Università degli studi di Padova

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Tesi di Laurea Magistrale in

Ingegneria delle Telecomunicazioni

Decodifica di codici LDPC con tecniche di programmazione lineare

Relatore

Prof. Tomaso Erseghe

Candidato

Riccardo Pollis

Anno Accademico 2012/2013

Alla mia famiglia

Sommario

I codici Low Density Parity Check (LDPC) sono dei codici a correzione d'errore introdotti da Gallager [8] nel 1960, ma, a causa della complessità troppo elevata per l'epoca, abbandonati per più di trent'anni.

Grazie al lavoro di Tanner [13], che nel 1981 ne introdusse una rappresentazione grafica, furono riscoperti all'inizio degli anni '90. Vennero inventate tecniche di decodifica che sfruttavano tale rappresentazione per costruire un algoritmo basato sullo scambio di messaggi (Sum-Product).

I codici LDPC riescono a raggiungere prestazioni molto vicine al limite di Shannon e per questo sono utilizzati in molte applicazioni pratiche come ad esempio DVB, IEEE 802.11n, IEEE 802.16e.

Il problema degli algoritmi per la decodifica attualmente usati, risulta essere che la probabilità d'errore tende a saturare per valori di SNR (Signal to Noise Ratio) elevati. Questo impedisce l'utilizzo degli LDPC in applicazioni come la trasmissione su fibra o la registrazione magnetica in cui sono richieste probabilità d'errore particolarmente basse.

In questa tesi si analizzeranno degli algoritmi che considerano la decodifica come un problema di programmazione lineare (PL) e lo risolvono. La decodifica tramite PL sembra non soffrire la saturazione, tuttavia, utilizzando tecniche tradizionali di risoluzione di problemi PL (ad esempio il simplesso), la complessità diventa troppo elevata. Risulta quindi necessario trovare degli approcci diversi.

Dopo aver rivisto alcuni concetti generali, sui codici LDPC e sulla decodifica attraverso l'utilizzo dell'algoritmo Sum-Product (SP), si vogliono analizzare alcuni degli algoritmi PL proposti in letteratura per la decodifica. Si svolgeranno, infine, delle simulazioni per vedere la differenza tra le prestazioni degli algoritmi PL e del Sum-Product.

In molti degli articoli analizzati, vengono fatte delle simulazioni molto lunghe, al fine di dimostrare che gli algoritmi PL, a differenza del Sum Product, non saturano. Nelle simulazioni fatte su questa tesi, invece, si analizzeranno le prestazioni degli algoritmi PL, in termini di BER (Bit Error Ratio) e di tempo di decodifica, senza focalizzarsi sulla saturazione.

L'obiettivo della tesi è capire se la decodifica con tecniche di programmazione lineare rappresenta una valida alternativa all'algoritmo Sum Product.

Indice

Capitolo 1

Introduzione

Si vuole fare una breve introduzione alla codifica di canale, con particolare attenzione alla decodifica di codici binari. Verranno inoltre introdotti alcuni concetti sull'ottimizzazione convessa e si spiegherà che vantaggi ha la decodifica con tecniche di programmazione lineare rispetto a quella tradizionale.

1.1 Codifica di canale

La codifica di canale ha lo scopo di rendere un segnale da trasmettere più robusto agli errori, per fare questo si trasforma il messaggio di partenza $\mathbf{u} \in \mathcal{U}$ in uno più lungo $\mathbf{c} \in \mathcal{C}$. Per trasmettere il segnale $\mathbf{c}$ saranno necessarie più risorse, essendo questo più grande di quello originale, ma si può in compenso sfruttare la ridondanza per avere trasmissioni meno soggette ad errori. Il ricevitore può correggere in due modi diversi il messaggio che gli arriva:

1. Automatic repeat request (ARQ): Si rileva se sono stati commessi errori ed eventualmente si chiede al trasmettitore di inviare nuovamente il messaggio.

2. Foward error correction (FEC): Non solo si rilevano gli errori, ma si effettua anche la correzione del messaggio ricevuto.

Con la tecnica FEC non c'è ovviamente nessuna sicurezza che il messaggio venga corretto in modo esatto, l'ARQ invece è più affidabile, ma il tempo richiesto per la ritrasmissione può risultare troppo elevato, soprattutto nel caso di applicazioni real-time.

In questa tesi ci si occuperà solo dei codici LDPC che sono di tipo FEC.

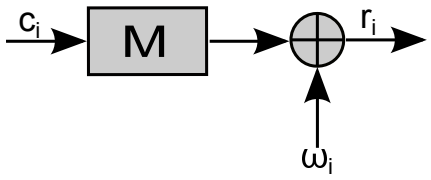


Figura 1.1: Canale AWGN

1.2 Codifica di codici a blocco binari

Si consideri una parola di codice $\mathbf{u}$ di lunghezza k che viene codificata in $\mathbf{c}$ di lunghezza n, dove

$$\begin{aligned} \mathbf{u} \in \mathcal{U} = \{0,1\}^k, \quad & |\mathcal{U}| &= 2^k \\ \mathbf{c} \in \mathcal{C} = \{0,1\}^n, \quad & |\mathcal{C}| &= 2^k \end{aligned}$$

Nel seguito lavoreremo solo con codici aventi parole binarie.

Si definisce code rate ($R = \frac{k}{n}$) il rapporto tra le lunghezze della parola di codice originale e di quella codificata. Si ha $0 < R < 1$, più ci si avvicina allo 0, maggiore sarà la ridondanza e quindi serviranno più risorse per trasmettere il segnale.

Il canale considerato sarà di tipo AWGN e la sua rappresentazione equivalente viene riportata in Figura 1.1, dove il segnale codificato $\mathbf{c}$ entra nel modulatore $\mathbf{M}$, gli si somma il rumore $\boldsymbol{\omega} \in \mathcal{N}(0, \sigma_\omega^2)$ e si ottiene così il segnale al ricevitore $\mathbf{r}$.

1.3 Decodifica di codici a blocco binari

Al ricevitore il segnale verrà decodificato con uno dei seguenti metodi:

- **Hard decoding**: La demodulazione e la decodifica del segnale avvengono in momenti diversi; Il modulatore decide il valore di ogni bit e poi il decodificatore sceglie la parola di codice stimata.

- **Soft decoding**: C'è un unico blocco che fa sia da demodulatore che da decodificatore.

Nel seguito si considererà solo la decodifica di tipo Soft, che sarà poi quella utilizzata da tutti gli algoritmi analizzati.

Si vede ora in maggior dettaglio come viene effettuata la decisione del simbolo stimato nel soft decoder. A causa del rumore, il segnale ricevuto può essere teoricamente un qualsiasi punto in $\mathbb{R}^n$. Si divide allora lo spazio in partizioni, $\mathcal{R}_\mathbf{a}$ con $\mathbf{a} \in \mathcal{C}$, una per ogni parola di codice ammissibile, in modo che $\bigcup_{\mathbf{a}\in\mathcal{C}} \mathcal{R}_\mathbf{a} = \mathbb{R}^n$. Il criterio di decisione è vedere a che regione

appartiene il vettore ricevuto $\mathbf{r}$ e scegliere, quindi, come segnale stimato $\hat{\mathbf{c}} = \mathbf{a}$. Si individuano delle regioni ottime tali che

$$\mathcal{R}_{\mathbf{a}} = \arg\min_{\mathcal{R}_{\mathbf{a}}} \; p(\hat{\mathbf{c}} \neq \mathbf{c}) = \arg\max_{\mathcal{R}_{\mathbf{a}}} \; p(\hat{\mathbf{c}} = \mathbf{c})$$

oppure equivalentemente

$$\begin{aligned} p(\hat{\mathbf{c}} = \mathbf{c}) &= \sum_{\mathbf{a} \in \mathcal{C}} p(\hat{\mathbf{c}} = \mathbf{a} | \mathbf{c} = \mathbf{a}) p(\mathbf{c} = \mathbf{a}) \\ &= \sum_{\mathbf{a} \in \mathcal{C}} p(\mathbf{r} \in \mathcal{R}_{\mathbf{a}} | \mathbf{c} = \mathbf{a}) p(\mathbf{a}) \end{aligned}$$

dove per semplificare la notazione si è posto $p(\mathbf{c} = \mathbf{a}) = p(\mathbf{a})$. Integrando sulle regioni si ottiene

$$p(\hat{\mathbf{c}} = \mathbf{c}) = \sum_{\mathbf{a} \in \mathcal{C}} \int_{\mathcal{R}_{\mathbf{a}}} p(\mathbf{x}|\mathbf{a}) p(\mathbf{a}) \, d\mathbf{x} \tag{1.1}$$

Si distinguono due regole di decisione in base a come vengono scelte le regioni: maximum a posteriori (MAP) e maximum likelihood (ML).

1.3.1 Maximum a posteriori (MAP)

Una possibile scelta delle regioni, per massimizzare la (1.1), consiste nel prendere come $\mathcal{R}_{\mathbf{a}}$, tutti i punti in cui la funzione da integrare è massima per la parola di codice $\mathbf{a}$ ovvero

$$\mathcal{R}_{\mathbf{a}} = \left\{ \mathbf{x} \;\middle|\; p(\mathbf{x}|\mathbf{a}) p(\mathbf{a}) = \max_{\mathbf{b}} p(\mathbf{x}|\mathbf{b}) p(\mathbf{b}) \right\} \tag{1.2}$$

Nel caso MAP, quindi, si costruiscono le regioni come descritto in (1.2) e, quando al ricevitore arriva il simbolo $\mathbf{r}$, si vede a che regione $\mathcal{R}_{\mathbf{a}}$ appartiene e si decodifica il messaggio in $\hat{\mathbf{c}} = \mathbf{a}$. Di conseguenza il criterio di decisione risulta essere

$$\hat{\mathbf{c}} = \arg\max_{\mathbf{a} \in \mathcal{C}} \; p(\mathbf{r}|\mathbf{a}) p(\mathbf{a}) \tag{1.3}$$

La (1.3) può anche essere riscritta come

$$\hat{\mathbf{c}} = \arg\max_{\mathbf{a} \in \mathcal{C}} \; p(\mathbf{c} = \mathbf{a}|\mathbf{r}) p(\mathbf{r}) \tag{1.4a}$$

$$= \arg\max_{\mathbf{a} \in \mathcal{C}} \; p(\mathbf{c} = \mathbf{a}|\mathbf{r}) \tag{1.4b}$$

Come si vedrà in seguito l'algoritmo di decodifica Sum-Product per codici LDPC è di tipo MAP.

1.3.2 Maximum likelihood (ML)

Nella maggior parte delle applicazioni pratiche i simboli sono equiprobabili ($p(\mathbf{c} = \mathbf{a}) = \frac{1}{\mathcal{C}}$) e, se non è così, li si può rendere tali con lo scrambling. Grazie a queste considerazioni la (1.3) può essere riscritta, ottenendo la formulazione ML

$$\hat{\mathbf{c}} = \arg\max_{\mathbf{c}\in\mathcal{C}} \; p(\mathbf{r}|\mathbf{c}) \tag{1.5}$$

Nel caso di assenza di memoria del canale vale

$$p(\mathbf{r}|\mathbf{c}) = \prod_{j=1}^{n} p(r_j|c_j) \tag{1.6}$$

Al posto della (1.6) si può scegliere di massimizzare $\log\, p(\mathbf{r}|\mathbf{c}) = \sum_{j=1}^{n} \log\, p(r_j|c_j)$, si definisce poi la log-likelihood ratio come $\gamma_j = \log(\frac{p(r_j|0)}{p(r_j|1)})$. Nel caso di parole di codice binarie $c_j \in 0, 1$ vale la relazione

$$\log\, p(r_j|c_j) = -\gamma_j c_j + \log\, p(r_j|0)$$

Il problema ML diventa minimizzare $\sum_{j=1}^{n} \gamma_j c_j$. Di conseguenza $\hat{\mathbf{c}}$ si trova risolvendo il problema

$$\text{Minimize} \quad \gamma^T \mathbf{c} \tag{1.7a}$$

$$\text{Subject to} \quad \mathbf{c} \in \mathcal{C} \tag{1.7b}$$

Nei Capitoli 4 e 5 il problema (1.7) verrà risolto con l'uso di tecniche di programmazione lineare (Linear Programming decoder).

1.4 Ottimizzazione convessa

In [4] un problema di ottimizzazione convessa viene scritto come

$$\text{Minimize} \quad f_0(\mathbf{x}) \tag{1.8a}$$

$$\text{Subject to} \quad f_i(\mathbf{x}) \leq b_i, \quad i = 1, ..., m \tag{1.8b}$$

Il nome ottimizzazione convessa deriva dal fatto che le funzioni $f_0, ..., f_m : \mathbb{R}^n \to \mathbb{R}$ sono convesse ovvero

$$f_i(\alpha\mathbf{x} + \beta\mathbf{y}) \leq \alpha f_i(\mathbf{x}) + \beta f_i(\mathbf{y}), \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n, \forall \alpha, \beta \in \mathbb{R} \tag{1.9}$$

Un problema di programmazione lineare è un caso di particolare del problema (1.8), che si distingue per il fatto che tutte le funzioni sono lineari. In generale un problema di programmazione lineare può essere scritto come

$$\text{Minimize} \quad \mathbf{c}^T \mathbf{x} \tag{1.10a}$$

$$\text{Subject to} \quad \mathbf{a}_i^T \mathbf{x} \leq b_i, \quad i = 1, ..., m \tag{1.10b}$$

Dove i vettori $\mathbf{c}, \mathbf{a}_i, ..., \mathbf{a}_m \in \mathbb{R}^n$ e gli scalari $b_1, ..., b_m \in \mathbb{R}$ sono i parametri del problema. Si vede allora come la (1.7) sia molto simile alla (1.10), per farla diventare un problema lineare basta esprimere il vincolo $\mathbf{c} \in \mathcal{C}$ nella forma usata in (1.10b).

1.4.1 Lagrangian

Si consideri il problema di ottimizzazione scritto in forma standard come

$$\begin{aligned} \text{Minimize} \quad & f_0(\mathbf{x}) && \text{(1.11a)} \\ \text{Subject to} \quad & f_i(\mathbf{x}) \leq 0, \quad i = 1, ..., m && \text{(1.11b)} \\ & h_i(\mathbf{x}) = 0, \quad i = 1, ..., p && \text{(1.11c)} \end{aligned}$$

con $\mathbf{x} \in \mathcal{D} \subset \mathbb{R}^n$ e sia $\mathbf{x}^*$ il valore ottimo. La (1.11) è una formulazione generale del problema e quindi non sono state fatte ipotesi sulle funzioni f e h, se fosse un problema di programmazione lineare tali funzioni sarebbero lineari. Se il problema non è scritto in forma standard può essere sempre riportato in tale forma.

Il Lagrangian del problema (1.11) si scrive come

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i f_i(\mathbf{x}) + \sum_{i=1}^{p} \mu_i h_i(\mathbf{x}) \tag{1.12}$$

Dove si sono introdotte le variabili $\boldsymbol{\lambda} \in \mathbb{R}^m$ e $\boldsymbol{\mu} \in \mathbb{R}^p$ dette moltiplicatori di Lagrange, che hanno lo scopo di assegnare un peso a tutte le funzioni rappresentanti i vincoli del problema.

1.4.2 Lagrange dual function

L'idea che sta alla base del Lagrangian è quella di sommare alla funzione obiettivo dei termini di penalità, composti da costanti (i moltiplicatori di Lagrange) che moltiplicano le funzioni dei vincoli. In questo modo il Lagrangian aumenta in modo proporzionale a quanto sono violati i vincoli. Si definisce quindi la Lagrange dual function come

$$g(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \min_{\mathbf{x} \in \mathcal{D}} f_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i f_i(\mathbf{x}) + \sum_{i=1}^{p} \mu_i h_i(\mathbf{x}) \tag{1.13}$$

1.4.3 Lagrange dual problem

Per ogni coppia $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ con $\boldsymbol{\lambda} \geq 0$ la Lagrange Dual function (1.13) fornisce un lower bound per il valore ottimo $\mathbf{x}^*$ del problema primale (1.11). Risulta quindi necessario trovare qual'è il miglior lower bound ottenibile dalla

Lagrange dual function, per fare questo si introduce il problema chiamato Lagrange dual problem ovvero

$$\begin{aligned} \text{Mazimize} \quad & g(\boldsymbol{\lambda}, \boldsymbol{\mu}) \\ \text{Subject to} \quad & \boldsymbol{\lambda} \geq 0 \end{aligned} \tag{1.14}$$

1.4.4 Augmented Lagrangian

In [3] viene spiegato l'Augmented Lagrangian, che nel Capitolo 5 verrà usato per la decodifica con il metodo ADMM.
Si riscrive il problema di programmazione lineare (1.10) in forma più compatta

$$\text{Minimize} \quad f(\mathbf{x}) \tag{1.15a}$$

$$\text{Subject to} \quad \mathbf{A}\mathbf{x} = \mathbf{b} \tag{1.15b}$$

dove

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1^T \\ . \\ . \\ . \\ \mathbf{a}_m^T \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ . \\ . \\ . \\ b_m \end{bmatrix}$$

Invece di risolvere il problema (1.15), si consideri il problema equivalente

$$\text{Minimize} \quad f(\mathbf{x}) + \frac{c}{2}||\mathbf{A}\mathbf{x} - \mathbf{b}||^2 \tag{1.16a}$$

$$\text{Subject to} \quad \mathbf{A}\mathbf{x} = \mathbf{b} \tag{1.16b}$$

con c costante positiva, che ha influenza sulla convergenza dell'algoritmo per trovare il valore ottimo di $\mathbf{x}$. L'Augmented Lagrangian risulta quindi essere il Lagrangian del problema (1.16) ovvero

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\lambda}(\mathbf{A}\mathbf{x} - \mathbf{b}) + \frac{c}{2}||\mathbf{A}\mathbf{x} - \mathbf{b}||^2 \tag{1.17}$$

1.5 Saturazione codici LDPC e soluzione

Nei capitoli successivi verranno analizzati in dettaglio i codici LDPC e vari algoritmi di decodifica. Si può introdurre che, per questi codici, la decodifica tradizionale, fatta con decoder di tipo Sum-Product, ha prestazioni vicine al limite di Shannon per valori di SNR (Signal to Noise Ratio) bassi. A SNR alti, invece, saturano e, di conseguenza, la BER (Bit Error Ratio) non scende sotto una certa soglia. Questo preclude l'uso degli LDPC in molte applicazioni pratiche (trasmissione su fibra ottica, registrazione magnetica).

Le tecniche di decodifica che usano metodi di programmazione lineare, invece, non soffrono la saturazione, inoltre possiedono anche la proprietà del ML Certificate, la quale assicura che, se la decodifica fornisce una soluzione intera è quella corretta. L'obiettivo di questa tesi è quindi di capire se la decodifica tramite la PL è competitiva con le tecniche tradizionali, in termini di BER e tempi di decodifica.

La suddivisione della tesi sarà la seguente: Nel Capitolo 2 si enuncia qualche cenno teorico sui codici LDPC, con particolare attenzione a quelli usati dall'IEEE 802.11n e ai codici di Margulis, in quanto saranno poi usati per le simulazioni . Nel Capitolo 3 verrà analizzato l'algoritmo Sum-Product che, come già detto, è un metodo di tipo MAP usato nella decodifica tradizionale di codici LDPC. Si vedranno poi nei Capitoli 4 e 5 dei metodi di decodifica di tipo ML che usano tecniche di programmazione lineare (PL). Nel Capitolo 6 si riporteranno le simulazioni fatte ed infine nel Capitolo 7 si ricapitoleranno i risultati ottenuti.

Capitolo 2

Codici Low Density Parity Check (LDPC)

Si analizzano ora i codici a correzione d'errore Low Density Parity Check (LDPC) per i quali, in seguito, verranno analizzati i vari algoritmi di decodifica.

2.1 Introduzione ai codici LDPC

I low density parity check code sono dei codici a correzione d'errore, lineari a blocco. Furono introdotti per la prima volta da Gallanger nel 1963 [8], ma, visto che la complessità computazionale era troppo elevata per l'epoca, furono abbandonati per più di trent'anni.

