\\
0&0&0&1&0&0&1&0&1&1
\end{bmatrix}_{5\times 10} \tag{5.1}$$

il grafo bipartito associato è in fig. 5.1:

Vediamo che i nodi b_0, b_1, b_2 e b_3 sono legati al nodo c_0 conseguentemente al fatto che sulla prima riga di $\mathbf{H}$ abbiamo $h_{00} = h_{01} = h_{02} = h_{03} = 1$ (gli altri elementi di $\mathbf{H}$ sulla prima riga sono nulli). Cosa analoga accade anche per gli altri nodi, rispettando così la regola enunciata sopra. Inoltre, come conseguenza del fatto che $\mathbf{cH^T} = \mathbf{0}$, il valore assunto dai bit connessi allo stesso nodo di controllo deve essere tale che la loro somma risulti uguale a zero.

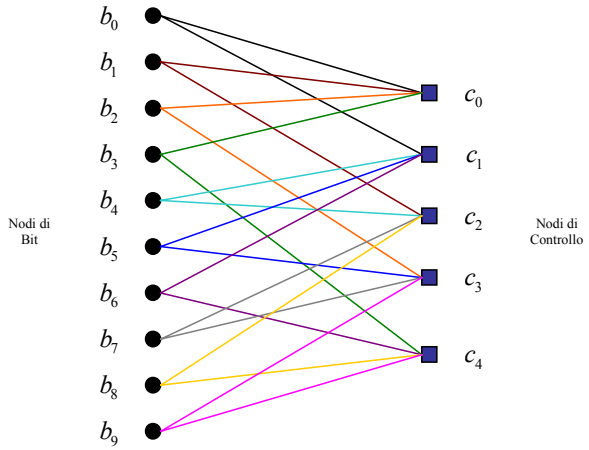

Fig. 5.1 – Esempio di Tanner graph.

Ritornando alla fig. 5.1, si nota anche che il grafo risulta essere regolare: ogni nodo di bit ha 2 rami (connessioni), cioè ha *grado* 2, ed ogni nodo di controllo ha 4 rami, cioè ha *grado* 4. Anche questo è in accordo col fatto che $w_c = 2$ e $w_r = 4$. Si comprende quindi che stiamo trattando un LDPC regolare, anche perché vale $w_r = w_c \cdot n/(n-k)$.

6. Processo di Costruzione del Codice e Codifica

Nella sezione 3 ci eravamo chiesti come è possibile realizzare e sfruttare in fase di decodifica la matrice $\mathbf{H}$ di controllo di parità di un codice LDPC Γ.

Per costruire un codice LDPC regolare occorre dapprima generare una matrice $\mathbf{H}$ di tipo "low-density". Questo compito può essere effettuato a calcolatore seguendo alcuni passi:

1. Si scelgono k (lunghezza della parola di sorgente), tasso del codice r, w_c (numero di 1 per colonna);
2. Si calcolano n (lunghezza della parola di codice) e $w_r = w_c \cdot n/(n-k)$ (numero di 1 per riga);
3. Si azzera il contatore delle colonne $i_c = 0$;
4. Si genera in maniera pseudocasuale un vettore di peso w_c (cioè contenente w_c simboli 1) nella colonna i_c;
5. Se il peso di ogni riga della matrice in costruzione è $\le w_r$ e l'overlap tra ogni coppia di colonne è ≤ 1 (low-density), $i_c = i_c + 1$, altrimenti si torna passo 4;
6. Se $i_c = n-1$ l'algoritmo termina, altrimenti si torna al passo 4.

Una volta costruita $\mathbf{H}$ si provvede poi a metterla nella forma:

$$\mathbf{H} = \left[\mathbf{P}^{\mathrm{T}} \middle| \mathbf{I}\right] \tag{6.1}$$

tramite il metodo di *Gauss-Jordan*, dove la matrice $\mathbf{P}^{\mathrm{T}}$ è la stessa che compare nella matrice di generazione $\mathbf{G} = \left[\mathbf{I} \middle| \mathbf{P}\right]$, utilizzata per codificare una parola di sorgente e $\mathbf{I}$ è la matrice identità di opportune dimensioni. Poiché $\mathbf{H}$ viene generata in modo pseudocasuale, può accadere che non abbia rango pieno (abbia cioè rango minore di $n-k$). In tal caso il metodo di Gauss-Jordan restituisce una matrice nella forma:

$$\mathbf{H} = \begin{bmatrix} \tilde{\mathbf{P}}^{\mathrm{T}} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}_{(n-k)\times n} \tag{6.2}$$

dove $\mathbf{I}$ ha dimensioni $m' \times m'$ e quindi $\tilde{\mathbf{P}}^{\mathrm{T}}$ $m' \times (n - m')$.
In questa situazione non si può usare $\mathbf{H}$ nella forma (6.1), ma la si deve 'ridurre' a $\tilde{\mathbf{H}}$ del tipo:

$$\tilde{\mathbf{H}} = \left[\tilde{\mathbf{P}}^{\mathrm{T}} \middle| \mathbf{I} \right]_{m' \times n} \tag{6.3}$$

con dimensione $m' \times n$. Ovviamente ciò modifica il tasso del codice, che non sarà più k/n, ma $(n - m')/n$ e quindi leggermente maggiore. Se questa procedura non è soddisfacente, si deve procedere daccapo alla generazione di una nuova $\mathbf{H}$ in modo casuale.

Immaginiamo però di essere riusciti nel nostro intento e di aver riportato la matrice di parity-check nella forma:

$$\mathbf{H}_{(n-k)\times n} = \left[\mathbf{P}^{\mathrm{T}}_{(n-k)\times k} \middle| \mathbf{I}_{(n-k)\times(n-k)} \right] \tag{6.4}$$

Ricordando che

$$\mathbf{G}_{k\times n} = \left[\mathbf{I}_{k\times k} \middle| \mathbf{P}_{k\times(n-k)} \right] \tag{6.5}$$

la parola di codice viene costruita sulla base di quella di sorgente secondo

$$\mathbf{c} = \mathbf{uG} = \left[\mathbf{u} \middle| \mathbf{uP} \right] \tag{6.6}$$

dove $\mathbf{u}$ è la parte *sistematica*. In sintesi, la matrice che costruisce la parola di codice, $\mathbf{G}$, può essere ricavata direttamente della matrice di parità, riportando $\mathbf{H}$ nella forma (6.4) ed estraendo da essa la sottomatrice $\mathbf{P}$.

7. Decodifica Iterativa

L'algoritmo di decodifica iterativa utilizzato per la rivelazione di codici LDPC è noto con il nome di *Message-Passing* o *Belief-Propagation*. Di esso esistono diverse versioni, alcune operanti nel dominio della probabilità, altre nel dominio del logaritmo (*Sum-Product*), e tra queste ultime esistono versioni sub-ottime ma a complessità ridotta (*Min-Sum* o *Normalized Min-Sum*). In questo paragrafo forniremo una descrizione panoramica degli principali algoritmi operanti nel dominio del logaritmo (più stabili dal punto di vista numerico). In particolare introdurremo l'algoritmo *Sum-Product* e le sue versioni a complessità ridotta *Min-Sum* e *Normalized Min-Sum*.
Prima di passare alla descrizione di questi algoritmi di tipo SISO (Soft-In Soft-Out), per meglio comprendere come le equazioni di parity-check giochino un ruolo fondamentale nel processo di decodifica, facciamo un esempio di decodifica di tipo "hard" che è conosciuta col nome di *Bit Flipping*.

Consideriamo un codice (formalmente assimilabile ad un LDPC regolare, anche se $\mathbf{H}$ non è affatto low-density) caratterizzato dalla seguente matrice di controllo di parità:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{7.1}$$

e supponiamo di aver trasmesso la parola di codice $\mathbf{c}$

$$\mathbf{c} = [1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1] \tag{7.2}$$

la quale viene ricevuta correttamente a meno di un errore nella seconda componente:

$$\mathbf{y} = \mathbf{c} + [0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0] \tag{7.3}$$

$$\mathbf{y} = [1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1] \tag{7.4}$$