Gli LDPC sono caratterizzati da una parity check matrix $\mathbf{H}$ contenente tanti 0 e qualche 1, da questo deriva il nome low density parity check. Definendo con k la lunghezza delle parole di codice in ingresso al codificatore ($\mathbf{u}$) e n di quelle in uscita ($\mathbf{c}$), la parity check matrix $\mathbf{H}$ ha dimensione $n-k \times n$ e rango pieno. L'insieme delle possibili parole di codice dipende da $\mathbf{H}$ e risulta essere

$$\mathcal{C} = \{\mathbf{c} \in \mathbb{F}_2^n | \mathbf{Hc} = \mathbf{0}\} \tag{2.1}$$

Esistono due classi di codici LDPC:

- Codici regolari : La parity check matrix ha esattamente d_ρ elementi a 1 per ogni riga e d_λ per ogni colonna
- Codici irregolari: ogni riga-colonna può avere un numero diverso di elementi ad 1

2.1.1 Rappresentazione grafica

La rappresentazione grafica, dovuta a Tanner, è un grafo bipartito in cui gli insiemi dei nodi sono:

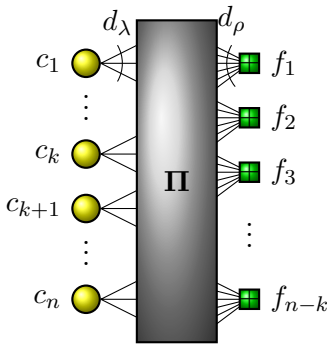


Figura 2.1: Codici LDPC regolari (3,6)

- Variable node: Un nodo per ogni bit del segnale codificato $\mathbf{c}$ o equivalentemente per ogni colonna di $\mathbf{H}$, quindi in totale n nodi
- Check node: Un nodo per ogni riga della parity check matrix, complessivamente $n - k$ nodi.

Esiste un lato tra il variable node j e il check node i se nella parity check matrix si ha $H(i, j) = 1$.

Nella Figura 2.1 si vede l'esempio del grafo di un codice regolare avente $d_\rho = 3$ e $d_\lambda = 6$, per avere una rappresentazione più chiara non è stato disegnato tutto il percorso dei lati, ma è presente un blocco di permutazione $\mathbf{\Pi}$.

2.1.2 Prestazioni

I codici LDPC hanno prestazioni molto vicine al limite di Shannon, in particolare, in [9] si dimostra che sono i codici LDPC irregolari quelli che ci si avvicinano di più. Per avere un'idea delle prestazioni si riporta in Figura 2.2 il confronto, in termini di Bit Error Ratio (BER), tra dei codici regolari e irregolari entrambi con Rate $R = \frac{1}{2}$.

In generale si può affermare che le prestazioni in termini di BER sono migliori quanto più maggiore è la dimensione del blocco da codificare, d'altra parte avere blocchi di dimensioni elevati implica l'introduzione di un ritardo nella codifica e nella decodifica del segnale.

2.2 Esempi di codici LDPC

2.2.1 IEEE 802.11n

Lo standard IEEE 802.11n [1] prevede l'utilizzo di codici LDPC irregolari. Nella Figura 2.3 viene riportata la tabella con tutte le possibili dimensioni

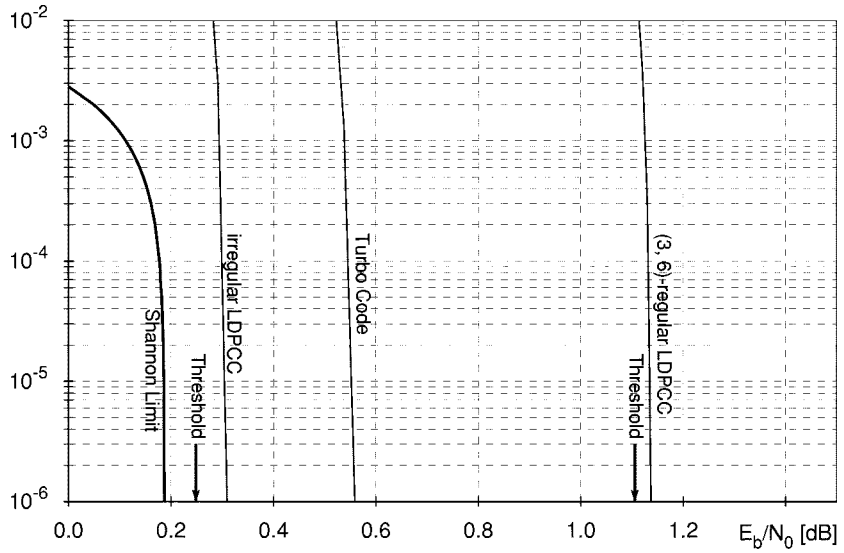


Figura 2.2: Confronto tra BER di codici LDPC regolari (3,6), turbo codici e LDPC irregolari: *"Richardson, Shokrollahi, Urbanke. Design of Capacity-Approaching Irregular Low-Density Parity-Check Codes." 2001*

della sequenza in ingresso e in uscita al decodificare.

Nelle simulazioni sono stati usati codici con $n = 648$, $k = 324$ e $n = 1296$, $k = 648$.

2.2.2 Costruzione della parity check matrix

In Figura 2.4 viene riportato un esempio di matrici di permutazione cicliche P_i, dove i indica lo shift rispetto alla matrice identità (P_0 è la matrice identità). La parity check matrix sarà quindi la composizione di queste sottomatrici, secondo un certo schema, che viene specificato (attraverso una matrice) per tutte le dimensioni della sequenza codificata e per i rate previsti nello standard.

Per quanto riguarda i codici LDPC IEEE802.11n, usati nella simulazione: Per il (648,324) la matrice per generare la parity check matrix è quella riportata in Figura 2.5, mentre per il (1296,648) quella in Figura 2.6. Ogni numero rappresenta una matrice di dimensione $Z \times Z$ che può essere nulla $(-)$, la matrice identità (0) oppure la sua i-esima permutazione (i).

Si ricorda, infine, che questo standard utilizza codici LDPC irregolari e quindi non c'è un numero fisso di vicini per ogni check e variable node. Per quanto riguarda il codice $(648, 324)$, costruito secondo le regole riportate sopra, si ha che il numero di vicini di ogni check node può essere 7 o 8, mentre per i variable node $2, 3, 4$ oppure 11. Per il codice $(1296, 648)$ ogni check node ha sempre 7 o 8 vicini mentre i variable node $2, 3$ oppure 12

Coding rate (R)	LDPC information block length (bits)	LDPC codeword block length (bits)
1/2	972	1944
1/2	648	1296
1/2	324	648
2/3	1296	1944
2/3	864	1296
2/3	432	648
3/4	1458	1944
3/4	972	1296
3/4	486	648
5/6	1620	1944
5/6	1080	1296
5/6	540	648

Figura 2.3: Possibili dimensioni dei codici LDPC: *"IEEE Std 802.11n." 2009*

$$P_0 = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix}, P_1 = \begin{bmatrix} 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \\ 1&0&0&0&0&0&0&0 \end{bmatrix}, P_5 = \begin{bmatrix} 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \\ 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \end{bmatrix}$$

Figura 2.4: Esempio matrici di permutazione con $Z = 8$: *"IEEE Std 802.11n." 2009*

0	-	-	-	0	0	-	-	0	-	-	0	1	0	-	-	-	-	-	-	-	-	-	-
22	0	-	-	17	-	0	0	12	-	-	-	-	0	0	-	-	-	-	-	-	-	-	-
6	-	0	-	10	-	-	-	24	-	0	-	-	-	0	0	-	-	-	-	-	-	-	-
2	-	-	0	20	-	-	-	25	0	-	-	-	-	-	0	0	-	-	-	-	-	-	-
23	-	-	-	3	-	-	-	0	-	9	11	-	-	-	-	0	0	-	-	-	-	-	-
24	-	23	1	17	-	3	-	10	-	-	-	-	-	-	-	-	0	0	-	-	-	-	-
25	-	-	-	8	-	-	-	7	18	-	-	0	-	-	-	-	-	0	0	-	-	-	-
13	24	-	-	0	-	8	-	6	-	-	-	-	-	-	-	-	-	-	0	0	-	-	-
7	20	-	16	22	10	-	-	23	-	-	-	-	-	-	-	-	-	-	-	0	0	-	-
11	-	-	-	19	-	-	-	13	-	3	17	-	-	-	-	-	-	-	-	-	0	0	-
25	-	8	-	23	18	-	14	9	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0
3	-	-	-	16	-	-	2	25	5	-	-	1	-	-	-	-	-	-	-	-	-	-	0

Figura 2.5: Matrice per generare la parity check matrix nel caso $n = 648$ e $k = 324$, $Z = 37$: *"IEEE Std 802.11n." 2009*

40	-	-	-	22	-	49	23	43	-	-	-	1	0	-	-	-	-	-	-	-	-	-	-
50	1	-	-	48	35	-	-	13	-	30	-	-	0	0	-	-	-	-	-	-	-	-	-
39	50	-	-	4	-	2	-	-	-	-	49	-	-	0	0	-	-	-	-	-	-	-	-
33	-	-	38	37	-	-	4	1	-	-	-	-	-	-	0	0	-	-	-	-	-	-	-
45	-	-	-	0	22	-	-	20	42	-	-	-	-	-	-	-	0	0	-	-	-	-	-
51	-	-	48	35	-	-	-	44	-	18	-	-	-	-	-	-	-	0	0	-	-	-	-
47	11	-	-	-	17	-	-	51	-	-	-	0	-	-	-	-	-	-	0	0	-	-	-
5	-	25	-	6	-	45	-	13	40	-	-	-	-	-	-	-	-	-	-	0	0	-	-
33	-	-	34	24	-	-	-	23	-	-	46	-	-	-	-	-	-	-	-	-	0	0	-
1	-	27	-	1	-	-	-	38	-	44	-	-	-	-	-	-	-	-	-	-	-	0	0	-
-	18	-	-	23	-	-	8	0	35	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0
49	-	17	-	30	-	-	-	34	-	-	19	1	-	-	-	-	-	-	-	-	-	-	0

Figura 2.6: Matrice per generare la parity check matrix nel caso $n = 1296$ e $k = 648$, $Z = 54$: *"IEEE Std 802.11n." 2009*

2.2.3 Margulis LDPC

Si introducono ora i codici LDPC di Margulis [12] , che sono codici regolari $(3, 6)$ aventi dimensioni $n = 2640$ e $k = 1320$. La parity check matrix può essere espressa come un array di 120×240 matrici permutazione aventi dimensione 11×11. Non si analizzerà in dettaglio come è fatto il codificatore. Per le simulazioni effettuate non risulta necessario, in quanto si è supposto di utilizzare un segnale tutto a 0.

Il punto debole di questi codici sta nel fatto che, con le tecniche di decodifica tradizionali (Sum-Product), sono molto soggetti al problema della saturazione come dimostrato in [11]. Nell'articolo appena citato vengono fatte delle simulazioni su vari codici di Margulis per vedere se saturano e i risultati vengono riportati in Figura 2.7. In questo caso la simulazione è stata fatta con canale AWGN e 7 milioni di blocchi di codice. La curva riguardante i codici di Margulis (2640,1320) utilizzati in questa tesi è quella centrale, si può notare che a 2.4 dB inizia a saturare.

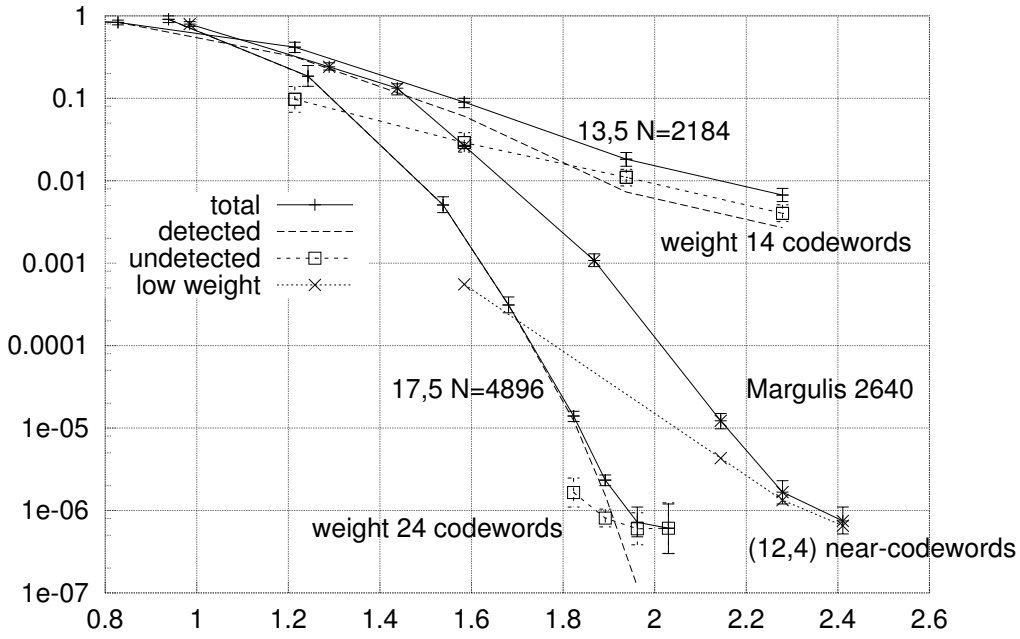


Figura 2.7: Block error rate di tre codici diversi in funzione di $\frac{E_b}{N_0}$ [dB] : *"MacKay, Postol. Weaknesses of Margulis and Ramanujan-Margulis Low-Density Parity-Check Codes." 2003*

Capitolo 3

Algoritmo Sum Product

3.1 A posteriori probability

Nel Capitolo 1 si è visto che la decodifica con il metodo maximum a posteriori (MAP) prevede un criterio di decisione del tipo

$$\hat{\mathbf{c}} = \arg\max_{\mathbf{c}\in\mathcal{C}} \; p(\mathbf{c}|\mathbf{r}) \tag{3.1}$$

Riscrivendo il segnale $\mathbf{c}$ in funzione del messaggio di ingresso al decodificatore $\mathbf{c} = E(\mathbf{u})$ si ottiene

$$\hat{\mathbf{u}} = \arg\max_{\mathbf{u}\in\mathcal{U}} p(\mathbf{c} = E(\mathbf{u})|\mathbf{r})$$

Concentrando la analisi sul l-esimo bit u_l del segnale $\mathbf{u}$ si trova

$$\begin{aligned}\hat{u_l} &= \arg\max_{u_l} p(u_l|\mathbf{r}) \\ &= \arg\max_{u_l} \sum_{\mathbf{u}\sim u_l} p(\mathbf{u}|\mathbf{r})\end{aligned}$$

Dove con $\sum_{\mathbf{u}\sim u_l}$ si intende la sommatoria tra tutti i possibili valori di $\mathbf{u}$ ottenuti fissando il simbolo l-esimo.

Si può riformulare $p(\mathbf{u}|\mathbf{r})$ come

$$\begin{aligned}p(\mathbf{u}|\mathbf{r}) &= \sum_{\mathbf{c}\in\mathcal{C}} p(\mathbf{u},\mathbf{c}|\mathbf{r}) \\ &= \sum_{\mathbf{c}\in\mathcal{C}} p(\mathbf{u}|\mathbf{c},\mathbf{r})p(\mathbf{c}|\mathbf{r}) \\ &= \sum_{\mathbf{c}\in\mathcal{C}} p(\mathbf{u}|\mathbf{c})p(\mathbf{c}|\mathbf{r}) \\ &= \sum_{\mathbf{c}\in\mathcal{C}} \mathcal{M}(\mathbf{c},\mathbf{u})p(\mathbf{c}|\mathbf{r}) \\ &= \sum_{\mathbf{c}\in\mathcal{C}} \mathcal{M}(\mathbf{c},\mathbf{u})\frac{p(\mathbf{r}|\mathbf{c})p(\mathbf{c})}{p(\mathbf{r})}\end{aligned}$$

dove si è introdotta la membership function $\mathcal{M}(\mathbf{c}, \mathbf{u}) = p(\mathbf{u}|\mathbf{c})$. Dato che la funzione da massimizzare è indipendente da $\mathbf{r}$ e che $p(\mathbf{c}) = p(\mathbf{u})$ si ha

$$\hat{\mathbf{u}} = \arg\max_{u_l} \sum_{\mathbf{c},\mathbf{u}\sim u_l} \mathcal{M}(\mathbf{c}, \mathbf{u}) p(\mathbf{r}|\mathbf{c}) p(\mathbf{u})$$

Essendo il canale AWGN e privo di memoria e assumendo i simboli i.i.d. si ottiene

$$\hat{\mathbf{u}} = \arg\max_{u_l} \sum_{\mathbf{c},\mathbf{u}\sim u_l} \mathcal{M}(\mathbf{c}, \mathbf{u}) \prod_{i=1}^{n} e^{-\frac{(r_i - c_i)^2}{2\sigma_\omega^2}} \prod_{j=1}^{k} p(u_j) \tag{3.2}$$

3.2 Marginalizzazione e factor graph

Le funzioni marginali $g_i(x_i)$ (con $i = 1, .., n$) si ottengono sommando $g(x_1, .., x_n)$ su tutte le variabili tranne che x_i ovvero

$$g_i(x_i) = \sum_{\mathbf{x}\sim x_i} g(x_1, ..., x_n) \qquad i = 1, ..., n \tag{3.3}$$

In [10] viene proposto un metodo per ricavare le funzioni marginali attraverso l'uso della rappresentazione di Tanner, che nel Capitolo 2 è stata introdotta per i codici LDPC. In questa tesi useremo una rappresentazione più compatta chiamata Forney Style factor graph (FFG), che viene spiegata di seguito.

Data una funzione $g(x_1, ..., x_n)$ che fattorizza in

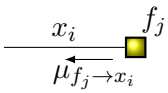
$$g(x_1, ..., x_n) = \prod f_j(X_j)$$

con f_j funzione locale e X_j sottoinsieme di $\{x_1, ..., x_n\}$, la rappresentazione FFG si ricava nel seguente modo:

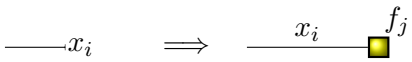
1. Disegnare un nodo (factor nodes) per ogni funzione locale f_j
2. Per ogni variabile si disegna un lato, che congiunge i factor node aventi funzioni dipendenti da essa. Nel caso in cui la variabile sia presente in una sola funzione, si disegna un semi-lato entrante nel factor node.
3. Se ci sono variabili incidenti in più di due factor node vengono replicate con dei nodi ausiliari (equality factor)

Avendo la rappresentazione grafica della funzione di partenza, la marginalizzazione si ottiene aggiornando iterativamente il messaggio uscente-entrante dai vari nodi con le seguente regole (algoritmo Sum-Product):

1. Per i nodi foglia f_j, quelli con un solo lato incidente che collega alla variabile x_i, si inizializza il messaggio uscente $\mu_{f_j \to x_i}(x_i) = f_j(x_i)$

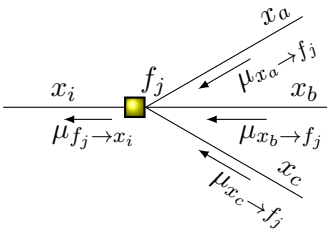


2. Per i semilati si usa la regola dei nodi foglia, con $f_j(x_i) = 1$



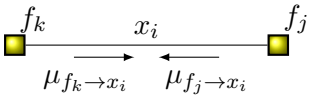
3. Per il generico nodo f_j che comunica con le variabili $x_i, x_a, x_b, x_c, ...$ il messaggio diretto a x_i si ottiene con la formula

$$\mu_{f_j \to x_i}(x_i) = \sum_{X_j \sim x_j} f_j(X_j) \mu_{x_a \to f_j}(x_a) \mu_{x_b \to f_j}(x_b) ...$$



Dopo aver trovato tutti i messaggi, le funzioni marginali si ottengono come

$$g_i(x_i) = \mu_{f_k \to x_i}(x_i) \mu_{f_j \to x_i}(x_i)$$



3.3 Decodifica di codici LDPC con algoritmo Sum-Product

La formula (3.2) esprime il fatto che decodificare il simbolo l-esimo equivale a trovare il valore che massimizza una funzione marginale, la quale può essere fattorizzata dalle funzioni $\mathcal{M}$, da $p(r_l|u_l)$ e da $p(u_l)$.