Disegnando il Grafo di Tanner in questa situazione (fig. 7.1), si può notare come sia possibile sfruttare l'informazione risiedente nei nodi di bit per verificare le equazioni di controllo di parità. In una prima fase (Semi-Iterazione verso Destra) i nodi di bit forniscono un'informazione 'a priori' ai nodi di controllo, che è costituita dal valore che essi assumono. Ricevuta questa informazione, i nodi di controllo verificano le equazioni di controllo di parità e infine 'ritrasmettono' ai nodi di bit il risultato dei loro controlli (Semi-Iterazione verso Sinistra). Nel nostro caso, si nota che l'equazione di parity-check al nodo c_2 è soddisfatta (mentre non sono soddisfatte le altre) e quindi è presumibilmente sbagliato il valore assunto da uno dei nodo di bit *non* collegati al nodo di controllo c_2. Controllando esaustivamente questi bit e provando a invertirne il loro valore (*flipping*) si arriva a capire che y_1 è il bit sbagliato, e quindi lo si corregge, soddisfacendo quindi simultaneamente tutte le equazioni di parity-check.

E' ovvio che quando le dimensioni della matrice $\mathbf{H}$ aumentano, il controllo esaustivo diventa improponibile. In più il rumore additivo non assume valori discreti, ma è generalmente modellabile come un processo Gaussiano bianco. Non avendo quindi la certezza sul valore dei nodi di bit o sulla verifica dei nodi di controllo, si sfrutta la loro distribuzione di probabilità condizionata, calcolabile secondo un algoritmo iterativo (*Sum-Product*) descritto in seguito.

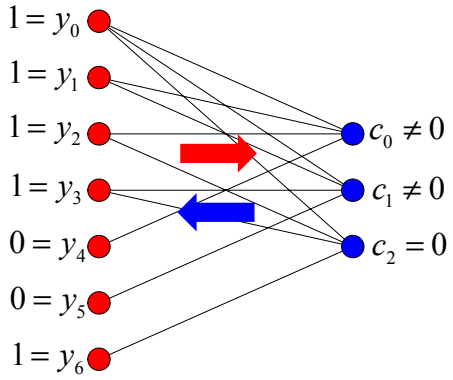

Fig. 7.1 – Esempio di decodifica iterativa.

Prima di introdurre questo algoritmo, ricorreremo ad una particolare notazione, introdotta da MacKay, che ci permetterà di alleggerire le formule che incontreremo più avanti.
Indichiamo con:

$$q_{i,j}(a) \triangleq \Pr\{u_i = a \,|\, y_i, c_{j' \neq j}\} \tag{7.5}$$

la probabilità che il bit trasmesso u_i sia uguale a a, con $a \in \{0,1\}$, condizionata al simbolo di canale y_i ricevuto e all'informazione proveniente da tutti i nodi di controllo j' escluso j. Questa grandezza di tipo "soft" rappresenta una sorta di "informazione estrinseca" che viene trasmessa dal nodi

di bit b_i al nodi di controllo c_j, quindi da sinistra verso destra (Semi-Iterazione verso Destra) (vedi fig. 7.2).

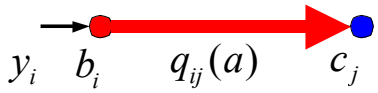

Fig. 7.2 – Semi-Iterazione verso Destra.

Indichiamo poi con

$$r_{j,i}(a) \triangleq \Pr\left\{c_j = 0 \middle| u_i = a, \left\{q_{i,j'}\right\}_{j' \neq j}\right\} \tag{7.6}$$

la probabilità che la *j*-ma equazione di controllo di parità sia soddisfatta condizionatamente al fatto che il bit u_i sia uguale a a (con $a \in \{0,1\}$) e al fatto che gli altri bit abbiano una distribuzione di probabilità pari a $\left\{q_{i,j'}\right\}_{j' \neq j}$. Questa seconda grandezza "soft" rappresenta invece il messaggio trasmesso dal nodo di controllo c_j verso il nodo di bit b_i, quindi da destra verso sinistra (Semi-Iterazione verso Sinistra) (fig. 7.3).

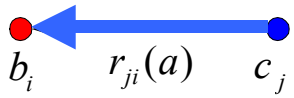

Fig. 7.3 – Semi-Iterazione verso Sinistra.

Questo operazioni "da sinistra verso destra" e "da destra verso sinistra" nel Grafo di Tanner possono essere viste come "*decodifica verticale*" e "*decodifica orizzontale*" rispettivamente sulle righe e sulle colonne della matrice di controllo di parità $\mathbf{H}$, come sarà più chiaro alla fine di questa sezione. Ricordiamo che l'algoritmo *Sum-Product* opera nel dominio logaritmico: a tal fine definiamo le seguenti quantità:

$$L(c_i) \triangleq \ln\left[\frac{\Pr(u_i = +1 | y_i)}{\Pr(u_i = -1 | y_i)}\right] \tag{7.7}$$

$$L(q_{i,j}) \triangleq \ln\left[\frac{q_{i,j}(0)}{q_{i,j}(1)}\right] = sign\left(L(q_{ij})\right) \cdot \left|L(q_{ij})\right| = \alpha_{ij}\beta_{ij} \tag{7.8}$$

$$L(r_{j,i}) \triangleq \ln\left[\frac{r_{j,i}(0)}{r_{j,i}(1)}\right] \tag{7.9}$$

$$\phi(x) \triangleq -\ln\left[\tanh\left(\frac{x}{2}\right)\right] = \ln\left[\frac{e^x + 1}{e^x - 1}\right] \tag{7.10}$$

La funzione $\phi(x)$ ha l'andamento riportato in fig. 7.4:

L'algoritmo *Sum-Product* opera nel seguente modo:

1. Inizializzazione, basata solo sull'osservazione dell'uscita del canale:

$$L(q_{i,j}) = L(c_i) = \frac{2 \cdot y_i}{\sigma^2} \qquad \forall i, j \Rightarrow h_{ij} = 1 \tag{7.11}$$

2. Semi-Iterazione verso Destra (dai nodi di bit verso i nodi di controllo):

$$L(r_{j,i}) = \left(\prod_{i' \in R_{j \setminus i}} \alpha_{i'j} \right) \cdot \phi \left(\sum_{i' \in R_{j \setminus i}} \phi\left(\beta_{i'j}\right) \right) \tag{7.12}$$

(La notazione $R_{j \setminus i} = \{ i' : h_{ji'} = 1 \} \setminus \{ i \}$ indica l'insieme di tutte le locazioni colonna in cui c'è un 1 nella riga *j*-sima, escludendo la locazione *i*)

3. Semi-Iterazione verso Sinistra (Dai nodi di controllo verso i nodi di bit):

$$L(q_{i,j}) = L(c_i) + \sum_{j' \in C_{i \setminus j}} L\left(r_{j'i}\right) \tag{7.13}$$

(La notazione $C_{i \setminus j} = \{ j' : h_{j'i} = 1 \} \setminus \{ j \}$ indica l'insieme di tutte le locazioni riga in cui c'è un 1 nella colonna *i*-sima, escludendo la locazione *j*)

4. Si calcolano le statistiche di decisione:

$$L(Q_i) = L(c_i) + \sum_{j \in C_i} L\left(r_{j,i}\right) \qquad i = 1, \ldots, n \tag{7.14}$$

5. Decisione a soglia e controllo:

$$\hat{c}_i = \begin{cases} 1, & \text{se } L(Q_i) < 0 \\ 0, & \text{altrimenti} \end{cases} \qquad i = 1, \ldots, n \tag{7.15}$$

se vale $\hat{\mathbf{c}}\mathbf{H}^{\mathrm{T}} = \mathbf{0}$ oppure se si è raggiunto il massimo numero di iterazioni allora la decodifica termina, altrimenti si torna allo step 2.

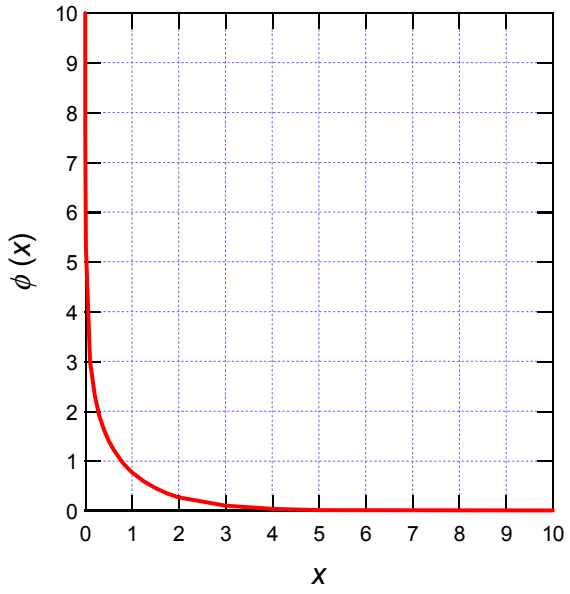

Fig. 7.4 – Andamento della funzione $\phi(x)$.