La decodifica di tipo Sum-Product seguirà quindi i seguenti passaggi:

1. Disegnare il FFG della funzione $\mathcal{M}(\mathbf{c}, \mathbf{u}) \prod_{i=1}^{n} e^{-\frac{(r_i - c_i)^2}{2\sigma_\omega^2}} \prod_{j=1}^{k} p(u_j)$

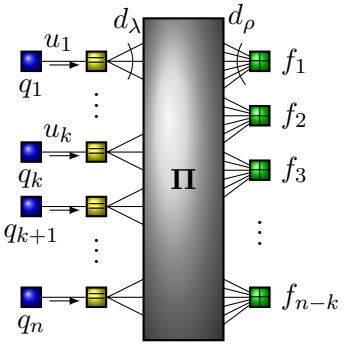


Figura 3.1: FFG per codici LDPC

2. Trovare, con l'algoritmo Sum-Product, le funzioni marginalizzate per ogni variabile u_l

3. Per ogni u_l calcolare i simboli stimati trovando il valore che massimizza la funzione marginalizzata riferita alla variabile u_l

3.3.1 FFC per la decodifica di codici LDPC

In Figura 3.1 viene riportato il Forney style Factor Graph, costruito secondo le regole enunciate nella sezione precedente, per la funzione

$$\mathcal{M}(\mathbf{c}, \mathbf{u}) \prod_{i=1}^{n} e^{-\frac{(r_i - c_i)^2}{2\sigma_\omega^2}} \prod_{j=1}^{k} p(u_j)$$

La membership function $\mathcal{M}(\mathbf{c}, \mathbf{u})$ è rappresentata dal grafo dei codici LDPC già visto nel Capitolo 2. Per quanto riguarda la seconda parte della funzione, ad ogni variable node u_l è associata una funzione $q_l(u_l)$ avente valore

$$q_l(u_l) = \begin{cases} e^{-\frac{(r_l - c_l)^2}{2\sigma_\omega^2}} p(u_l), & \text{se } l \in \{1, .., k\} \\ p(u_l), & \text{se } l \in \{k+1, .., n\} \end{cases} \tag{3.4}$$

3.3.2 Algoritmo Sum-Product applicato alla decodifica di codici LDPC

L'algoritmo per la decodifica di codici LDPC è formato da 3 fasi:

1. Inizializzazione del messaggio dei check node: Figura 3.2

2. Aggiornamento del messaggio dei variable node: Figura 3.3

3. Aggiornamento del messaggio dei check node: Figura 3.4

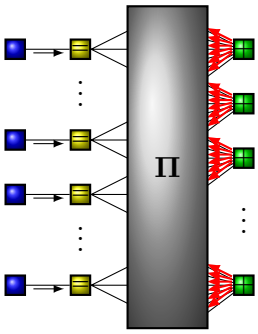


Figura 3.2: Inizializzazione variable node

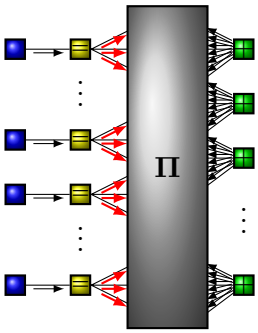


Figura 3.3: Aggiornamento dei messaggi ai check node

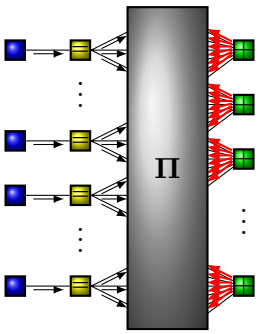
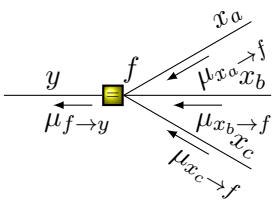


Figura 3.4: Aggiornamento dei messaggi ai variable node

I punti 2 e 3 sono ripetuti per un certo numero di volte (iterazioni) in modo da permettere all'algoritmo di convergere.

Si vede ora in maggior dettaglio in cosa consiste l'aggiornamento dei messaggi sui variable e sui check node.

Aggiornamento variable node



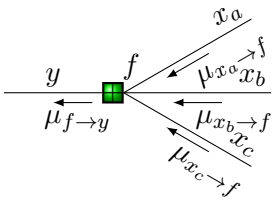
Il messaggio uscente, per le regole dell'algoritmo Sum-Product spiegato in precedenza, sarà

$$\mu_{f \to y}(y) = \sum_{x_a, x_b, x_c, \ldots} f(y, x_a, x_b, ...)\mu_{x_a \to f}(x_a)\mu_{x_b \to f}(x_b)... \tag{3.5}$$

Visto che il nodo in questione è di tipo uguaglianza si ha anche

$$\mu_{f \to y}(y) = \mu_{x_a \to f}(x_a)\mu_{x_b \to f}(x_b)\mu_{x_c \to f}(x_c)... \tag{3.6}$$

Aggiornamento check node



Anche in questo caso il messaggio uscente si calcola con la (3.5), ma questa volta

$$f(y, x_a, x_b, ...) = \delta_{y, x_a + x_b + x_c + \ldots}$$

Segnale stimato

Dopo aver aggiornato iterativamente tutti i messaggi, di tutti i nodi, la decisione sul simbolo stimato sarà fatta con la regola

$$\hat{u}_l = \arg\max_{u_l} \mu_{f_1 \to x_l}(u_l)\mu_{f_2 \to x_l}(u_l) \tag{3.7}$$

3.3.3 Decodifica con LLR

Dato che le variabili considerate sono di tipo binario scriviamo i messaggi usando le log likelihood ratios (LLR)

$$LLR = \log\left(\frac{\mu(0)}{\mu(1)}\right) \tag{3.8}$$

Si possono quindi rivedere le equazioni di aggiornamento, utilizzando le LLR, in modo da non dover calcolare tutti i messaggi per 0 e 1.

Per i variable node la (3.6) si può riformulare usando le LLR come

$$LLR_{f\to y} = \log\left(\frac{\mu_{x_a\to f}(0)\mu_{x_b\to f}(0)\mu_{x_c\to f}(0)\dots}{\mu_{x_a\to f}(1)\mu_{x_b\to f}(1)\mu_{x_c\to f}(1)\dots}\right)$$

che equivale a

$$LLR_{f\to y} = \sum_i LLR_{x_i\to f} \tag{3.9}$$

Per quanto riguarda i CN, invece, la (3.8) diventa

$$LLR_{f\to y} = log\left(\frac{\sum\limits_{x_a,x_b,x_c,\dots} \delta_{0,x_a+x_b+x_c+\dots}\mu_{x_a\to f}(x_a)\mu_{x_b\to f}(x_b)\dots}{\sum\limits_{x_a,x_b,x_c,\dots} \delta_{1,x_a+x_b+x_c+\dots}\mu_{x_a\to f}(x_a)\mu_{x_b\to f}(x_b)\dots}\right) \tag{3.10}$$

Si vuole trovare una formulazione più compatta che usi le LLR per i check node, riscriviamo la formula (3.10) nel caso in cui il check node abbia due ingressi

$$LLR_{f\to y} = \log\left(\frac{\mu_{x_a\to f}(1)\mu_{x_b\to f}(1) + \mu_{x_a\to f}(0)\mu_{x_b\to f}(0)}{\mu_{x_a\to f}(0)\mu_{x_b\to f}(1) + \mu_{x_a\to f}(1)\mu_{x_b\to f}(0)}\right) \tag{3.11a}$$

$$= \log\left(\frac{1 + \frac{\mu_{x_a\to f}(0)\mu_{x_b\to f}(0)}{\mu_{x_a\to f}(1)\mu_{x_b\to f}(1)}}{\frac{\mu_{x_a\to f}(0)}{\mu_{x_a\to f}(1)} + \frac{\mu_{x_b\to f}(0)}{\mu_{x_b\to f}(1)}}\right) \tag{3.11b}$$

$$= \log\left(\frac{1 + e^{LLR_{x_a\to f}+LLR_{x_b\to f}}}{e^{LLR_{x_a\to f}} + e^{LLR_{x_b\to f}}}\right) \tag{3.11c}$$

Introducendo la funzione

$$\varphi(x) = \frac{e^x - 1}{e^x + 1} = \tanh\frac{x}{2}$$

Si può pensare ad una formulazione più compatta per la (3.11c), infatti si trova facilmente che

$$\varphi(LLR_{f\to y}) = \varphi(LLR_{x_a\to f})\varphi(LLR_{x_b\to f})$$

di conseguenza

$$LLR_{f\to y} = \varphi^{-1}(\varphi(LLR_{x_a\to f})\varphi(LLR_{x_b\to f}))$$

In generale nel caso in cui si abbia un numero qualsiasi di variabili si ha

$$LLR_{f\to y} = \varphi^{-1}\left(\prod_i \varphi(LLR_{x_i\to f})\right) \tag{3.12}$$

La (3.12) contiene un elevato numero di moltiplicazioni e questo può aumentare la complessità computazione, per risolvere questo problema di introduce un'altra funzione

$$\tilde{\varphi}(x) = -\log\varphi(x) = \tilde{\varphi}^{-1}(x)$$

Di conseguenza la (3.12) diventa

$$LLR_{f\to y} = \tilde{\varphi}\left(\sum_i \tilde{\varphi}(|LLR_{x_i\to f}|)\right)\prod_i sign(LLR_{x_i\to f}) \tag{3.13}$$

A questo punto si sono introdotte sia la regola di aggiornamento per i variable node (3.9) che per i check node (3.13), il criterio di decisione (3.7) nel caso LLR è quindi del tipo

$$\begin{cases} 0, & \text{se } LLR_{f_1\to x_l} + LLR_{f_2\to x_l} > 0 \\ 1, & \text{altrimenti} \end{cases} \tag{3.14}$$

Capitolo 4

Decodificatore PL di Vontobel-Koetter

Viene ora introdotto un algoritmo per la decodifica ML di codici LDPC, che risolve il problema di programmazione lineare (PL) introdotto nel Capitolo 1.

4.1 Notazione usata

Sia dato un codice LDPC avente parity check matrix $\mathbf{H}$ di dimensione $n-k \times n$, con k lunghezza sequenza in ingresso al codificatore e n lunghezza sequenza decodificata.

Si definiscono gli insiemi

$$\mathcal{I} \triangleq \{1, ..., n\} \qquad \mathcal{J} \triangleq \{1, ..., n-k\}$$

$$\mathcal{I}_j \triangleq \{i \in \mathcal{I} | [\mathbf{H}_{j,i} = 1]\},\ j \in \mathcal{J} \qquad \mathcal{J}_i \triangleq \{j \in \mathcal{J} | [\mathbf{H}_{j,i} = 1]\},\ i \in \mathcal{I}$$

Sia $\mathbf{h}_j$ la j-esima riga di $\mathbf{H}$, per ogni check node ($j \in \mathcal{J}$) l'insieme delle possibili parole di codice risulta essere

$$\mathcal{C}_j \triangleq \{\mathbf{x} \in \mathbb{F}_2^n | \mathbf{h}_j \mathbf{x}^T = 0\}$$

Infine, per coerenza con la notazione usata in [15], il segnale codificato verrà chiamato $\mathbf{x}$ (invece di $\mathbf{c}$) e quello ricevuto $\mathbf{y}$ (invece di $\mathbf{r}$).

4.2 Problema primale

Come già discusso nel Capitolo 1, la decodifica di tipo ML prevede la risoluzione del problema di programmazione lineare

$$\text{Minimize} \quad \sum_{i \in \mathcal{I}} \gamma_i x_i \tag{4.1a}$$

$$\text{Subject to} \quad \mathbf{x} \in \mathcal{C} \tag{4.1b}$$

Dato che $\mathcal{C} = \mathcal{C}_1 \cap ... \cap \mathcal{C}_m$ e di conseguenza $conv(\mathcal{C}) = conv(\mathcal{C}_1) \cap ... \cap conv(\mathcal{C}_m)$ Feldman,Wainright e Karger in [7] dimostrano che invece di risolvere la (4.1) si può trovare la soluzione del problema sub-ottimo

$$\text{Minimize} \quad \sum_{i \in \mathcal{I}} \gamma_i x_i \tag{4.2a}$$

$$\text{Subject to} \quad \mathbf{x} \in \text{conv}(\mathcal{C}_\text{j}) \qquad \forall \text{j} \in \mathcal{J} \tag{4.2b}$$

Il problema (4.2) può essere riformulato, introducendo delle variabili ausiliarie, in modo da riflettere le caratteristiche della rappresentazione grafica degli LDPC tramite FFG (Forney style factor graphs).

$$\text{Minimize} \quad \sum_{i \in \mathcal{I}} \gamma_i x_i \tag{4.3a}$$

$$\text{Subject to} \quad x_i = u_{i,0} \qquad i \in \mathcal{I} \tag{4.3b}$$

$$u_{i,j} = v_{j,i} \qquad i \in \mathcal{I},\, j \in \mathcal{J}_i \tag{4.3c}$$

$$\sum_{\mathbf{a}_i \in \mathbf{A}_i} \alpha_{\mathbf{a}_i} \mathbf{a}_i = \mathbf{u}_i \qquad i \in \mathcal{I} \tag{4.3d}$$

$$\sum_{\mathbf{b}_j \in \mathbf{B}_j} \beta_{\mathbf{b}_j} \mathbf{b}_j = \mathbf{v}_j \qquad j \in \mathcal{J} \tag{4.3e}$$

$$\alpha_{\mathbf{a}_i} \geq 0 \qquad i \in \mathcal{I},\, \mathbf{a}_i \in \mathbf{A}_i \tag{4.3f}$$

$$\beta_{\mathbf{b}_j} \geq 0 \qquad j \in \mathcal{J},\, \mathbf{b}_j \in \mathbf{B}_j \tag{4.3g}$$

$$\sum_{\mathbf{a}_i \in \mathbf{A}_i} \alpha_{\mathbf{a}_i} = 1 \qquad i \in \mathcal{I} \tag{4.3h}$$

$$\sum_{\mathbf{b}_j \in \mathbf{B}_j} \beta_{\mathbf{b}_j} = 1 \qquad j \in \mathcal{J} \tag{4.3i}$$

Le variabili $u_{i,j}$, con $i \in \mathcal{I}$, $j \in \{0\} \cup \mathcal{J}_i$, rappresentano l'uscita (per $j \in \mathcal{J}_i$) e l'ingresso (per $j = 0$) dell'i-esimo variable node. Le $v_{j,i}$, con $j \in \mathcal{J}$, $i \in \mathcal{I}_j$, sono invece l' ingresso al j-esimo check node. I vettori $\mathbf{a}_i \in \mathbf{A}_i$ e $\mathbf{b}_j \in \mathbf{B}_j$ sono le combinazioni che possono assumere rispettivamente $\mathbf{u}_i$ e $\mathbf{v}_j$. Nelle equazioni (4.3d), (4.3f) e (4.3h) si assegna ad $\mathbf{u}_i$ uno tra i possibili valori di $\mathbf{a}_i$, mettendo $\alpha_{\mathbf{a}_i} = 1$ per questo valore e $\alpha_{\mathbf{a}_i} = 0$ per gli altri $\mathbf{a}_i$. Nelle equazioni (4.3e), (4.3g) e (4.3i) si fa la stessa cosa per $\mathbf{v}_j$, usando le variabili $\mathbf{b}_j$ e $\beta_{\mathbf{b}_j}$.

4.3 Problema duale

Vediamo ora come definire il problema duale del primale proposto in (4.3). Per prima cosa si scrive il Langrangian del problema (4.3) come spiegato nel Capitolo 1.

$$
\begin{aligned}
L = &\sum_{i\in\mathcal{I}} \gamma_i x_i + \sum_{i\in\mathcal{I}} C_i(x_i - u_{i,0}) + \sum_{\substack{i\in\mathcal{I}\\ j\in\mathcal{J}_i}} D_{i,j}(u_{i,j} - v_{j,i}) \\
&+ \sum_{\substack{i\in\mathcal{I}\\ j\in\mathcal{J}_i\cup\{0\}}} E_{i,j}(u_{i,j} - \sum_{\mathbf{a}_i\in\mathbf{A}_i} \alpha_{\mathbf{a}_i}\mathbf{a}_{i,j}) + \sum_{\substack{j\in\mathcal{J}\\ i\in\mathcal{I}_j}} F_{j,i}(v_{j,i} - \sum_{\mathbf{b}_j\in\mathbf{B}_j} \beta_{\mathbf{b}_j}\mathbf{b}_{j,i}) \\
&+ \sum_{\substack{i\in\mathcal{I}\\ \mathbf{a}_i\in\mathbf{A}_i}} G_{i,\mathbf{a}_i}(-\alpha_{i,\mathbf{a}_i}) + \sum_{\substack{j\in\mathcal{J}\\ \mathbf{b}_j\in\mathbf{B}_j}} H_{j,\mathbf{b}_j}(-\beta_{j,\mathbf{b}_j}) + \sum_{i\in\mathcal{I}} K_i(1 - \sum_{\mathbf{a}_i\in\mathbf{A}_i} \alpha_{\mathbf{a}_i}) \\
&+ \sum_{j\in\mathcal{J}} L_j(1 - \sum_{\mathbf{b}_b\in\mathbf{B}_j} \beta_{\mathbf{b}_j})
\end{aligned}
$$

dove $G_{i,\mathbf{a}_i} \geq 0$, $H_{j,\mathbf{b}_j} \geq 0$. Questo risultato può essere riformulato come

$$
\begin{aligned}
L = &\sum_{i\in\mathcal{I}} \gamma_i x_i + \sum_{i\in\mathcal{I}} C_i x_i - \sum_{i\in\mathcal{I}} C_i u_{i,0} + \sum_{\substack{i\in\mathcal{I}\\ j\in\mathcal{J}_i}} D_{i,j} u_{i,j} + \sum_{\substack{i\in\mathcal{I}\\ j\in\mathcal{J}_i\cup\{0\}}} E_{i,j} u_{i,j} \\
&- \sum_{\substack{i\in\mathcal{I}\\ j\in\mathcal{J}_i}} D_{i,j} v_{j,i} + \sum_{\substack{j\in\mathcal{J}\\ i\in\mathcal{I}_j}} F_{j,i} v_{j,i} - \sum_{\substack{i\in\mathcal{I}\\ j\in\mathcal{J}_i\cup\{0\}}} E_{i,j} \sum_{\mathbf{a}_i\in\mathbf{A}_i} \alpha_{\mathbf{a}_i}\mathbf{a}_{i,j} \\
&- \sum_{\substack{i\in\mathcal{I}\\ \mathbf{a}_i\in\mathbf{A}_i}} G_{i,\mathbf{a}_i}\alpha_{i,\mathbf{a}_i} - \sum_{i\in\mathcal{I}} K_i \sum_{\mathbf{a}_i\in\mathbf{A}_i} \alpha_{\mathbf{a}_i} - \sum_{\substack{j\in\mathcal{J}\\ i\in\mathcal{I}}} F_{j,i} \sum_{\mathbf{b}_j\in\mathbf{B}_j} \beta_{\mathbf{b}_j}\mathbf{b}_{j,i} \\
&- \sum_{\substack{j\in\mathcal{J}\\ \mathbf{b}_j\in\mathbf{B}_j}} H_{j,\mathbf{b}_j}\beta_{j,\mathbf{b}_j} - \sum_{j\in\mathcal{J}} L_j \sum_{\mathbf{b}_b\in\mathbf{B}_j} \beta_{\mathbf{b}_j} + \sum_{i\in\mathcal{I}} K_i + \sum_{j\in\mathcal{J}} L_j
\end{aligned}
$$

raccogliendo si ottiene

$$
\begin{aligned}
L = &\sum_{i\in\mathcal{I}} K_i + \sum_{j\in\mathcal{J}} L_j + \sum_{i\in\mathcal{I}} x_i(\gamma_i + C_i) + \sum_{i\in\mathcal{I}} u_{i,0}(E_{i,0} - C_i) + \sum_{\substack{i\in\mathcal{I}\\ j\in\mathcal{J}_i}} u_{i,j}(D_{i,j} + E_{i,j}) \\
&+ \sum_{\substack{i\in\mathcal{I}\\ j\in\mathcal{J}_i}} v_{j,i}(F_{j,i} - D_{i,j}) - \sum_{i\in\mathcal{I}} \sum_{\mathbf{a}_i\in\mathbf{A}_i} \alpha_{\mathbf{a}_i}(G_{i,\mathbf{a}_i} + K_i + \sum_{j\in\mathcal{J}_i\cup\{0\}} E_{i,j}\mathbf{a}_{i,j}) \\
&- \sum_{j\in\mathcal{J}} \sum_{\mathbf{b}_j\in\mathbf{B}_j} \beta_{\mathbf{b}_j}(H_{j,\mathbf{b}_j} + L_j + \sum_{i\in\mathcal{I}_j} F_{j,i}\mathbf{b}_{j,i})
\end{aligned}
$$

Imponendo le condizioni $C_i = E_{i,0} = -\gamma_i$, $D_{i,j} = -E_{i,j}$ e $D_{i,j} = F_{j,i}$, il problema duale (Lagrange dual problem) diventa

$$
\begin{aligned}
\text{Maximize} \quad & \sum_{i\in\mathcal{I}} K_i + \sum_{j\in\mathcal{J}} L_j \\
\text{Subject to} \quad & (G_{i,\mathbf{a}_i} + K_i + \sum_{j\in\mathcal{J}_i\cup\{0\}} E_{i,j}\mathbf{a}_{i,j}) = 0 && i\in\mathcal{I},\ \mathbf{a}_i\in\mathbf{A}_i \\
& (H_{j,\mathbf{b}_j} + L_j + \sum_{i\in\mathcal{I}_j} F_{j,i}\mathbf{b}_{j,i}) = 0 && j\in\mathcal{J},\ \mathbf{b}_j\in\mathbf{B}_j \\
& G_{i,\mathbf{a}_i} \geq 0 && i\in\mathcal{I},\ \mathbf{a}_i\in\mathbf{A}_i \\
& H_{j,\mathbf{b}_j} \geq 0 && j\in\mathcal{J},\ \mathbf{b}_j\in\mathbf{B}_j \\
& E_{i,j} = -F_{j,i} && i\in\mathcal{I},\ j\in\mathcal{J} \\
& E_{i,0} = -\gamma_i && i\in\mathcal{I}
\end{aligned}
$$

il che equivale a

$$
\begin{aligned}
\text{Maximize} \quad & \sum_{i\in\mathcal{I}} K_i + \sum_{j\in\mathcal{J}} L_j && && (4.4a) \\
\text{Subject to} \quad & K_i \leq \min_{\mathbf{a}_i\in\mathbf{A}_i} \langle -\mathbf{E}_i, \mathbf{a}_i\rangle && i\in\mathcal{I} && (4.4b) \\
& L_i \leq \min_{\mathbf{b}_j\in\mathbf{B}_j} \langle -\mathbf{F}_j, \mathbf{b}_j\rangle && j\in\mathcal{J} && (4.4c) \\
& E_{i,j} = -F_{j,i} && i\in\mathcal{I},\ j\in\mathcal{J} && (4.4d) \\
& E_{i,0} = -\gamma_i && i\in\mathcal{I} && (4.4e)
\end{aligned}
$$

Il problema (4.4) diventa

$$
\begin{aligned}
\text{Maximize} \quad & \sum_{i\in\mathcal{I}} \min_{\mathbf{a}_i\in\mathbf{A}_i} \langle -\mathbf{E}_i, \mathbf{a}_i\rangle + \sum_{j\in\mathcal{J}} \min_{\mathbf{b}_j\in\mathbf{B}_j} \langle -\mathbf{F}_j, \mathbf{b}_j\rangle && && (4.5a) \\
\text{Subject to} \quad & E_{i,j} = -F_{j,i} && i\in\mathcal{I},\ j\in\mathcal{J} && (4.5b) \\
& E_{i,0} = -\gamma_i && i\in\mathcal{I} && (4.5c)
\end{aligned}
$$

4.4 Algoritmo di decodifica

Vontobel e Kotter in [15] propongono un algoritmo per risolvere il problema (4.5): Si scelgono i lati (i, j) del grafo seguendo qualche schema (ad esempio ciclicamente) e si aggiorna la variabile $E_{i,j}$ in modo da aumentare la funzione obiettivo del problema duale.

Definendo la funzione

$$h(E_{i,j,t}) = \min_{\mathbf{a}_i\in\mathbf{A}_i} \langle -\mathbf{E}_{i,t}\mathbf{a}_i\rangle + \min_{\mathbf{b}_j\in\mathbf{B}_j} \langle -\mathbf{F}_{j,t}, \mathbf{b}_j\rangle$$

dove si è introdotto l'indice t dell'iterazione, si ottiene che

$$E_{i,j,t+1} = \arg\max_{E_{i,j,t}} h(E_{i,j,t}) \tag{4.6}$$

In [15] si dimostra che un valore di $E_{i,j,t+1}$ che soddisfi la 4.6 è dato da

$$E_{i,j,t+1} = \frac{1}{2}\Big[(S_{i,0,t} - S_{i,1,t}) - (T_{j,0,t} - T_{j,1,t})\Big] \tag{4.7}$$

dove

$$S_{i,0,t} \triangleq -\min_{\substack{\mathbf{a}_i \in \mathbf{A}_i \\ a_{i,j}=0}} \left\langle -\tilde{\mathbf{E}}_{i,t}\tilde{\mathbf{a}}_i \right\rangle \qquad T_{j,0,t} \triangleq -\min_{\substack{\mathbf{b}_j \in \mathbf{B}_j \\ b_{j,i}=0}} \left\langle -\tilde{\mathbf{F}}_{j,t}\tilde{\mathbf{b}}_j \right\rangle$$

$$S_{i,1,t} \triangleq -\min_{\substack{\mathbf{a}_i \in \mathbf{A}_i \\ a_{i,j}=1}} \left\langle -\tilde{\mathbf{E}}_{i,t}\tilde{\mathbf{a}}_i \right\rangle \qquad T_{j,1,t} \triangleq -\min_{\substack{\mathbf{b}_j \in \mathbf{B}_j \\ b_{j,i}=1}} \left\langle -\tilde{\mathbf{F}}_{j,t}\tilde{\mathbf{b}}_j \right\rangle$$

I vettori $\tilde{\mathbf{a}}_i$ e $\tilde{\mathbf{E}}_{i,t}$ sono i vettori $\mathbf{a}_i$ e $\mathbf{E}_{i,t}$, ma senza il valore nella j-esima posizione, nel caso di $\tilde{\mathbf{b}}_j$ e $\tilde{\mathbf{F}}_{j,t}$, invece, è stato omesso il valore in posizione i.

Si può notare che in realtà $S_{i,0,t} = 0$, essendo $\mathbf{a}_i = \mathbf{0}$ nel caso in cui un suo elemento sia a 0.

Dopo aver aggiornato ciclicamente le variabili $E_{i,j}\ \forall(i,j)$ un certo numero di volte, il segnale stimato $\hat{\mathbf{x}}$ si trova con la seguente regola:

$$\hat{x}_i = \begin{cases} 0, & \text{se } \sum_{j=1}^{m} -E_{i,j} \geq 0 \\ 1, & \text{se } \sum_{j=1}^{m} -E_{i,j} < 0 \end{cases} \tag{4.9}$$

La (4.9), in pratica, esprime come passare dalla soluzione del problema duale a quella del primale.

4.5 Risultati articolo

Vontobel e Koetter, nel loro articolo, simulano l'algoritmo per dei codici LDPC regolari (3,6), con $k = 500$ e $n = 1000$. La parity check matrix viene generata in modo casuale e, nel relativo grafico di Tanner, vengono eliminati i cicli di lunghezza 4.

L'algoritmo usato nell'articolo per le simulazione, in realtà, non è esattamente quello descritto sopra, ma ha alcune differenze:

- Invece di calcolare il minimo si usa una funzione approssimata

$$\min_{l}^{(k)} z_l \triangleq -\frac{1}{k}\log\left(\sum_{l} e^{-kz_l}\right) \tag{4.10}$$

  dove k è una costante. Si può dimostrare che per $k \to \infty$ si ha $\min_l^{(k)} z_l \to \min_l z_l$

- L'aggiornamento di $E_{i,j}$ viene fatto prendendo un valore casuale nell'intervallo $\big[(S_{i,0} - S_{i,1}), -(T_{i,0} - T_{i,1})\big]$

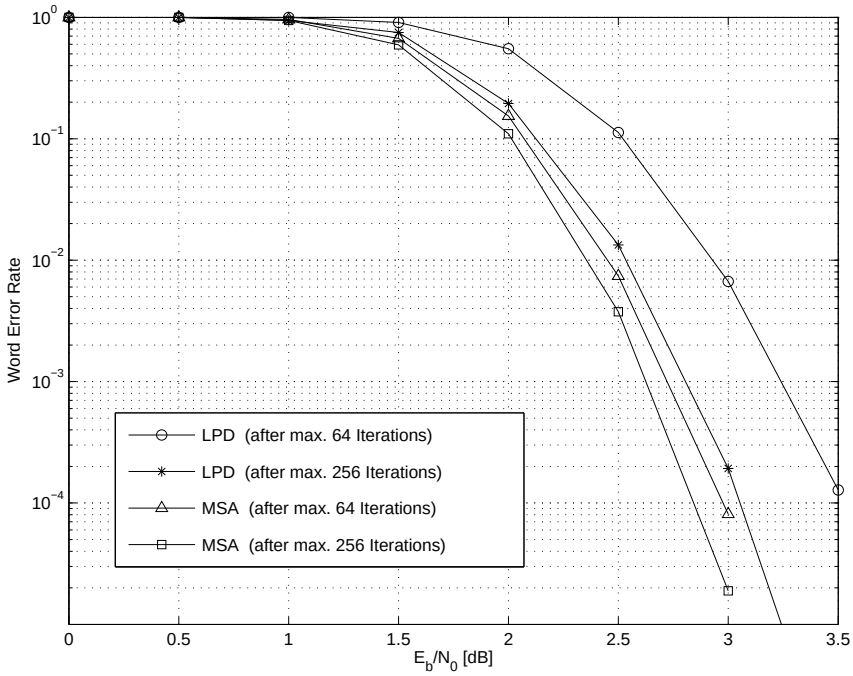


Figura 4.1: Decodifica di codici LDPC regolari (3,6): *Vontobel,Koetter. "On low-complexity linear programming decoding of LDPC codes." 2006*

I risultati ottenuti sono riportati in Figura 4.1. Viene fatto il confronto tra il Min-Sum-Algorithms MSA (una variante del Sum-Product) e l'algoritmo descritto in questo capitolo, denominato LPD (Linear Programming Decoding). In entrambi i casi si fa la simulazione sia con 64 che con 256 iterazioni e vengono tracciati i grafici della Word Error Rate (WER) in funzione del valore di $\frac{E_b}{N_0}$.

Nel Capitolo 6 verranno fatte delle altre simulazioni di questo algoritmo, comunque dalla Figura 4.1 si possono già fare delle osservazioni: Per questo tipo di codice, in particolare nel caso 256 iterazioni, la WER dell'algoritmo LPD risulta abbastanza vicina a quella del MSA, la distanza aumenta nel caso 64 iterazioni. Tuttavia, si vedrà in seguito, che l'algoritmo LPD risulta molto più lento del Sum-Product tradizionale e quindi, avere un numero di iterazioni elevato, risulta problematico.

4.6 Algoritmo di Burshtein

Nel suo articolo [5] Bushtein propone, partendo dall'implementazione di Vontobel e Koetter, una versione alternativa dell'algoritmo.

4.6.1 Notazione usata

Viene usata una notazione leggermente diversa rispetto all'articolo di Vontobel-Koetter. L'insieme dei vicini del variable node i viene chiamato $\mathcal{N}_i$ invece di $\mathcal{J}_i$, quello del check node j viene denominato $\mathcal{N}_j$ al posto di $\mathcal{I}_j$. Si definisce poi ε_j come l'insieme

$$\varepsilon_j \triangleq \{S \subseteq \mathcal{N}_j : |S|\, pari\} \tag{4.11}$$

In pratica tutti i possibili sottoinsiemi del j-esimo check node, contenenti un numero pari di elementi. In questo modo le possibili combinazioni, che soddisfano il vincolo del j-esimo check node, sono quelle ottenibili mettendo a 1 i vicini in S e a 0 gli altri.

Si definisce poi la funzione $\delta_S(i)$ tale che

$$\delta_S(i) = \begin{cases} 1, & \text{se } i \in S \\ 0, & \text{se } i \notin S \end{cases}$$

4.6.2 Riformulazione problema Vontobel-Koetter

Il problema duale di Vontobel Koetter (4.5), usando la nuova notazione, diventa

$$\text{Maximize} \quad \sum_{j \in \mathcal{J}} \min_{S \in \varepsilon_j} \Big\{ \sum_{i \in \mathcal{N}_j} u_{i,j} \delta_S(i) \Big\} \tag{4.12a}$$

$$\text{Subject to} \sum_{j \in \mathcal{N}_i} u_{i,j} = \gamma_i, \qquad \forall i \in \mathcal{I} \tag{4.12b}$$

La variabile da massimizzare in questo caso è chiamata $\mathbf{u} \triangleq \{u_{i,j}\}_{i \in \mathcal{I}, j \in \mathcal{N}_j}$ e non $\mathbf{E}$ come nella (4.5).

Usando l'approssimazione già utilizzata da Vontobel e Koetter

$$\min(x_1, x_2, ..., x_m) \simeq -\frac{1}{K} \log \sum_{i=1}^{m} e^{-Kx_i}$$

con K costante, si può riscrivere la (4.12) come

$$\text{Maximize} \quad -\frac{1}{K} \sum_{j \in \mathcal{J}} \log \sum_{S \in \varepsilon_j} e^{-K \sum\limits_{i \in \mathcal{N}_j} u_{i,j} \delta_S(i)}$$

$$\text{Subject to} \sum_{j \in \mathcal{N}_i} u_{i,j} = \gamma_i, \qquad \forall i \in \mathcal{I}$$

Invertendo il segno alla funzione obiettivo si trova

$$\text{Mimimize} \quad \frac{1}{K} \sum_{j \in \mathcal{J}} \log \sum_{S \in \varepsilon_j} e^{-K \sum\limits_{i \in \mathcal{N}_j} u_{i,j} \delta_S(i)} \tag{4.14a}$$

$$\text{Subject to} \sum_{j \in \mathcal{N}_i} u_{i,j} = \gamma_i, \qquad \forall i \in \mathcal{I} \tag{4.14b}$$

Il Lagrangian del problema (4.14) rispetto al variable node i vale

$$L_i(\mathbf{u}, \lambda_i) = \frac{1}{K} \sum_{j \in \mathcal{N}_i} \log \sum_{S \in \varepsilon_j} e^{-K \sum_{i \in \mathcal{N}_j} u_{i,j} \delta_S(i)} + \lambda_i \left(\sum_{j \in \mathcal{N}_i} u_{i,j} - \gamma_i \right) \tag{4.15}$$

In [5] Burshtein dimostra che, derivando il Lagrangian e mettendolo a zero, il valore di $u_{k,j}$ ottimo per il problema vale

$$u_{k,j} = v_{k,j} + \frac{\gamma_k}{|\mathcal{N}_k|} - \frac{1}{|\mathcal{N}_k|} \sum_{i \in \mathcal{N}_k} v_{k,i}, \qquad \forall j \in \mathcal{N}_k \tag{4.16}$$

dove

$$v_{k,j} = -\frac{1}{K} \log \frac{\prod_{i \in \mathcal{N}_{j \setminus k}} (1 + e^{-K u_{i,j}}) + \prod_{i \in \mathcal{N}_{j \setminus k}} (1 - e^{-K u_{i,j}})}{\prod_{i \in \mathcal{N}_{j \setminus k}} (1 + e^{-K u_{i,j}}) - \prod_{i \in \mathcal{N}_{j \setminus k}} (1 - e^{-K u_{i,j}})} \tag{4.17}$$

La (4.17) può essere scritta in modo più compatto

$$v_{k,j} = \frac{1}{K} \log \frac{1 - l_{k,j}}{1 + l_{k,j}} \tag{4.18}$$

dove si è introdotta la variabile

$$l_{k,j} = \prod_{i \in \mathcal{N}_{j \setminus k}} \frac{1 - e^{-K u_{i,j}}}{1 + e^{-K u_{i,j}}}$$

4.6.3 Algortimo proposto

L'algoritmo proposto nell'articolo di Burshtein viene riportato in Figura 4.2. Dopo una prima fase di inizializzazione, l'algoritmo sceglie un variable node i e aggiorna prima le variabili $u_{i,j}$ con la regola (4.16) e successivamente $\lambda_{i,j}$, λ_i e ϵ_i definite rispettivamente come

$$\lambda_{i,j} \triangleq \frac{1}{1 + e^{K(u_{i,j} - v_{i,j})}}, \qquad \lambda_i \triangleq \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \lambda_{i,j}, \qquad \epsilon_i \triangleq \max_{j \in \mathcal{N}_i} |\lambda_{i,j} - \lambda_i|$$

Dopo aver svolto tutte le iterazioni dell'algoritmo e, introducendo la variabile $\epsilon \triangleq \max_{i \in \mathcal{I}} \epsilon_i$, il segnale decodificato si trova con la regola

$$\hat{c}_i = \begin{cases} 1, & \text{se } \lambda_i(1 - 6\epsilon) + 3\epsilon \geq 0.5 \\ 0, & \text{se } \lambda_i(1 - 6\epsilon) + 3\epsilon < 0.5 \end{cases} \tag{4.19}$$

I parametri dell'algoritmo sono K ed ϵ_0: Il primo dipende dall'approssimazione della funzione minimo, mentre il secondo determina il numero di iterazioni dell'algoritmo. Se ϵ_0 è grande l'algoritmo farà poche iterazioni e,

1) **Initialization**:

$$\forall i \in \mathcal{I}, j \in \mathcal{N}_i : u_{i,j} = \gamma_i / |\mathcal{N}_i|$$
$$\forall i \in \mathcal{I}, j \in \mathcal{N}_i : v_{i,j} = \frac{1}{K} \log \frac{1 - l_{i,j}}{1 + l_{i,j}}$$

$$\forall i \in \mathcal{I}, j \in \mathcal{N}_i : \lambda_{i,j} = \frac{1}{1 + e^{K(u_{i,j} - v_{i,j})}}$$
$$\forall i \in \mathcal{I} : \lambda_i = \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \lambda_{i,j}$$
$$\forall i \in \mathcal{I} : \epsilon_i = \max_{j \in \mathcal{N}_i} |\lambda_{i,j} - \lambda_i|$$
$$\mathcal{A} = \{i \in \mathcal{I} : \epsilon_i \geq \epsilon_0\}.$$

2) **Iteration**: Pick an arbitrary element $k \in \mathcal{A}$ and make the following updates:

$$\forall j \in \mathcal{N}_k : u_{k,j} = v_{k,j} + \frac{\gamma_k}{|\mathcal{N}_k|} - \frac{1}{|\mathcal{N}_k|} \sum_{l \in \mathcal{N}_k} v_{k,l}$$
$$\forall i \in \mathcal{N}_j \setminus k, j \in \mathcal{N}_k : v_{i,j} = \frac{1}{K} \log \frac{1 - l_{i,j}}{1 + l_{i,j}}$$

$$\forall i \in \mathcal{N}_j, j \in \mathcal{N}_k : \lambda_{i,j} = \frac{1}{1 + e^{K(u_{i,j} - v_{i,j})}}$$
$$\forall i \in \mathcal{N}_j, j \in \mathcal{N}_k : \lambda_i = \frac{1}{|\mathcal{N}_i|} \sum_{j' \in \mathcal{N}_i} \lambda_{i,j'}$$
$$\forall i \in \mathcal{N}_j, j \in \mathcal{N}_k : \epsilon_i = \max_{j' \in \mathcal{N}_i} |\lambda_{i,j'} - \lambda_i|$$
$$\forall i \in \mathcal{N}_j, j \in \mathcal{N}_k : \text{If } i \in \mathcal{A} \text{ and } \epsilon_i < \epsilon_0 \text{ then } \mathcal{A} = \mathcal{A} \setminus i$$
$$\forall i \in \mathcal{N}_j, j \in \mathcal{N}_k : \text{If } i \notin \mathcal{A} \text{ and } \epsilon_i \geq \epsilon_0 \text{ then } \mathcal{A} = \mathcal{A} \cup i.$$

3) **Loop Control**: If $\mathcal{A} \neq \emptyset$ then repeat step 2. Otherwise proceed to step 4.
4) **Produce the final solution and Exit**:

Figura 4.2: Algoritmo di decodifica: *Burshtein. "Iterative Approximate Linear Programming Decoding of LDPC Codes With Linear Complexity." 2009*

di conseguenza, il segnale decodificato potrebbe contenere un numero elevato di errori. Invece, scegliendo ϵ_0 più piccolo del dovuto, potrebbero essere necessarie troppe iterazioni. Nelle simulazioni fatte nell'articolo, viene scelto $K = 1000$ oppure 10000 e $\epsilon_0 = 0.0001$ o 0.00001.

In questa tesi è stata implementata anche la simulazione dell'algoritmo di Burshtein riportato in Figura 4.2. Visto l'uso di esponenziali aventi come parametro la costante K, che può assumere valori anche elevati, per evitare problemi di instabilità numerica sono state usate alcune approssimazioni proposte anche nell'articolo di Burshtein.

Si sceglie di non riportare i dettagli implementativi e le simulazioni di tale algoritmo per i seguenti motivi: Per prima cosa si è trovato che con l'algoritmo di Burshtein i risultati erano molto simili a quelli dell'algoritmo di Vontobel-Koetter. Si è visto poi che questo algoritmo è abbastanza veloce per SNR molto alti, quando ci sono pochi errori, ma per SNR più bassi la convergenza diventa problematica e ci vuole troppo tempo. Di conseguenza tra i due algoritmi si è preferito focalizzare l'attenzione su quello di Vontobel-Koetter.