L'algoritmo *Min-Sum* nasce come versione sub-ottima ma a complessità ridotta dell'algoritmo Sum-Product. Osservando l'equazione (7.12), e conoscendo l'andamento di $\phi(x)$ in fig. 7.4, è possibile concludere che il termine dominante nella sommatoria $\sum_{i'} \phi\left(\beta_{i',j}\right)$ è quello che presenta il minimo $\beta_{i',j}$. Alla luce di questo fatto, scriviamo:

$$\phi\left[\sum_{i'} \phi\left(\beta_{i',j}\right) \right] \cong \phi\left[\phi\left(\min_{i'} \beta_{i',j} \right) \right] = \min_{i'} \beta_{i',j} \tag{7.16}$$

dato che $\phi\left[\phi(x)\right]=x$. A questo punto il *Min-Sum* segue gli stessi passi del *Sum-Product* ad esclusione dell'equazione (7.12) che diventa:

$$L\left(r_{j,i}\right)=\left(\prod_{i'\in R_{j\backslash i}}\alpha_{i',j}\right)\cdot\min_{i'\in R_{j\backslash i}}\beta_{i',j}\ . \tag{7.17}$$

Sia nel *Sum-Product* che nel *Min-Sum* è richiesta la conoscenza della varianza del rumore termico (σ^2) per poter eseguire la decodifica. Esiste, tuttavia, un formulazione alternativa del *Min-Sum* che non richiede la conoscenza di questo parametro. Questo algoritmo è noto con il nome di *Normalized Min-Sum*. Infatti, si nota che normalizzando per $\frac{2}{\sigma^2}>0$ la (7.11) le prestazioni non cambiano, in quanto il minimo $\beta_{i',j}$ in (7.17) rimane comunque l'elemento minimo anche se normalizzato per una quantità positiva.

Per meglio comprendere le operazioni formalizzate in (7.12) e (7.13) riportiamo, a titolo di esempio, il calcolo delle quantità $L\left(r_{2,3}\right)$ e $L\left(q_{1,1}\right)$, rispettivamente, assegnata una particolare matrice di controllo di parità

$$\mathbf{H}=\begin{bmatrix}1&1&1&0&1&0&0\\1&1&0&1&0&1&0\\1&0&1&1&0&0&1\end{bmatrix} \tag{7.18}$$

Nel primo caso (fig. 7.5) abbiamo $j=2$ e $i=3$. Inizialmente si fissa la riga della matrice **H** di indice $j=2$. In seguito, si scorrono gli elementi con indice di colonna i appartenenti alla riga selezionata "raccogliendo" l'informazione presente ogni qual volta si incontra un elemento uguale ad '1' caratterizzato da un indice di colonna $i'\neq 3$. Più in generale, fissato il nodo di controllo *j*, l'informazione estrinseca passatagli dal nodo di bit *i*, è quella pervenuta da tutti gli altri nodi di bit *i'* collegati al nodo di controllo *j*, con *i'*≠*i* ("*decodifica orizzontale*").

$$\mathbf{H}=\begin{bmatrix}1&1&1&0&1&0&0\\1&1&0&1&0&1&0\\1&0&1&1&0&0&1\end{bmatrix}\qquad j=2$$

$$i'\neq 3$$

Fig. 7.5 – Esempio di "decodifica orizzontale": calcolo di $L\left(r_{2,3}\right)$.

Nel secondo caso (fig. 7.6) abbiamo $i=1$ e $j=1$. Inizialmente si fissa la colonna della matrice **H** di indice $i=1$. Quindi si scorrono gli elementi con indice di riga j appartenenti alla colonna selezionata "raccogliendo" l'informazione presente ogni qual volta si incontra un elemento uguale ad '1' caratterizzato da un indice di riga $j'\neq 1$. Più in generale, fissato il nodo di bit *i*, l'informazione estrinseca passatagli dal nodo di controllo *j*, è quella pervenuta da tutti gli altri nodi di controllo *j'* collegati al nodo di bit *i*, con *j'*≠*j* ("*decodifica verticale*").