Capitolo 5

Decodifica con ADMM

5.1 Introduzione ADMM

Il metodo Alternating Direction Multiplier Method (ADMM), spiegato con maggior dettaglio in [3], risolve un problema del tipo

$$(\mathbf{x}^*, \mathbf{z}^*) = \underset{\substack{\mathbf{x}\in C_1 \\ \mathbf{z}\in C_2 \\ \mathbf{A}\mathbf{x}=\mathbf{b}}}{\arg\min}\, G_1(\mathbf{x}) + G_2(\mathbf{z}) \tag{5.1}$$

Per fare questo si considera lo augmented Lagrangian

$$L(\mathbf{x}, \mathbf{z}, \mathbf{p}) = G_1(\mathbf{x}) + G_2(\mathbf{z}) + \mathbf{p}^T(\mathbf{A}\mathbf{x} - \mathbf{z}) + \frac{c}{2}||\mathbf{A}\mathbf{x} - \mathbf{z}||^2 \tag{5.2}$$

dove $c > 0$ è una costante positiva, che ha influenza sul tempo che impiega l'algoritmo a convergere.

La ricerca di un punto ottimo, che soddisfi i vincoli, viene fatta fissando i valori iniziali $\mathbf{z}_0$, $\mathbf{p}_0$ e aggiornando iterativamente le variabili secondo le seguenti regole:

$$\mathbf{x}_{t+1} = \underset{\mathbf{x}\in\mathcal{C}_1}{\arg\min}\ L(\mathbf{x}, \mathbf{z}_t, \mathbf{p}_t) \tag{5.3a}$$

$$\mathbf{z}_{t+1} = \underset{\mathbf{z}\in\mathcal{C}_2}{\arg\min}\ L(\mathbf{x}_{t+1}, \mathbf{z}, \mathbf{p}_t) \tag{5.3b}$$

$$\mathbf{p}_{t+1} = \mathbf{p}_t + c(\mathbf{A}\mathbf{x}_{t+1} - \mathbf{z}_{t+1}) \tag{5.3c}$$

5.2 Algoritmo BLDR

Barman, Liu, Draper, Recht, in [2] propongono un algoritmo per la decodifica di codici LDPC con il metodo ADMM, il quale verrà chiamato BLDR dalle iniziali dei suoi autori.

In questa tesi verrà riformulato l'algoritmo in modo leggermente diverso rispetto all'articolo. Si vuole un algoritmo, che come il Sum-Product, sfrutti la struttura grafica dei codici LDPC e aggiorni i messaggi uscenti dai vari nodi.

5.2.1 Notazione usata

Si definiscono $\mathcal{A}_i$, $i = 1, ..., n-k$ e $\mathcal{B}_j$, $j = 1, ..., n$ rispettivamente gli insiemi degli elementi non nulli della riga i-esima e della colonna j-esima di $\mathbf{H}$.

Sia poi $\mathcal{C}_i = \{\mathbf{c} \in \mathbb{F}_2^n | \sum_{i \in \mathcal{A}_i} c_i = 0\}$, $i = 1 \dots n-k$. Dato l'insieme delle parole di codice $\mathcal{C} = \{\mathbf{c} \in \mathbb{F}_2^n | \mathbf{Hc} = \mathbf{0}\}$, vale la relazione $\mathcal{C} = \cap_i \mathcal{C}_i$, di conseguenza per l'inviluppo convesso conv($\mathcal{C}$) si ha

$$\mathrm{conv}(\mathcal{C}) = \mathrm{conv}(\cap_\mathrm{i} \mathcal{C}_\mathrm{i}) \subset \cap_\mathrm{i} \mathrm{conv}(\mathcal{C}_\mathrm{i}) \tag{5.4}$$

Infine si definisca il parity polytope di dimensione q come

$$\mathbb{P}_q = conv(\{\mathbf{c} \in \mathbb{F}_2^q | \mathbf{1}_q^T \mathbf{c} = 0\}) \tag{5.5}$$

dove $\mathbf{1}_q$ è il vettore con tutti i valori ad 1 di dimensione q.

5.2.2 Formulazione ADMM

Sia $\mathbf{x} \triangleq \mathbf{c}$ e si introducono le matrici binarie $\mathbf{A}_i$ con $i = 1, .., n-k$, di dimensione $n \times |\mathcal{A}_i|$, che selezionano le $|\mathcal{A}_i|$ componenti di $\mathbf{x}$ che entrano nell'i-esimo check node.

Si utilizzano inoltre le variabili ausiliarie $\mathbf{z}_i \in \mathbb{P}_{|\mathcal{A}_i|}$ in modo tale da poter formulare il problema di PL come

$$\begin{aligned}
&\text{Minimize} && \boldsymbol{\gamma}^T \mathbf{x} && && (5.6a)\\
&\text{Subject to} && \mathbf{A}_i \mathbf{x} = \mathbf{z}_i, && i = (1, \dots, n-k) && (5.6b)\\
& && \mathbf{z}_i \in \mathbb{P}_{|\mathcal{A}_i|} && i = (1, \dots, n-k) && (5.6c)
\end{aligned}$$

In pratica la (5.6b) e la (5.6c) impongono che l'insieme dei vicini di ogni check node sia una delle possibili combinazioni ammissibili (quelle per le quali la somma modulo due è 0).

La formulazione ADMM del problema può essere scritta come

$$\mathbf{x}^* = \underset{\substack{\mathbf{x} \in \mathbb{R}^n \\ \mathbf{z} \in C_2 \\ \mathbf{Ax} = \mathbf{z}}}{\arg\min} \boldsymbol{\gamma}^T \mathbf{x} \tag{5.7}$$

$$\mathbf{z} = \begin{bmatrix} \mathbf{z_1} \\ . \\ . \\ . \\ \mathbf{z_{n-k}} \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} \mathbf{A_1} \\ . \\ . \\ . \\ \mathbf{A_{n-k}} \end{bmatrix}, \quad C_2 = \mathbb{P}_{|\mathcal{A}_1|} \times \cdots \times \mathbb{P}_{|\mathcal{A}_{n-k}|}$$

con $\mathbf{A}^T \mathbf{A}$ invertibile.

Lo augmented Lagrangian con cui lavora l'ADMM per questo problema risulta essere

$$L(\mathbf{x}, \mathbf{z}, \mathbf{p}) = \boldsymbol{\gamma}^T \mathbf{x} + \mathbf{p}^T(\mathbf{A}\mathbf{x} - \mathbf{z}) + \frac{c}{2}||\mathbf{A}\mathbf{x} - \mathbf{z}||^2 \tag{5.8}$$

Come è già stato detto, dopo aver fissato i valori iniziali $\mathbf{z}_0$ e $\mathbf{p}_0$, L'ADMM procede all'aggiornamento delle variabili secondo le seguenti regole:

$$\mathbf{x}_{t+1} = \underset{\mathbf{x}\in\mathbb{R}^n}{\arg\min}\ L(\mathbf{x}, \mathbf{z}_t, \mathbf{p}_t) \tag{5.9a}$$

$$\mathbf{z}_{t+1} = \underset{\mathbf{z}\in\mathcal{C}_2}{\arg\min}\ L(\mathbf{x}_{t+1}, \mathbf{z}, \mathbf{p}_t) \tag{5.9b}$$

$$\mathbf{p}_{t+1} = \mathbf{p}_t + c(\mathbf{A}\mathbf{x}_{t+1} - \mathbf{z}_{t+1}) \tag{5.9c}$$

Le equazioni (5.9) possono essere riscritte, eliminando i valori costanti rispetto alla variabile da minimizzare, come

$$\begin{aligned}
\mathbf{x}_{t+1} &= \underset{\mathbf{x}\in\mathbb{R}^n}{\arg\min}\ \boldsymbol{\gamma}^T\mathbf{x} + \mathbf{p}_t^T\mathbf{A}\mathbf{x} + \frac{c}{2}\mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} - c\mathbf{z}_t^T\mathbf{A}\mathbf{x} \\
&= \underset{\mathbf{x}\in\mathbb{R}^n}{\arg\min}\ \frac{2}{c}\boldsymbol{\gamma}^T\mathbf{x} + \mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} - 2\left(\mathbf{A}^T\mathbf{z}_t - \mathbf{A}^T\frac{\mathbf{p}_t}{c}\right)^T\mathbf{x} \\
\mathbf{z}_{t+1} &= \underset{\mathbf{z}\in\mathcal{C}_2}{\arg\min}\ -\mathbf{p}_t^T\mathbf{z} + \frac{c}{2}||\mathbf{z}||^2 - c\mathbf{z}^T\mathbf{A}\mathbf{x}_{t+1} \\
&= \underset{\mathbf{z}\in\mathcal{C}_2}{\arg\min}\ ||\mathbf{z}||^2 - 2\left(\mathbf{A}\mathbf{x}_{t+1} + \frac{\mathbf{p}_t}{c}\right)^T\mathbf{z} \\
\frac{\mathbf{p}_{t+1}}{c} &= \frac{\mathbf{p}_t}{c} + (\mathbf{A}\mathbf{x}_{t+1} - \mathbf{z}_{t+1})
\end{aligned}$$

Si definisca $\tilde{\mathbf{p}}_t = \frac{\mathbf{p}_t}{c}$ e sia $\mathcal{P}_{\mathcal{C}_2}$ l'operatore proiezione su $\mathcal{C}_2$ si ottiene allora

$$\begin{aligned}
\mathbf{x}_{t+1} &= \underset{\mathbf{x}\in\mathbb{R}^n}{\arg\min}\ \frac{2}{c}\boldsymbol{\gamma}^T\mathbf{x} + \mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} - 2\big(\mathbf{A}^T\mathbf{z}_t - \mathbf{A}^T\tilde{\mathbf{p}}_t\big)^T\mathbf{x} \\
\mathbf{z}_{t+1} &= \underset{\mathbf{z}\in\mathcal{C}_2}{\arg\min}\ ||\mathbf{z} - (\mathbf{A}\mathbf{x}_{t+1} + \tilde{\mathbf{p}}_t)||^2 - ||\mathbf{A}\mathbf{x}_{t+1} + \tilde{\mathbf{p}}_t||^2 \\
&= \underset{\mathbf{z}\in\mathcal{C}_2}{\arg\min}\ ||\mathbf{z} - (\mathbf{A}\mathbf{x}_{t+1} + \tilde{\mathbf{p}}_t)||^2 \\
&= \mathcal{P}_{\mathcal{C}_2}[\mathbf{A}\mathbf{x}_{t+1} + \tilde{\mathbf{p}}_t] \\
\tilde{\mathbf{p}}_{t+1} &= \tilde{\mathbf{p}}_t + \mathbf{A}\mathbf{x}_{t+1} - \mathbf{z}_{t+1}
\end{aligned}$$

L'equazione di aggiornamento della $\mathbf{x}$ può essere riscritta come

$$\begin{aligned}
\mathbf{x}_{t+1} &= \underset{\mathbf{x}\in\mathbb{R}^n}{\arg\min}\ \frac{2}{c}\boldsymbol{\gamma}^T\mathbf{x} + \mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} - 2\big(\mathbf{A}^T(\mathbf{z}_t^T - \tilde{\mathbf{p}}_t)\big)^T\mathbf{x} \\
&= \underset{\mathbf{x}\in\mathbb{R}^n}{\arg\min}\ \mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} - 2\big(\mathbf{A}^T(\mathbf{z}_t - \tilde{\mathbf{p}}_t) - \frac{\boldsymbol{\gamma}}{c}\big)^T\mathbf{x} \\
&= \underset{\mathbf{x}\in\mathbb{R}^n}{\arg\min}\ \mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} - 2\big(\mathbf{A}^T(\mathbf{z}_t - \tilde{\mathbf{p}}_t) - \frac{\boldsymbol{\gamma}}{c}\big)^T(\mathbf{A}^T\mathbf{A})^{-1}(\mathbf{A}^T\mathbf{A})\mathbf{x} \\
&= \underset{\mathbf{x}\in\mathbb{R}^n}{\arg\min}\ \mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} - 2\big((\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T(\mathbf{z}_t - \tilde{\mathbf{p}}_t) - (\mathbf{A}^T\mathbf{A})^{-1}\frac{\boldsymbol{\gamma}}{c}\big)^T(\mathbf{A}^T\mathbf{A})\mathbf{x}
\end{aligned}$$

Definendo $\mathbf{M} = \mathbf{A}\mathbf{A}^T$ e $\mathbf{v} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T(\mathbf{z}_t - \tilde{\mathbf{p}}_t) - (\mathbf{A}^T\mathbf{A})^{-1}\frac{\gamma}{c}$ si ha

$$\begin{aligned}
\mathbf{x}_{t+1} &= \arg\min_{\mathbf{x}\in\mathbb{R}^n} \ \mathbf{x}^T\mathbf{M}\mathbf{x} - 2\mathbf{v}^T\mathbf{M}\mathbf{x} \\
&= \arg\min_{\mathbf{x}\in\mathbb{R}^n} \ \mathbf{x}^T\mathbf{M}\mathbf{x} - 2\mathbf{v}^T\mathbf{M}\mathbf{x} + \mathbf{v}^T\mathbf{M}\mathbf{v} - \mathbf{v}^T\mathbf{M}\mathbf{v} \\
&= \arg\min_{\mathbf{x}\in\mathbb{R}^n} \ (\mathbf{x}-\mathbf{v})^T\mathbf{M}(\mathbf{x}-\mathbf{v}) - \mathbf{v}^T\mathbf{M}\mathbf{v}
\end{aligned}$$

Risulta evidente che il valore minimo si ottiene per

$$\mathbf{x} = \mathbf{v} = \mathbf{M}^{-1}\mathbf{A}^T(\mathbf{z}_t - \tilde{\mathbf{p}}_t) - \mathbf{M}^{-1}\frac{\gamma}{c}$$

Si può quindi concludere che l'aggiornamento delle variabili dell'ADMM avviene secondo le seguenti relazioni:

$$\mathbf{x}_{t+1} = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \ \mathbf{M}^{-1}\mathbf{A}^T(\mathbf{z}_t - \tilde{\mathbf{p}}_t) - \frac{1}{c}\mathbf{M}^{-1}\gamma \tag{5.10a}$$

$$\mathbf{z}_{t+1} = \arg\min_{\mathbf{z}\in\mathcal{C}_2} \ \mathcal{P}_{\mathcal{C}_2}[\mathbf{A}\mathbf{x}_{t+1} + \tilde{\mathbf{p}}_t] \tag{5.10b}$$

$$\tilde{\mathbf{p}}_{t+1} = \tilde{\mathbf{p}}_t + \mathbf{A}\mathbf{x}_{t+1} - \mathbf{z}_{t+1} \tag{5.10c}$$

5.2.3 Algoritmo

Data le equazioni (5.10) si scrive un algoritmo che impiega la struttura grafica dei codici LDPC, per aggiornare le variabili. Si aggiorna alternatamente, per un certo numero di volte, il messaggio uscente dai check node e dai variable node, come nel caso dell'algoritmo Sum Product.

Aggiornamento variable node

Aggiornamento variabili $\forall j \in \{1, ..., n\}$:

$$x_j = \frac{1}{|\mathcal{B}_j|}\left(\sum_{i\in\mathcal{B}_j} m_{i\to j} - \frac{1}{c}\gamma_j\right) \tag{5.11}$$

Messaggio uscente $\forall j \in \{1, ..., n\}$:

$$m_{j\to i} = x_j, \quad i \in \mathcal{B}_j \tag{5.12}$$

Aggiornamento check node

Aggiornamento variabili $\forall i \in \{1, ..., n-k\}$:

$$\mathbf{u}_i = [m_{j \to i}]_{j \in \mathcal{A}_i} + \tilde{\mathbf{p}}_i \tag{5.13a}$$

$$\mathbf{z}_i = \mathcal{P}_{\mathbb{P}_{|\mathcal{A}_i|}}(\mathbf{u}_i) \tag{5.13b}$$

$$\tilde{\mathbf{p}}_i = \mathbf{u}_i - \mathbf{z}_i \tag{5.13c}$$

Messaggio uscente $\forall i \in \{1, ..., n-k\}$:

$$m_{i \to j} = z_{i,j} - \tilde{p}_{i,j}, \quad j \in \mathcal{A}_i \tag{5.14}$$

La proiezione sul parity polytope presente in (5.13b) può essere effettuata usando l'algoritmo proposto in [2].

Dopo aver effettuato tutte le iterazioni dell'algoritmo, il segnale stimato $\hat{\mathbf{x}}$ si trova con la seguente regola:

$$\hat{x}_j = \begin{cases} 0, & \text{se } x_j < 0.5 \\ 1, & \text{se } x_j \geq 0.5 \end{cases} \tag{5.15}$$

5.2.4 Convergenza algoritmo

Nell'algoritmo appena descritto compare il parametro c dell'ADMM da cui dipende la convergenza. Si vuole capire quale valore di c scegliere per far si che l'algoritmo converga nel minor tempo possibile. Bisogna poi stimare quale è questo tempo in media, ovvero trovare quante volte bisogna iterare l'algoritmo.

Per fare questo è utile introdurre una metrica (Γ_t) che quantifica la variazione dei parametri del problema dall'iterazione $t+1$ alla t. Per l'algoritmo BLDR Γ_t si ottiene con la formula

$$\Gamma_t = ||\tilde{\mathbf{p}}_{\mathbf{t+1}} - \tilde{\mathbf{p}}_{\mathbf{t}}||^2 + ||\mathbf{x}_{\mathbf{t+1}} - \mathbf{x}_{\mathbf{t}}||^2 \tag{5.16}$$

Se l'algoritmo converge ci si aspetta di vedere il seguente andamento di Γ_t: Per valori di t piccoli dovrebbe decrescere in modo esponenziale per valori grandi decrescere linearmente.

Si chiami t^* l'iterazione a cui avviene il cambio di andamento, si vuole trovare un valore di c per il quale Γ_t diminuisca il più possibile, con il numero di iterazioni t^* minore possibile.

5.2.5 Algoritmo articolo

Viene riportato di seguito l'algoritmo presente in [2]. Si può notare che è molto simile a quello descritto sopra, anche se le variabili usate sono state

nominate in modo diverso, infatti l'Augmented Lagrangian in questo caso era

$$L(\mathbf{x}, \mathbf{z}, \lambda) = \boldsymbol{\gamma}^T \mathbf{x} + \boldsymbol{\lambda}(\mathbf{P}\mathbf{x} - \mathbf{z}) + \frac{\mu}{2}||\mathbf{P}\mathbf{x} - \mathbf{z}||^2 \tag{5.17}$$

Rispetto alla (5.8) la matrice $\mathbf{A}$ è stata chiamata $\mathbf{P}$ il parametro c diventa μ e $\mathbf{p}$ si trasforma in $\boldsymbol{\lambda}$.

In realtà nell'articolo sono stati usati nomi diversi anche per indicare i vicini dei check node e dei variable node. Tuttavia, per facilitare il confronto con l'algoritmo BLDR descritto in questo articolo, per questi verrà tenuta la notazione usata in precedenza.

L'algoritmo diventa:

- Inizializzare $\mathbf{z}_i$ e $\boldsymbol{\lambda}_i$ a 0 $\forall\, i \in \{1, ..., n-k\}$
- $\forall\, j \in \{1, ..., n\}$

$$x_j = \Pi_{[0,1]}\left(\frac{1}{|\mathcal{B}(j)|}\left(\sum_{i \in \mathcal{B}_j}(z_{i,j} - \frac{1}{\mu}\lambda i, j) - \frac{\gamma_j}{\mu}\right)\right) \tag{5.18}$$

- $\forall\, i \in \{1, ..., n-k\}$

$$\mathbf{v}_i = \mathbf{P}_i\mathbf{x} + \frac{\lambda_i}{\mu} \tag{5.19a}$$

$$\mathbf{z}_i = \Pi_{\mathbb{P}_{|\mathcal{A}_i|}}(\mathbf{v}_i) \tag{5.19b}$$

$$\lambda_i = \lambda_i + \mu(\mathbf{P}_i\mathbf{x} - \mathbf{z}_i) \tag{5.19c}$$

L'algoritmo è equivalente a quello visto in precedenza, anche se viene formulato in modo leggermente diverso. In questa versione compare l'operazione proiezione su $[0, 1]$ ($\Pi_{[0,1]}$), che mette le componenti maggiori di 1 ad 1 e quelle minori di 0 a 0. Facendo delle simulazioni si è trovato che questa operazione può essere omessa, infatti è sufficiente il vincolo $\mathbf{z}_i \in \mathbb{P}_{|\mathcal{A}_i|}$.

L'algoritmo formulato in questa tesi, a differenza di quello dell'articolo, è più simile al Sum Product, in cui c'è uno scambio di messaggi tra i variable node e i check node.

Nelle simulazioni fatte nel Capitolo 6, per il BLDR proposto nell'articolo, si è iterato l'algoritmo per un certo numero di volte che è stato stabilito facendo l'analisi su Γ_t. Nell'algoritmo dell'articolo, invece, non viene scelto un numero di iterazioni, ma l'algoritmo va avanti fino a che non si verifica la condizione

$$\max_{i \in \{1,..,n-k\}} ||\mathbf{P}_i\mathbf{x} - \mathbf{z}_i|| < \varepsilon \tag{5.20}$$

dove ε è una costante maggiore di 0 da cui dipende il numero di iterazioni che farà l'algoritmo. Minore è ε più iterazioni verranno fatte dall'algoritmo, tuttavia, se si sceglie ε troppo grande, si rischia di non arrivare alla convergenza in tempi accettabili.

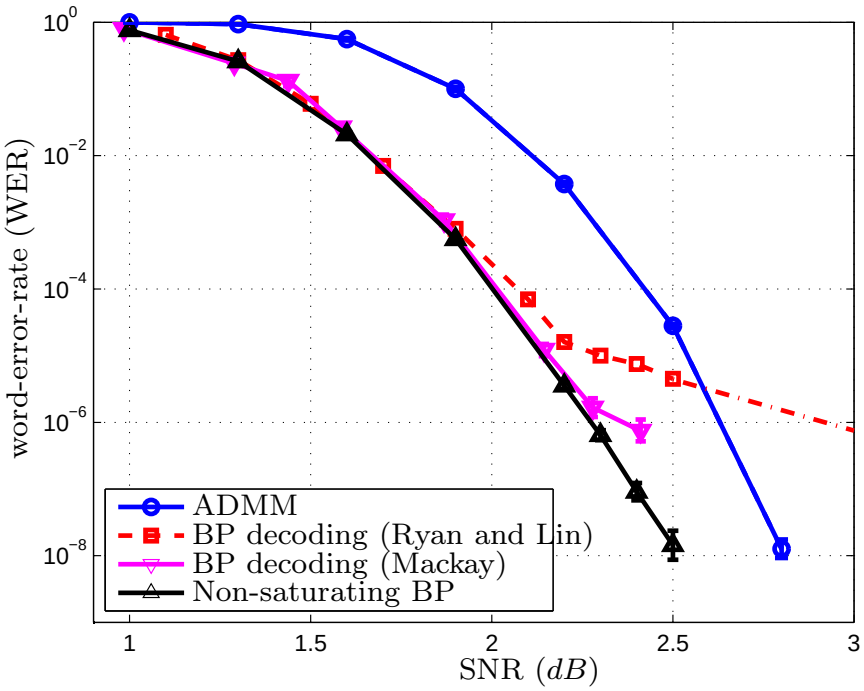


Figura 5.1: WER per codici di Margulis [2540,1320]: *Barman, Liu, Draper, Recht. "Decomposition Methods for Large Scale LP Decoding." 2012*

5.2.6 Risultati articolo

In [2] l'algoritmo BLDR presente nell'articolo viene simulato per i codici di Margulis, descritti nel Capitolo 2. Viene simulata la word error rate (WER), di vari codici, in funzione del rapporto segnale rumore (SNR).

Come si vede nella Figura 5.1, vengono confrontati l'algoritmo di tipo ADMM, simile a quello descritto in questo capitolo, con vari algoritmi di tipo Sum-Product, che nell'articolo vengono chiamati BP (Belief-Propagation). Due degli algoritmi BP proposti, saturano, mentre il terzo non lo fa. Si è infatti usato un algoritmo BP implementato in modo tale da evitare la saturazione.

Da questa simulazione si può avere una prima idea sulle prestazioni dell'algoritmo di tipo ADMM proposto. Per questo codice, infatti, si può vedere che la WER è molto vicina alle curve del BP e soprattutto non satura.

5.3 Algoritmo ZGCE

Si analizza ora un altro algoritmo di tipo ADMM chiamato ZGCE (Zhu Giannakis Cano Erseghe) dal nome di alcuni degli autori dei due articoli da cui deriva. Il primo articolo [16] considera la decodifica come un problema

distribuito, paragonandolo alla comunicazione su una rete di sensori, in [14], invece, l'approccio distribuito viene analizzato in un ambito più generale.

5.3.1 Formulazione ADMM

La formulazione del problema è diversa da quella vista per il BLDR, in questo caso, infatti, si duplica il messaggio su tutti i lati collegati ai variable node ed ai check node. Si ottiene così un vettore per ogni variable node $\mathbf{x}_j$ e uno per ogni check node $\breve{\mathbf{x}}_i$ che contengono il messaggio verso tutti i vicini. Si introducono inoltre delle variabili ausiliarie, $\mathbf{z}_j = \mathbf{x}_j$ e $\breve{\mathbf{z}}_i = \breve{\mathbf{x}}_i$, per imporre che le variabili appartenenti allo stesso lato abbiano lo stesso valore $z_{i,j} = \breve{z}_{i,j}$.

Utilizzando una notazione simile a quella del BLDR, il problema PL da minimizzare diventa

$$\begin{aligned}
&\text{Minimize} && \sum_{j=1}^{n} \gamma_j x_{j,1} && && (5.21a)\\
&\text{Subject to} && \breve{x}_{i,j} = \breve{z}_{i,j}, && \forall i,\, j \in \mathcal{A}_i && (5.21b)\\
& && x_{j,i} = z_{j,i}, && \forall j,\, i \in \mathcal{B}_j && (5.21c)\\
& && \breve{z}_{i,j} = z_{j,i}, && \forall i,\, j \in \mathcal{A}_i && (5.21d)\\
& && \breve{\mathbf{x}}_i \in \mathbb{P}_{|\mathcal{A}_i|}, && \forall i && (5.21e)\\
& && \mathbf{x}_j \in \mathbb{U}_{|\mathcal{B}_j|}, && \forall j && (5.21f)
\end{aligned}$$

dove $\mathbb{U}_{|\mathcal{B}_j|}$ è l'insieme contenente i vettori $\mathbf{1}_{|\mathcal{B}_j|}$ e $\mathbf{0}_{|\mathcal{B}_j|}$. La formulazione ADMM in questo caso è

$$\mathbf{x}^* = \underset{\mathbf{x}\in C_1, \mathbf{z}\in C_2, \mathbf{x}=\mathbf{z}}{\arg\min} \; \boldsymbol{\Theta}^T \mathbf{x} \tag{5.22}$$

dove

$$\mathbf{x} = \begin{bmatrix} \mathbf{x_1} \\ . \\ . \\ . \\ \mathbf{x_n} \\ \breve{\mathbf{x}}_\mathbf{1} \\ . \\ . \\ . \\ \breve{\mathbf{x}}_\mathbf{n-k} \end{bmatrix}, \quad \mathbf{z} = \begin{bmatrix} \mathbf{z_1} \\ . \\ . \\ . \\ \mathbf{z_n} \\ \breve{\mathbf{z}}_\mathbf{1} \\ . \\ . \\ . \\ \breve{\mathbf{z}}_\mathbf{n-k} \end{bmatrix}, \quad \boldsymbol{\Theta} = \begin{bmatrix} \boldsymbol{\theta}_1 \\ . \\ . \\ . \\ \boldsymbol{\theta}_n \\ \breve{\boldsymbol{\theta}}_1 \\ . \\ . \\ . \\ \breve{\boldsymbol{\theta}}_{n-k} \end{bmatrix}$$

$$C_1 = \mathbb{U}_{|\mathcal{B}_1|} \times \cdots \times \mathbb{U}_{|\mathcal{B}_n|} \times \mathbb{P}_{|\mathcal{A}_1|} \times \cdots \times \mathbb{P}_{|\mathcal{A}_{n-k}|}$$

e con

$$\boldsymbol{\theta}_j = [\gamma_j, 0, ..., 0], \quad j = [1, ..., n] \qquad \breve{\boldsymbol{\theta}}_i = \mathbf{0}_{|\mathcal{A}_i|}, \quad i = [1, ..., n-k]$$

Per quanto riguarda C_2, invece, risulta più comodo esprimerlo con la matrice di proiezione

$$\mathbf{L} = \Pi^T \left(\mathbf{I} \otimes \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} \right) \Pi$$

dove $\otimes$ è il prodotto di Kronecker tra matrici.

L' Augmented Lagrangian con cui lavora l'ADMM per questo problema risulta quindi essere

$$L(\mathbf{x}, \mathbf{z}, \mathbf{p}) = \boldsymbol{\Theta}^T \mathbf{x} + \mathbf{p}^T (\mathbf{x} - \mathbf{z}) + \frac{c}{2} ||\mathbf{x} - \mathbf{z}||^2 \tag{5.23}$$

L'aggiornamento delle variabili avviene sempre secondo la regola

$$\mathbf{x}_{t+1} = \underset{\mathbf{x} \in C_1}{\arg\min} \; L(\mathbf{x}, \mathbf{z}_t, \mathbf{p}_t) \tag{5.24a}$$

$$\mathbf{z}_{t+1} = \underset{\mathbf{z} \in C_2}{\arg\min} \; L(\mathbf{x}_{t+1}, \mathbf{z}, \mathbf{p}_t) \tag{5.24b}$$

$$\mathbf{p}_{t+1} = \mathbf{p}_t + c(\mathbf{x}_{t+1} - \mathbf{z}_{t+1}) \tag{5.24c}$$

Si svolgono ora dei passaggi, analoghi a quelli fatti per l'algoritmo BLDR, per trovare le regole di aggiornamento delle variabili. Per quanto riguarda $\mathbf{z}$ e $\tilde{\mathbf{p}} \triangleq \frac{\mathbf{p}}{c}$ si ha

$$\begin{aligned}
\mathbf{z}_{t+1} &= \underset{\mathbf{z} \in C_2}{\arg\min} \; -2\tilde{\mathbf{p}}_t^T \mathbf{z} + ||\mathbf{z}||^2 - 2\mathbf{x}_{t+1}^T \mathbf{z} \\
&= \underset{\mathbf{z} \in C_2}{\arg\min} \; ||\mathbf{z}||^2 - 2(\mathbf{x}_{t+1} + \tilde{\mathbf{p}}_t)^T \mathbf{z} \\
&= \underset{\mathbf{z} \in C_2}{\arg\min} \; ||\mathbf{z} - (\mathbf{x}_{t+1} + \tilde{\mathbf{p}}_t)||^2 \\
\tilde{\mathbf{p}}_{t+1} &= \tilde{\mathbf{p}}_t + \mathbf{x}_{t+1} - \mathbf{z}_{t+1}
\end{aligned}$$

Utilizzando la matrice $\mathbf{L}$ di proiezione su C_2 si ottiene

$$\begin{aligned}
\mathbf{z}_{t+1} &= \mathbf{L}(\mathbf{x}_{t+1} + \tilde{\mathbf{p}}_t) \\
\tilde{\mathbf{p}}_{t+1} &= (\mathbf{I} - \mathbf{L})(\tilde{\mathbf{p}}_t + \mathbf{x}_{t+1})
\end{aligned} \tag{5.25}$$

Imponendo le condizioni iniziali $\tilde{\mathbf{p}}_0 = \mathbf{0}$ e $\mathbf{z}_0 = \mathbf{L}\mathbf{x}_0$ si ottiene che

$$\begin{aligned}
&\mathbf{p}_t \perp C_2, && \forall t \\
&\mathbf{z}_t \in C_2, && \forall t
\end{aligned}$$

Di conseguenza la (5.25) si può riformulare come

$$\begin{aligned}
\mathbf{z}_{t+1} &= \mathbf{L}\mathbf{x}_{t+1} \\
\tilde{\mathbf{p}}_{t+1} &= \tilde{\mathbf{p}}_t + (\mathbf{I} - \mathbf{L})\mathbf{x}_{t+1}
\end{aligned} \tag{5.27}$$

Si analizza ora l'aggiornamento della variabile $\mathbf{x}$.

$$\begin{aligned}
\mathbf{x}_{t+1} &= \arg\min_{\mathbf{x}\in C_1} \frac{2}{c}\boldsymbol{\Theta}^T\mathbf{x} + 2\tilde{\mathbf{p}}_t^T\mathbf{x} + \mathbf{x}^T\mathbf{x} - 2\mathbf{z}_t^T\mathbf{x} \\
&= \arg\min_{\mathbf{x}\in C_1} \mathbf{x}^T\mathbf{x} - 2\left(\mathbf{z}_t - \tilde{\mathbf{p}}_t - \frac{\boldsymbol{\Theta}}{c}\right)^T\mathbf{x} \\
&= \arg\min_{\mathbf{x}\in C_1} ||\mathbf{x}||^2 - 2\left(\mathbf{z}_t - \tilde{\mathbf{p}}_t - \frac{\boldsymbol{\Theta}}{c}\right)^T\mathbf{x} \\
&= \mathbb{P}_{C_1}\left(\mathbf{z}_t - \tilde{\mathbf{p}}_t - \frac{\boldsymbol{\Theta}}{c}\right)
\end{aligned}$$

Quindi ricordando che nella (5.27) si era trovato $\mathbf{z}_t = \mathbf{L}\mathbf{x}_t$ e introducendo la variabile $\mathbf{m}_t \triangleq -\tilde{\mathbf{p}}_t$ si ottengono le equazioni di aggiornamento del problema ZGCE

$$\mathbf{x}_{t+1} = \mathbb{P}_{C_1}\left(\mathbf{L}\mathbf{x}_t + \mathbf{m}_t - \frac{\boldsymbol{\Theta}}{c}\right) \tag{5.28a}$$

$$\mathbf{m}_{t+1} = \mathbf{m}_t - \mathbf{x}_{t+1} + \mathbf{L}\mathbf{x}_{t+1} \tag{5.28b}$$

dove $\mathbb{P}_{C_1}$ è l'operatore proiezione su C_1.
A questo punto si vuole trovare un algoritmo che, come nel caso del BLDR, aggiorni alternatamente i messaggi uscenti dai variable node e dai check node fino ad arrivare alla convergenza.

5.3.2 Aggiornamento variable node

Partendo dalle equazioni (5.28) si vuole ottenere una formulazione per aggiornare il messaggio uscente dal variable node j. Dalla (5.28a), usando la notazione $\mathbf{y}_t \triangleq \mathbf{L}\mathbf{x}_t$, consegue che

$$\begin{aligned}
\mathbf{x}_{j,t+1} &= \mathbb{P}_{C_1}\left(\mathbf{y}_{j,t} + \mathbf{m}_{j,t} - \frac{\boldsymbol{\theta}_j}{c}\right) \\
&= \frac{\mathbf{1}_{|\mathcal{B}_j|}\mathbf{1}_{|\mathcal{B}_j|}^T}{|\mathcal{B}_j|}\left(\mathbf{y}_{j,t} + \mathbf{m}_{j,t} - \frac{\boldsymbol{\theta}_j}{c}\right)
\end{aligned}$$

Visto che il segnale viene duplicato su tutte le variabili in uscita dal variable node, per semplicità $\mathbf{x}_{j,t+1}$ si può indicare come

$$\mathbf{x}_{j,t+1} = \mathbf{1}_{|\mathcal{B}_j|}x_{j,t+1} \qquad x_{j,t+1} = \frac{\mathbf{1}_{|\mathcal{B}_j|}^T}{|\mathcal{B}_j|}\left(\mathbf{y}_{j,t} + \mathbf{m}_{j,t} - \frac{\boldsymbol{\theta}_j}{c}\right)$$

che diventa

$$x_{j,t+1} = \frac{1}{|\mathcal{B}_j|}\left(\sum_{i\in\mathcal{B}_j}\frac{x_{j,i,t} + \breve{x}_{i,j,t}}{2} + \sum_{i\in\mathcal{B}_j} m_{j,i,t} - \frac{\gamma_j}{c}\right) \tag{5.30}$$

Dalla (5.28b) deriva, invece,

$$\mathbf{m}_{j,t+1} = \mathbf{m}_{j,t} + \mathbf{y}_{j,t+1} - \mathbf{x}_{j,t+1} \tag{5.31}$$

Definendo $m_{j,t} \triangleq \sum_{i \in \mathcal{B}_j} m_{j,i,t}$ da (5.30) e (5.31) si possono riscrivere le equazioni di aggiornamento ai variable nodes come

$$\begin{aligned} x_{j,t+1} &= \frac{1}{|\mathcal{B}_j|} \sum_{i \in \mathcal{B}_j} \frac{x_{j,i,t} + \breve{x}_{i,j,t}}{2} + m_{j,t} - \frac{\gamma_j}{|\mathcal{B}_j| c} \\ m_{j,t+1} &= m_{j,t} + \frac{1}{|\mathcal{B}_j|} \sum_{i \in \mathcal{B}_j} \frac{x_{j,i,t+1} + \breve{x}_{i,j,t+1}}{2} - x_{j,t+1} \end{aligned}$$

Visto che $x_{j,i} = x_j \, \forall i \in \mathcal{B}_j$ si ottiene

$$x_{j,t+1} = \frac{1}{2}\left(x_{j,t} + \frac{1}{|\mathcal{B}_j|} \sum_{i \in \mathcal{B}_j} \breve{x}_{i,j,t} \right) - \frac{\gamma_j}{|\mathcal{B}_j| c} + m_{j,t} \tag{5.33a}$$

$$m_{j,t+1} = m_{j,t} + \frac{1}{2}\left(\frac{1}{|\mathcal{B}_j|} \sum_{i \in \mathcal{B}_j} \breve{x}_{i,j,t+1} - x_{j,t+1} \right) \tag{5.33b}$$

5.3.3 Aggiornamento check node

Per l'i-esimo check node dalla (5.28a), con la notazione $\breve{y}_{i,j,t} \triangleq \frac{x_{j,i,t} + \breve{x}_{i,j,t}}{2}$, si ha

$$\begin{aligned} \breve{\mathbf{x}}_{i,t+1} &= \mathcal{P}_{\mathbb{P}_{|\mathcal{A}_i|}}(\breve{\mathbf{y}}_{i,t} + \breve{\mathbf{m}}_{i,t}) \\ &= \mathcal{P}_{\mathbb{P}_{|\mathcal{A}_i|}} \left(\frac{1}{2}[x_{j,t}]_{j \in \mathcal{A}_i} + \frac{1}{2}\breve{\mathbf{x}}_{i,t} + \breve{\mathbf{m}}_{i,t} \right) \end{aligned}$$

L'equazione (5.28b), invece, implica

$$\begin{aligned} \breve{\mathbf{m}}_{i,t+1} &= \breve{\mathbf{m}}_{i,t} - \breve{\mathbf{x}}_{i,t+1} + \breve{\mathbf{y}}_{i,t+1} \\ &= \breve{\mathbf{m}}_{i,t} + \frac{1}{2}[x_{j,t+1}]_{j \in \mathcal{A}_i} - \frac{1}{2}\breve{\mathbf{x}}_{i,t+1} \end{aligned}$$

Ricapitolando l'aggiornamento dei messaggi ai check node avviene tramite le seguenti equazioni:

$$\breve{\mathbf{x}}_{i,t+1} = \mathcal{P}_{\mathbb{P}_{|\mathcal{A}_i|}} \left(\frac{1}{2}[x_{j,t}]_{j \in \mathcal{A}_i} + \frac{1}{2}\breve{\mathbf{x}}_{i,t} + \breve{\mathbf{m}}_{i,t} \right) \tag{5.36a}$$

$$\breve{\mathbf{m}}_{i,t+1} = \breve{\mathbf{m}}_{i,t} + \frac{1}{2}[x_{j,t+1}]_{j \in \mathcal{A}_i} - \frac{1}{2}\breve{\mathbf{x}}_{i,t+1} \tag{5.36b}$$

5.3.4 Algoritmo

Come nel caso del BLDR, l'algoritmo prevede di aggiornare tutte le variabili per un certo numero di volte, fino ad arrivare alla convergenza. In questo caso le variabili sono x_j e m_j, per i variable node ($j = 1, ..n$), e $\breve{\mathbf{x}}_i$, $\breve{\mathbf{m}}_i$, per quanto riguarda i check node ($i = 1, ..., n-k$). Le equazioni di aggiornamento per check e variable node sono state scritte rispettivamente in (5.33) e (5.36).

In questo modo però, ad ogni istante di tempo sarebbe necessario aggiornare una variabile dei variable node, una dei check node e solo successivamente l'altra dei variable node e dei check node. Si vogliono invece ricavare delle equazioni che permettano di aggiornare prima tutte le equazioni dei variable node e poi quelle dei check node come nel caso del BLDR.