8. Vantaggi degli LDPC Irregolari

In questa sezione cercheremo di capire perché gli LDPC irregolari hanno delle prestazioni potenzialmente migliori di quelli regolari. Facendo riferimento ad un Grafo di Tanner, se i nodi di bit avessero grado *alto*, essi permetterebbero di prendere decisioni sui bit con maggiore affidabilità, in

quanto riceverebbero più informazioni dai diversi nodi di controllo. D'altro canto, se i nodi di controllo avessero grado *basso*, essi manderebbero informazioni più *"pesanti"* ai nodi di bit, in quanto ripartirebbero la loro informazione (che, ricordiamo, è di tipo 'a priori' per i nodi di bit) tra un numero basso di nodi di bit.

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \quad j' \neq 1 \quad (i = 1)$$

Fig. 7.6 – Esempio di "decodifica verticale": calcolo di $L(q_{1,1})$.

Sappiamo, però, che negli LDPC regolari, questa situazione non è realizzabile, in quanto i nodi di bit e i nodi di controllo hanno lo stesso grado. Al contrario, negli LDPC irregolari, questi gradi sono variabili, e i nodi di bit di grado maggiore risultano convergere più velocemente al loro corretto valore (in segno), aiutando poi nel processo di convergenza i nodi di bit di grado "intermedio", i quali poi aiutano quelli di grado più basso.

9. Prestazioni

Nella Fig. 9.1 possiamo vedere le prestazioni di un LDPC regolare di tipo (1024,512) con r=1/2 e modulazione BPSK al variare del numero di iterazioni e al variare dell'algoritmo di decodifica. Come previsto, l'algoritmo Min-Sum presenta, a parità di BER, un perdita in termini di rapporto segnale rumore di circa 0.5 dB rispetto al Message-Passing (algoritmo Sum-Product).

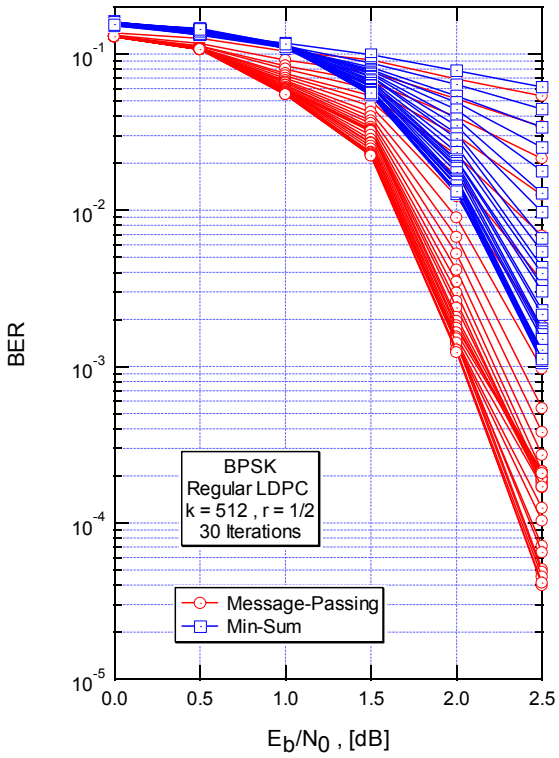

Fig. 9.1 – BER con BPSK e LDPC regolare con k=512 e differenti algoritmi di decodifica.

Nella Fig. 9.2 sono riportate le prestazioni di un LDPC regolare di tipo (8000,4000) con r=1/2 e modulazione BPSK con 50 iterazioni di decodifica. Dal confronto con la figura 9.1, è evidente il miglioramento delle prestazioni all'aumentare delle dimensioni del blocco di sorgente k.