Per fare questo si introducono le variabili $\mathbf{n}_t \triangleq \mathbf{m}_{t-1}$ e a $\breve{\mathbf{n}}_t \triangleq \breve{\mathbf{m}}_{t-1}$. Le equazioni di aggiornamento diventano per i variable node

$$n_{j,t+1} = n_{j,t} + \frac{1}{2}\left(\frac{1}{|\mathcal{B}_j|}\sum_{i\in\mathcal{B}_j}\breve{x}_{i,j,t} - x_{j,t}\right) \tag{5.37a}$$

$$x_{j,t+1} = \frac{1}{2}\left(x_{j,t} + \frac{1}{|\mathcal{B}_j|}\sum_{i\in\mathcal{B}_j}\breve{x}_{i,j,t}\right) - \frac{\gamma_j}{|\mathcal{B}_j|c} + n_{j,t+1} \tag{5.37b}$$

e per i check node

$$\breve{\mathbf{n}}_{i,t+1} = \breve{\mathbf{n}}_{i,t} + \frac{1}{2}[x_{j,t}]_{j\in\mathcal{A}_i} - \frac{1}{2}\breve{\mathbf{x}}_{i,t} \tag{5.38a}$$

$$\breve{\mathbf{x}}_{i,t+1} = \mathcal{P}_{\mathbb{P}_{|\mathcal{A}_i|}}\left(\frac{1}{2}[x_{j,t}]_{j\in\mathcal{A}_i} + \frac{1}{2}\breve{\mathbf{x}}_{i,t} + \breve{\mathbf{n}}_{i,t+1}\right) \tag{5.38b}$$

L'algoritmo ZGCE consiste quindi nel aggiornare, fino ad arrivare alla convergenza, i messaggi uscenti dai check e dai variable node secondo le seguenti regole:

Aggiornamento variable node:

Aggiornamento variabili $\forall j \in \{1, ..., n\}$:

$$n_j = n_j + \frac{1}{2}\left(\frac{1}{|\mathcal{B}_j|}\sum_{i\in\mathcal{B}_j} m_{i\to j} - x_j\right) \tag{5.39a}$$

$$x_j = \frac{1}{2}\left(x_j + \frac{1}{|\mathcal{B}_j|}\sum_{i\in\mathcal{B}_j} m_{i\to j}\right) - \frac{\gamma_j}{|\mathcal{B}_j|c} + n_j \tag{5.39b}$$

Messaggio uscente $\forall j \in \{1, ..., n\}$:

$$m_{j\to i} = x_j, \quad i \in \mathcal{B}_j \tag{5.40}$$

Aggiornamento check node:

Aggiornamento variabili $\forall i \in \{1, ..., n-k\}$:

$$\breve{\mathbf{n}}_i = \breve{\mathbf{n}}_i + \frac{1}{2}\left([m_{j\to i}]_{j\in\mathcal{A}_i} - \breve{\mathbf{x}}_i\right) \tag{5.41a}$$

$$\breve{\mathbf{x}}_i = \mathcal{P}_{\mathbb{P}_{|\mathcal{A}_i|}}\left(\frac{1}{2}[m_{j\to i}]_{j\in\mathcal{A}_i} + \frac{1}{2}\breve{\mathbf{x}}_i + \breve{\mathbf{n}}_i\right) \tag{5.41b}$$

Messaggio uscente $\forall i \in \{1, ..., n-k\}$:

$$m_{i\to j} = \breve{x}_{i,j}, \quad j \in \mathcal{A}_i \tag{5.42}$$

La proiezione sul parity poytope, presente in (5.41b), anche in questo caso può essere effettuata usando l'algoritmo proposto in [2].

Come per l'algoritmo BLDR, dopo aver fatto tutte le iterazioni, il segnale stimato $\hat{\mathbf{x}}$ si trova con la regola

$$\hat{x}_j = \begin{cases} 0, & \text{se } x_j < 0.5 \\ 1, & \text{se } x_j \geq 0.5 \end{cases} \tag{5.43}$$

5.3.5 Convergenza algoritmo

Anche nel caso dell'Algoritmo ZGCE, si vuole definire una metrica Γ_t, per capire il valore del parametro c e del numero di iterazioni necessarie per la convergenza.
Il concetto è lo stesso del caso BLDR, ovvero vedere di quanto cambiano le variabili da una iterazione all'altra, quindi in questo caso Γ_t vale

$$\begin{aligned}\Gamma_t =& ||\mathbf{n_{t+1}} - \mathbf{n_t}||^2 + ||\mathbf{x_{t+1}} - \mathbf{x_t}||^2 \\ &+ ||\breve{\mathbf{n}}_{\mathbf{t+1}} - \breve{\mathbf{n}}_{\mathbf{t}}||^2 + ||\breve{\mathbf{x}}_{\mathbf{t+1}} - \breve{\mathbf{x}}_{\mathbf{t}}||^2\end{aligned} \tag{5.44}$$

Capitolo 6

Simulazioni

Si analizzano ora le simulazioni fatte per gli algoritmi spiegati nei Capitoli 3, 4, 5, per i tre codici LDPC analizzati nel Capitolo 2.

6.1 Codici IEEE 802.11n (648-324)

Per questo codice si confronteranno le prestazioni dei seguenti algoritmi:

- Sum Product, sia quello già presente in Matlab (SP Matlab), che quello implementato con il metodo esposto nel Capitolo 3 (SP)
- Vontobel-Koetter (VK) così come descritto nel Capitolo 4
- ADMM, sia BLDR che ZGCE, descritti nel Capitolo 5.

Per prima cosa si analizzano i parametri da utilizzare nella simulazione dei due ADMM, tramite la metrica Γ_t descritta nel Capitolo 5. Vediamo quindi i risultati ottenuti per i due algoritmi.

6.1.1 Convergenza BLDR

Al fine di trovare il valore del parametro c e del numero di iterazioni t^*, si calcola la metrica Γ_t come descritto in (5.16). Per renderla meno dipendente dalla singola simulazione si è preferito mediare Γ_t su più realizzazioni, in ogni caso non sono state rilevate variazioni evidenti tra le varie realizzazioni.

Nella Figura 6.1 viene riportato l'andamento di Γ_t, in funzione del numero di iterazioni t, al variare di c per SNR $= 0\,\mathrm{dB}$ e SNR $= 4\,\mathrm{dB}$. Si vede che, nel caso SNR $= 4\,\mathrm{dB}$, le differenze tra le varie curve sono molto più evidenti e quindi è più facile fare delle considerazioni sulla scelta di c.

Si può escludere subito il valore $c = 1000$ in quanto l'algoritmo non converge entro le 1000 iterazioni, anche i casi $c = 0.1$ e $c = 100$ sono problematici, in quanto la curva scende dopo tante iterazioni. Tra $c = 1$ e $c = 10$ è da preferire il secondo valore, in quanto l'algoritmo converge prima.

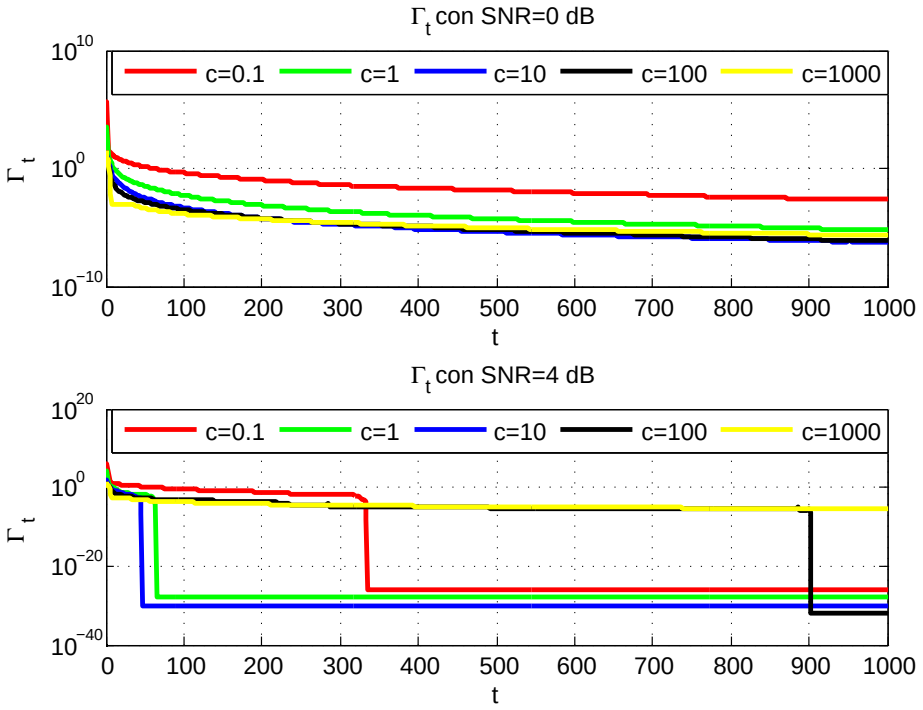


Figura 6.1: Γ_t per algoritmo BLDR e codice LDPC IEEE 802.11n (648,324)

Per quanto riguarda il numero di iterazioni t^* necessario per la convergenza dell'algoritmo, guardando le curva con $c = 10$, per entrambi gli SNR, si conclude che un buon valore risulta essere $t^* = 100$. Nel caso SNR $= 0\,\text{dB}$, infatti, il Γ_t della curva scende gradualmente e sembra assestarsi intorno a $t = 100$. Nel secondo caso (SNR $= 4\,\text{dB}$), invece, la curva scende con un gradino, intorno a $t = 50$, e poi rimane costante. Tuttavia, visto che il punto in cui scende la curva cambia abbastanza tra le varie simulazioni, per avere un margine di sicurezza si sceglie $t^* = 100$ anche in questo caso.

L'analisi della convergenza è stata fatta anche per altri valori di SNR e altri c, ma non vengono riportati in quanto i risultati ottenuti hanno confermato l'ottimalità del parametro $c = 10$ e del numero di iterazioni $t^* = 100$ per tutti gli SNR.

Si vuole sottolineare che più è grande il numero di iterazioni t^*, migliori sono le prestazioni in termini di BER, ma, aumentando di tanto t^* rispetto al valore scelto 100, si avrebbe un guadagno limitato a costo di un aumento notevole dei tempi di decodifica.

6.1.2 Convergenza ZGCE

Anche per l'algoritmo ZGCE risulta necessario trovare i parametri c e t^* e, per fare questo, ci si affida alla metrica Γ_t come definita in (5.44).

In Figura 6.2 viene riportato l'andamento di Γ_t al variare del numero di iterazioni t per l'algoritmo ZGCE.

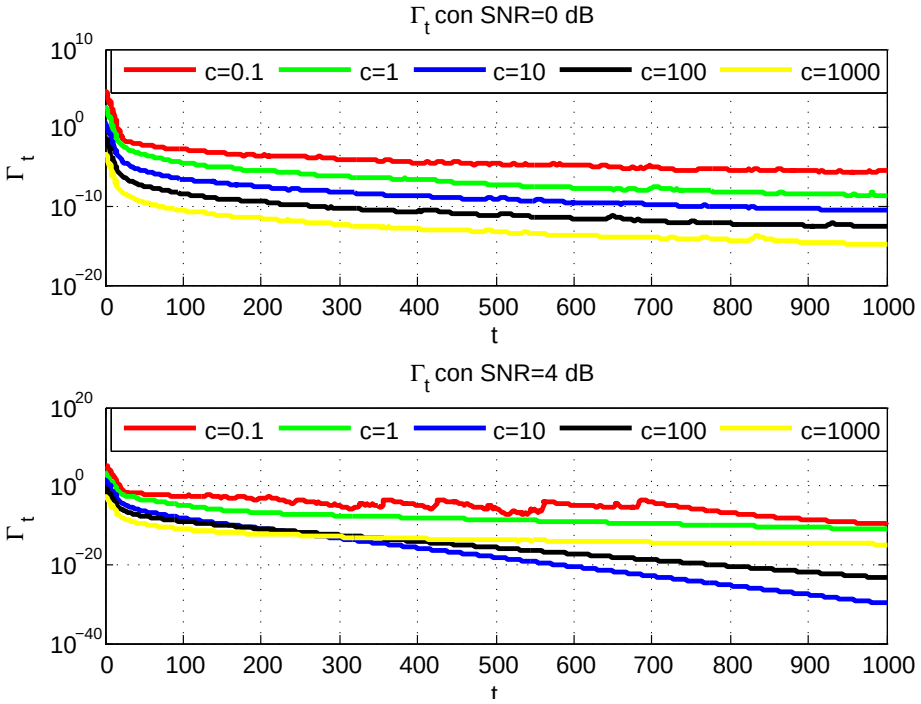


Figura 6.2: Γ_t per algoritmo ZGCE e codice LDPC IEEE 802.11n (648,324)

Potrebbe sembrare, osservando il caso SNR = 0dB, che più si aumenti il valore di c più sia rapida la convergenza. Facendo altre verifiche si è visto che questo non risulta essere vero, infatti, usando $c = 100 - 1000$, serve un numero molto elevato di iterazioni per far si che l'algoritmo converga.

Questo fatto è più evidente nel caso SNR = 4dB, dove si nota che le curve con $c = 100$ e 1000, all'inizio scendono più velocemente delle altre, ma poi non si assestano intorno ad un certo valore di Γ e continuano a scendere in modo rapido. Questo sta a indicare che, entro le 1000 iterazioni della simulazione fatta, non arrivano a convergenza.

Viste le osservazioni appena fatte, anche in questo caso si sceglie $c = 10$. Per quanto riguarda il numero di iterazioni necessarie t^* si osserva che, in entrambi i grafici della Figura 6.2, la curva con $c = 10$ si assesta dopo circa 100 iterazioni. Di conseguenza per tutti gli SNR i parametri utilizzati sono $c = 10$ e $t^* = 100$.

6.1.3 Simulazioni BER

Si analizzano ora le prestazioni, in termini di probabilità di errore sul bit (BER), per tutti gli algoritmi di decodifica descritti, per i quali sono stati

adottati i seguenti parametri per le simulazioni:

- SP e SP Matlab → Numero iterazioni: 20
- Vontobel-Koetter → Numero iterazioni: 128
- ADMM BLDR e ZGCE → c: 10, Numero iterazioni (t^*): 100

La scelta dei parametri per i due algoritmi ADMM è stata motivata, per quanto riguarda gli altri due algoritmi, si è scelto un numero di iterazioni rappresentate un buon compromesso tra la BER e la velocità di esecuzione, facendo anche riferimento alla letteratura esistente.

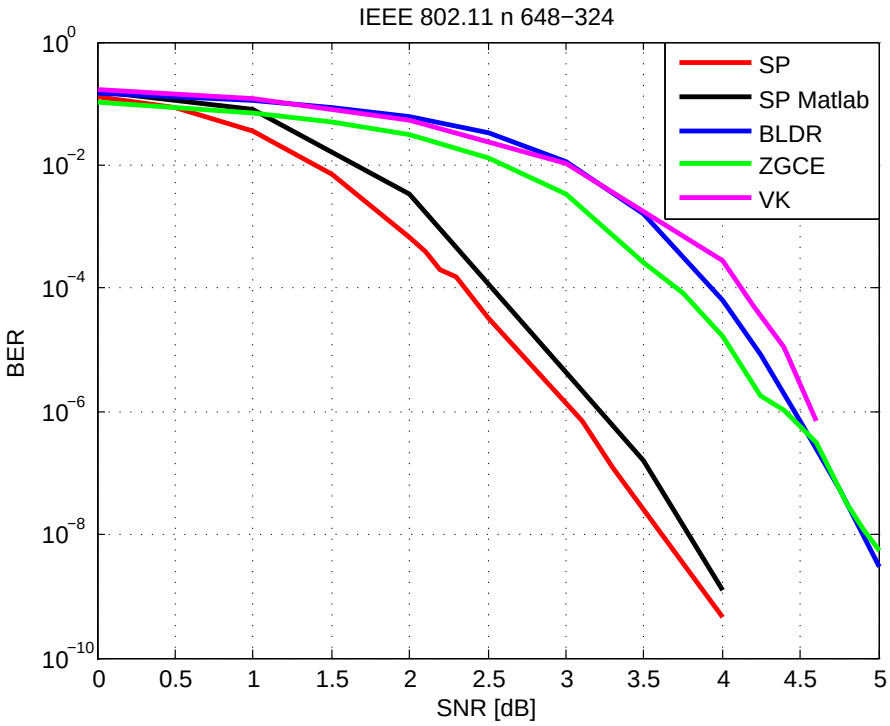


Figura 6.3: BER per il codice LDPC IEEE 802.11n (648,324)

In Figura 6.3 viene riportata la simulazione effettuata. Di seguito vengono elencate alcune osservazioni:

- Per quanto riguarda l'algoritmo Sum-Product, si può notare che la versione di Matlab ha una BER leggermente piú elevata, questo probabilmente a causa delle approssimazioni utilizzate per rendere il codice più veloce.
- I tre algoritmi che usano tecniche di PL hanno più o meno le stesse prestazioni in termini di BER, tuttavia si vedrà che l'algoritmo VK risulta molto più lento rispetto ai due ADMM.

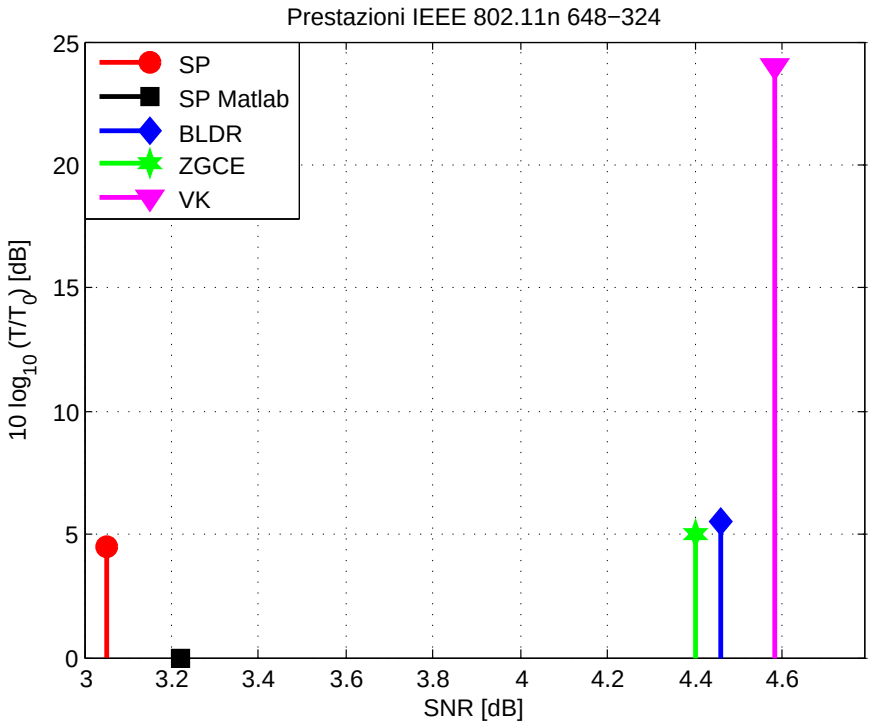


Figura 6.4: Confronto prestazioni per il codice LDPC IEEE 802.11n (648,324)

- Tra i due algoritmi ADMM, lo ZGCE sembra funzionare leggermente meglio per SNR bassi, mentre per valori più elevati le prestazioni sostanzialmente si equivalgono

- Si può notare come tra gli algoritmi che usano tecniche di programmazione lineare e quelli SP ci sia circa 1.5 dB di differenza

6.1.4 Confronto prestazioni

Si vogliono analizzare ora le prestazioni dei vari algoritmi, non solo in termini di probabilità di errore sul bit, ma considerando anche i tempi di esecuzione. Per fare questo si prenderà come tempo di riferimento l'algoritmo SP implementato in Matlab, essendo il più veloce, e si confronteranno i tempi di decodifica degli altri rispetto a questo.

Nella Figura 6.4 vengono riportati i tempi di decodifica per tutti gli algoritmi, normalizzati rispetto a quello del SP Matlab. In pratica la grandezza sull'asse delle ordinate è

$$10 \log_{10} \frac{T}{T_0}$$

dove T e T_0 sono rispettivamente il tempo di decodifica del generico algoritmo e del SP Matlab. L'SNR a cui sono stati calcolati i tempi è quello a cui i vari algoritmi raggiungono la BER $= 10^{-6}$.

L'algoritmo ideale dovrebbe avere tempo di decodifica limitato e raggiungere la BER di 10^{-6} per il valore di SNR minore possibile, di conseguenza dovrebbe trovarsi in basso a sinistra nel grafico riportato.

Anche in questo caso si nota che tra gli algoritmi che sfruttano tecniche PL e il Sum-Product esiste circa 1.5 dB di differenza e i due ADMM e il VK hanno BER simili tra loro. Tralasciando l'algoritmo SP Matlab, che è stato preso solo come riferimento, si vede che i tempi dei due ADMM sono in linea con quello del Sum-Product. Il Vontobel-Koetter, invece, oltre ad avere una BER peggiore risulta anche molto più lento.

Per quanto riguarda il confronto tra i due ADMM, il ZGCE risulta leggermente più performante rispetto al BLDR, sia in termini di BER che di tempi di decodifica.

6.2 Codici IEEE 802.11n (1296-648)

Si analizzano ora i risultati ottenuti per un altro codice LDPC, definito nello Standard IEEE 802.11n e introdotto nel Capitolo 2.