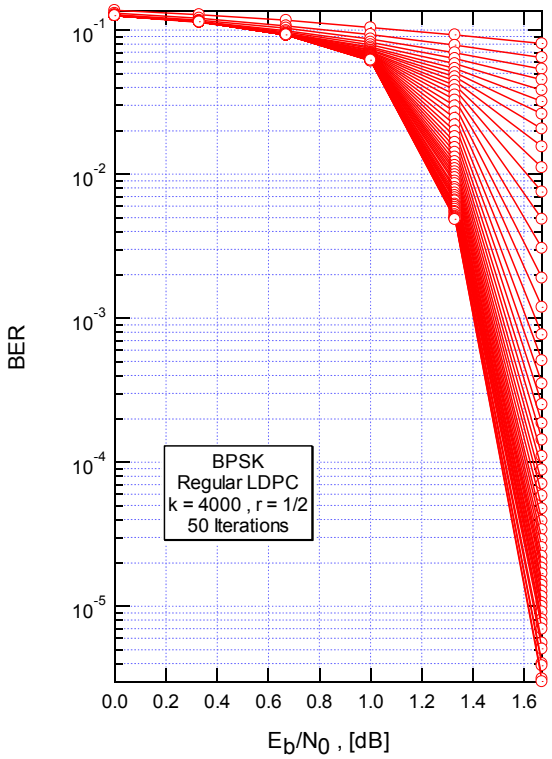

Fig. 9.2 – BER con BPSK e LDPC regolare con k=4000.

La Fig. 9.3 mostra infinele prestazioni di un LDPC regolare di tipo (2000,1500) con r=3/4 e modulazione 16-QAM con 30 iterazioni di decodifica.

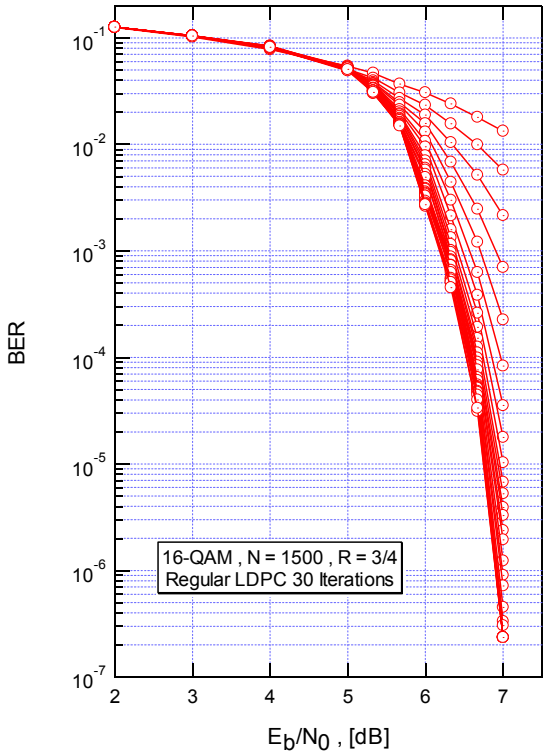

Fig. 9.3 – BER con 16-QAM e LDPC regolare con k=1500.

10. LDPC vs Turbo Codici

Nella fig. 10.1 sono riportate le BER di un Codice Turbo con modulazione BPSK, r=1/2 e k=500.000, paragonate a quelle di un LDPC irregolare e di un LDPC regolare con 3 '1' per colonna e 6 '1' per riga. Il guadagno del LDPC irregolare sul codice turbo è circa 0.2 dB in termini di rapporto segnale rumore (SNR) e la distanza dal limite di Shannon è circa 0.1 dB.

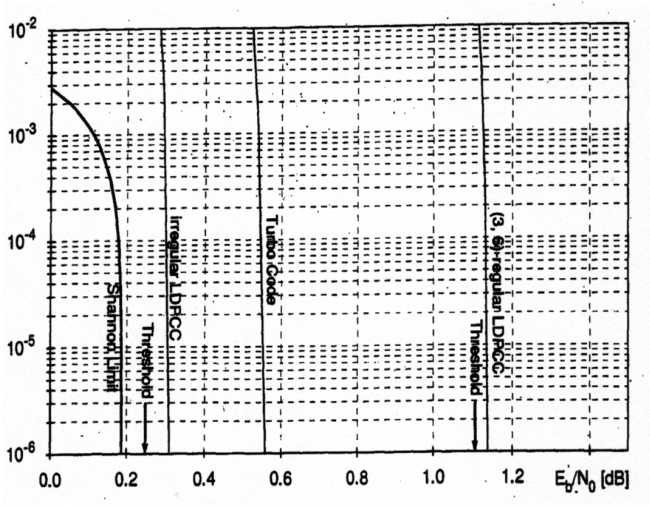

Fig. 10.1 – BER con Turbo Codice, LDPC regolare e LDPC irregolare con r=1/2 e k=500.000.

Nella fig. 10.2 sono riportate le BER con modulazione BPSK r=1/2 di un Codice Turbo (curve tratteggiate) paragonate a quelle di un LDPC irregolare (curve a tratto continuo) al variare della lunghezza della parola di sorgente k (nel grafico indicata con 'n'). Si nota che per lunghezza di blocco inferiori a 10.000 bit, i Codici Turbo garantiscono delle prestazioni migliori, mentre la situazione si inverte per blocchi più lunghi.

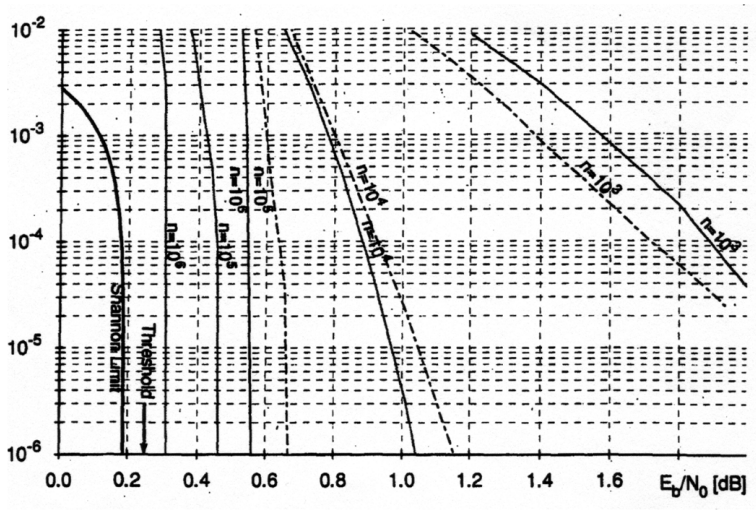

Fig. 10.2 – BER con Turbo Codice e LDPC irregolare con r=1/2 per vari valori di k.

Riassumendo, gli LDPC presentano i seguenti vantaggi nei confronti dei Turbo Codici:

1. la decodifica non richiede la conoscenza della varianza del rumore termico σ^2;
2. è possibile arrestare le iterazioni se sono soddisfatte tutte le equazioni di parity-check (almeno per alti SNR);

3. prestazioni migliori (almeno per LDPC irregolari e blocchi lunghi);
4. 1 solo decoder costituente SISO (Soft-In Soft-Out), contro i 2 dei Turbo Codici;
5. complessità di un'iterazione di decodifica minore rispetto a quella di un Turbo Codice.

Dal punto di vista degli svantaggi segnaliamo:

1. la necessità di impiegare un maggior numero di iterazioni (30-40) per raggiungere le prestazioni ottenibili con i Turbo Codici (10-12 iterazioni);
2. il processo di costruzione del codice sicuramente più complicato rispetto a quello dei Turbo Codici.

11. Bibliografia

[1] R.G. Gallager, "Low-Density Parity-Check Codes" *IRE Trans. on Inf. Theory*, IT-8, Jan 1962, pp. 21-28, Jan 1962.

[2] T.J. Richardson *et al.*, "Design of Capacity-Approaching Low-Density Parity-Check Codes", *IEEE Trans. on Inf. Theory*, Vol. 47, pp 616-637, Feb. 2001.

[3] D.J.C. MacKay, "Good Error Correcting Codes Based on Very Sparse Matrices", *IEEE Trans on Inf. Theory*, Vol. 45, No. 2, pp 399-431, 1999.

[4] M. Fossorier *et al.,* "Reduced Complexity Iterative Decoding of Low-DensityParity-Check Codes Based on Belief Propagation", *IEEE Trans. on Comm.*, pp.673-680, May 1999.

[5] T.J. Richardson *at al.*, "Efficient Encoding of Low-Density Parity-Check Codes", *IEEE Trans. on Inf. Theory*, Feb. 2001.

[6] D. MacKay *et al.*, "Comparison of Constructions of Irregular Gallager Codes", *IEEE Trans. on Comm.*, Oct. 1999.

[7] M. Fossorier *et al.*, "Low-Density Parity-Check Codes Based on Finite Geometries: A rediscovery and More", *IEEE Trans. on Inf. Theory*, Vol. 47, pp. 2711-2736, Nov. 2001.

[8] S. Haykin, *Communication Systems*, 4th Edition, John Wiley & Sons, Inc., 2000.