Questo codice è molto simile al precedente, ma avrà delle prestazioni in termini di BER migliori, in quanto, la dimensione dei blocchi di segnale è doppia. Si vuole vedere come si comportano i vari algoritmi di tipo PL con un codice più grande, osservando, in particolare, come cambia la distanza rispetto al Sum Product.

Come nel caso precedente iniziamo con l'analisi di convergenza dell'ADMM.

6.2.1 Convergenza

In Figura 6.5 viene riportato l'andamento di Γ_t per gli stessi valori di c e SNR usati nel caso del codice IEEE 802.11n 648-324.

Le curve ottenute per il nuovo codice, sono molto simili a quelle trovate in Figura 6.1, di conseguenza valgono gli stessi ragionamenti e si sceglie anche in questo caso $c = 10$ e $t^* = 100$.

Per quanto riguarda l'algoritmo ZGCE, invece, le curve vengono riportate in Figura 6.6. In questo caso il grafico con SNR $= 0$ dB è molto simile a quello del codice (648,324) mentre, per SNR $= 4$ dB, per tutti i valori di c, le curve tendono a scendere di più e a non assestarsi intorno ad un valore costante. Tuttavia, facendo delle prove sulla BER, si è trovato che i parametri ottimi sono anche in questo caso gli stessi.

Sulle simulazioni della BER si noterà che, per entrambi gli algoritmi ADMM, i parametri scelti vanno bene per quasi tutti i valori di SNR, ma per quelli più elevati sarebbe necessaria qualche iterazione in più.

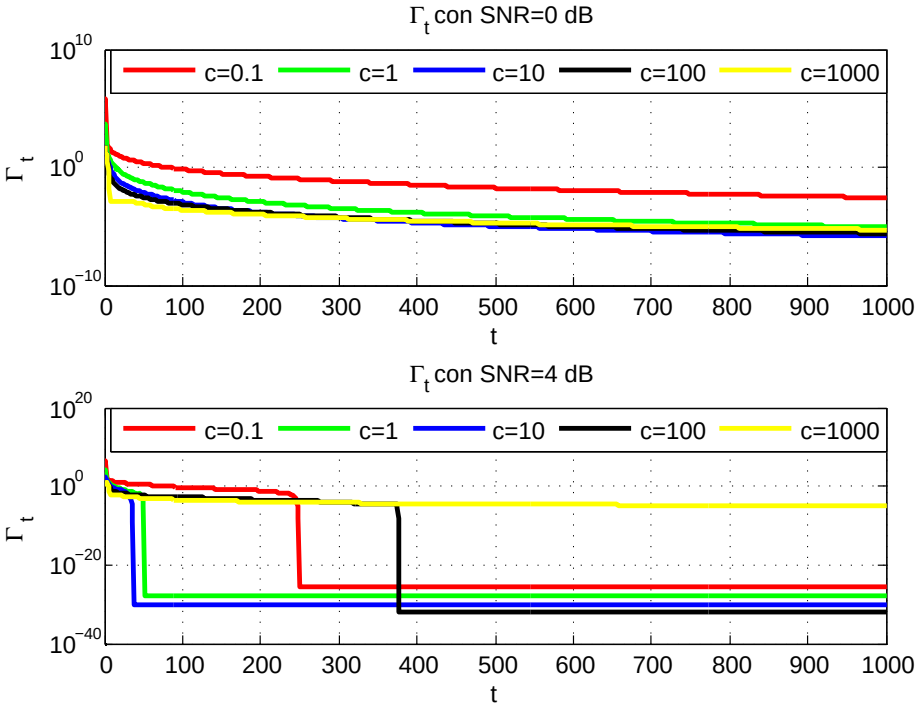


Figura 6.5: Γ_t per algoritmo BLDR e codice LDPC IEEE 802.11n (1296,648)

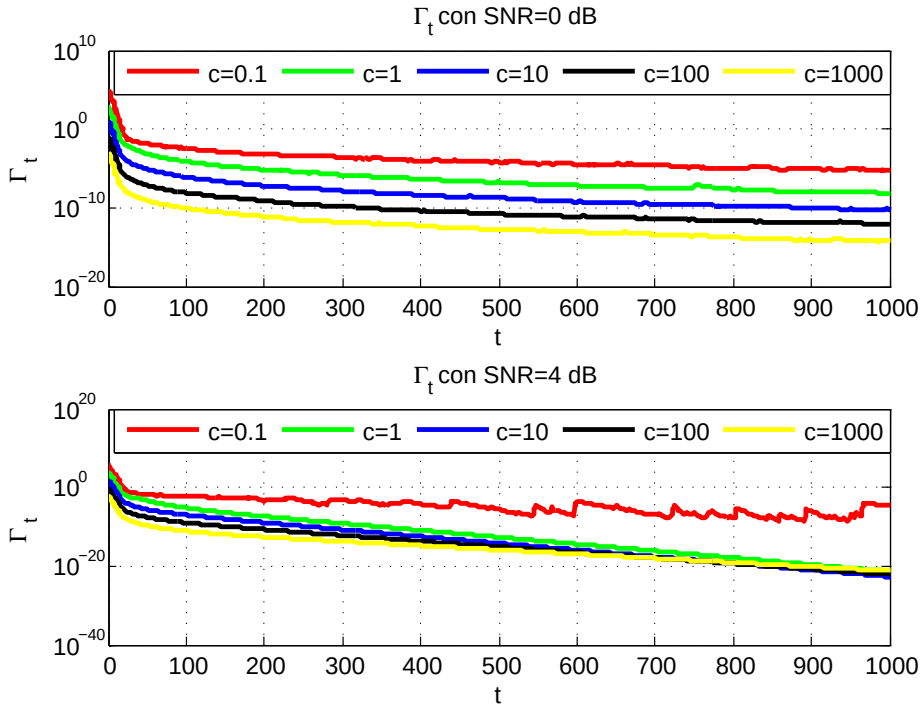


Figura 6.6: Γ_t per algoritmo ZGCE e codice LDPC IEEE 802.11n (1296,648)

6.2.2 Simulazione BER

I parametri scelti per le simulazioni sono gli stessi del caso precedente per tutti gli algoritmi.

In Figura 6.7 viene riportata la simulazione effettuata. Avendo utilizzato

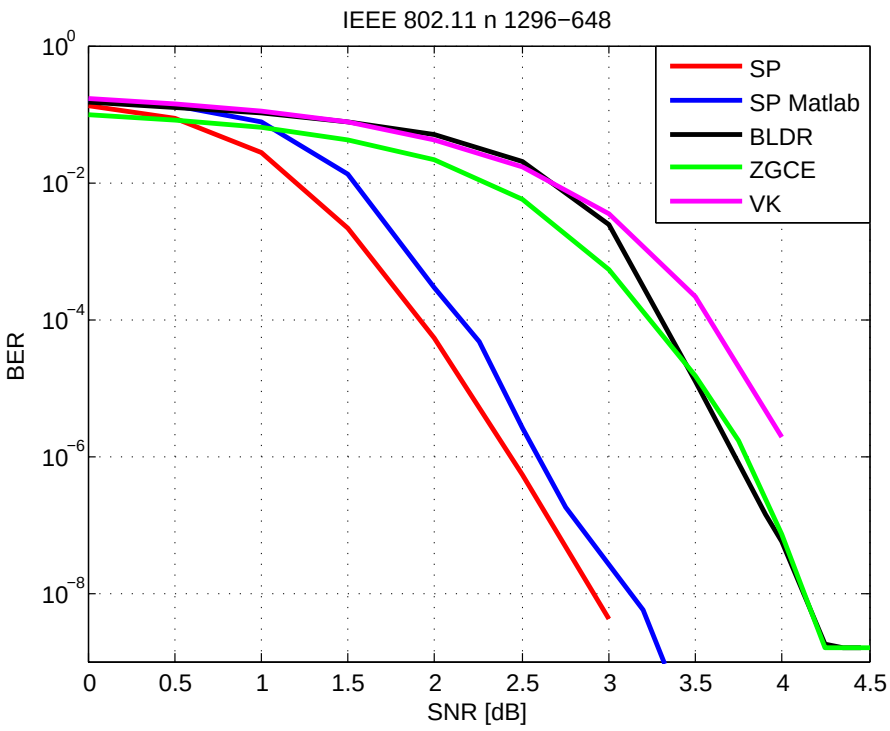


Figura 6.7: BER per il codice LDPC IEEE 802.11n (1296,648)

un codice di dimensione maggiore, le curve risultano spostate più a sinistra rispetto a prima (BER minore). Le relazioni tra i vari algoritmi rimangono quelle descritte in precedenza, con gli algoritmi di tipo PL che sono sempre a circa 1.5 dB dal Sum-Product.

I due algoritmi ADMM sembrano saturare intorno a 4.25dB, in realtà questo è dovuto al fatto che per SNR più elevati servirebbe qualche iterazioni in più per la convergenza. Si è notato che già facendo solo altre 10 iterazioni (ovvero con $t^* = 110$) il problema sparisce.

Come ci si aspettava, sia per questo codice che per il precedente, non si è riusciti a vedere la saturazione della BER per l'algoritmo Sum Product. Come è già stato spiegato in precedenza, la saturazione avviene a BER molto bassi e quindi, per rilevarla, sarebbero state necessarie simulazioni molto piú lunghe.

6.2.3 Confronto prestazioni

Si analizzano ora le prestazioni facendo un grafico che, come prima, metta a risalto contemporaneamente sia i tempi di decodifica che il valore di SNR a cui viene raggiunta la BER di 10^{-6} (Figura 6.8).

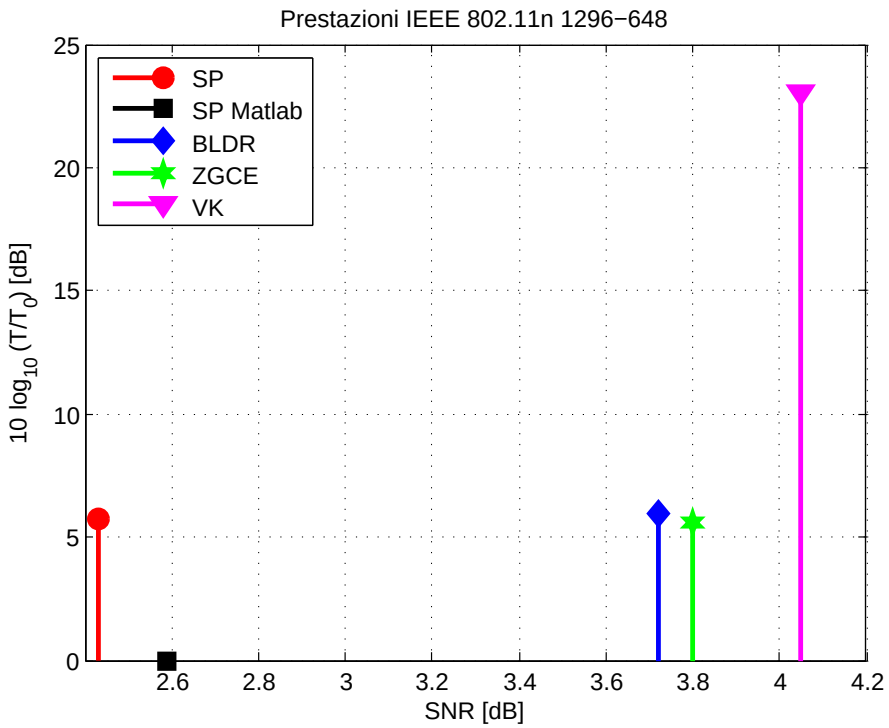


Figura 6.8: Confronto prestazioni per il codice LDPC IEEE 802.11n (1296,648)

Le differenze rispetto al codice precedente sono:

- L'algoritmo BLDR raggiunge la BER di 10^{-6} un po' prima rispetto al ZGCE
- Tralasciando l'algoritmo SP Matlab, preso come riferimento, il ZGCE è quello avente tempo di decodifica minore.
- Il VK continua ad essere molto lontano come tempi rispetto agli altri, ma in questo caso la differenza diminuisce leggermente.

In ogni caso differenze sostanziali tra le prestazioni degli algoritmi, rispetto al caso precedente, non ce ne sono. Si puó quindi affermare che i codici LDPC IEEE 802.11n hanno performance indipendenti dalla dimensione del codice.

6.3 Codici di Margulis

Analizziamo ora le prestazioni del codice di Margulis.

6.3.1 Convergenza

Ancora una volta vengono riportati i grafici di Γ_t per l'algoritmo BLDR (Figura 6.9) e per il ZGCE (Figura 6.10). I valori di c usati sono gli stessi mentre per l'SNR si sono scelti i valori di 0 e 2.5 dB.

I grafici in Figura 6.9 e 6.10 sono molto simili a quelli dei due codici

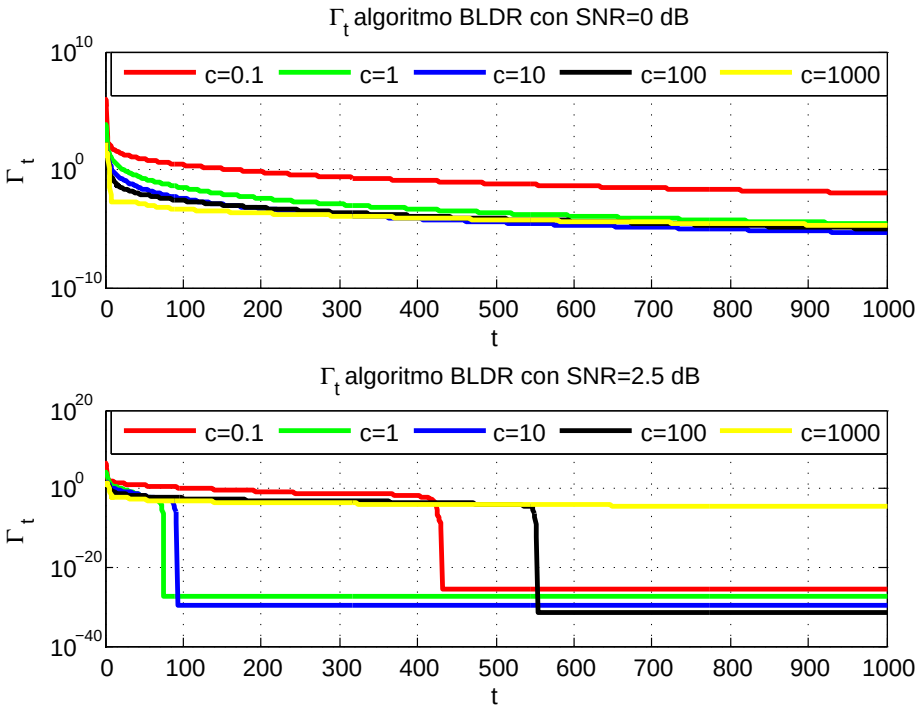


Figura 6.9: Γ_t per algoritmo BLDR e codice di Margulis (2640,1320)

precedenti e quindi si arriva alla stessa scelta di parametri ovvero $c = 10$ e $t^* = 100$.

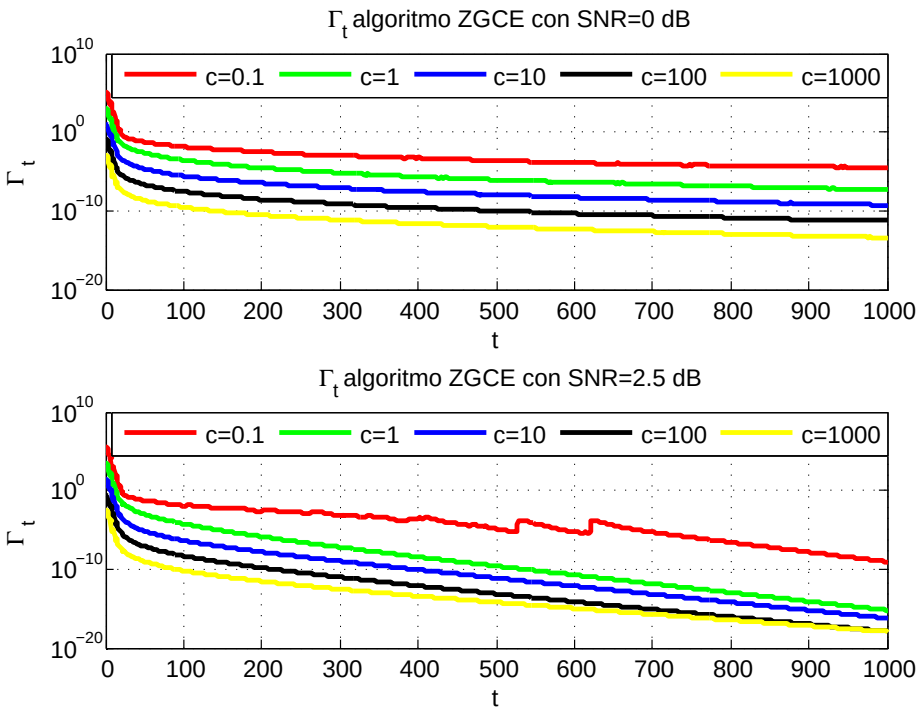


Figura 6.10: Γ_t per algoritmo ZGCE e codice di Margulis (2640,1320)

6.3.2 Simulazione BER

In Figura 6.11 sono riportate le curve di BER per il codice di Margulis.

L'algoritmo Sum Product di Matlab richiede una parity check matrix invertibile in GF(2). Di conseguenza non risulta possibile applicarlo al codice di Margulis, caratterizzato da una parity check matrix generata in modo casuale.

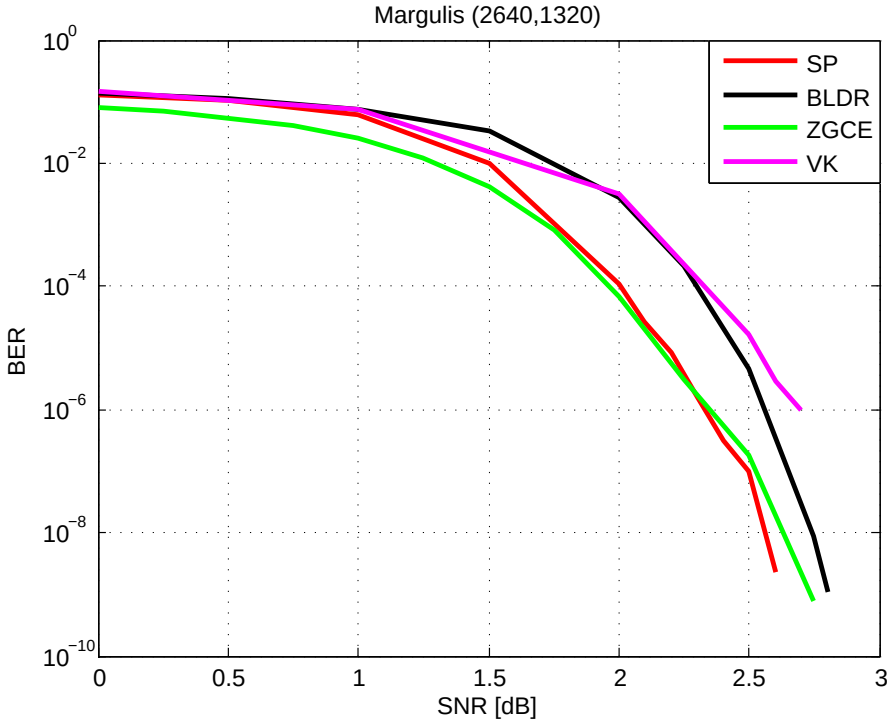


Figura 6.11: BER per il codice di Margulis (2640,1320)

Per il codice di Margulis i risultati in termini di BER sono leggermente diversi dagli altri codici, in particolare:

- Tutte le curve sono più vicine tra loro e la distanza tra i codici PL e il sum product si riduce di molto
- L'algoritmo ZGCE ha una BER prossima a quella del Sum Product e per SNR bassi è addirittura minore

Per concludere l'analisi sulla probabilità d'errore, si osserva che non è stato possibile osservare la saturazione nemmeno per i codici di Margulis, che notoriamente ne soffrono. Ancora una volta sarebbero servite simulazioni troppo lunghe.

6.3.3 Confronto prestazioni

In questo caso, visto che non viene simulato l'algoritmo SP Matlab, non è possibile prenderlo come tempo di riferimento T_0. Si nota che per entrambi i codici IEEE 802.11n si aveva

$$10 \log_{10} \frac{T_{SP}}{T_0} \simeq 5\text{dB}$$

dove si è indicato con T_{SP} il tempo di decodifica dell'algoritmo Sum Product. Ne consegue che $T_{SP} \simeq 3T_0$ e quindi si decide che, per il codice di Margulis, T_0 avrà un valore fittizio dato da

$$T_0 = \frac{T_{SP}}{3}$$

In Figura 6.12 viene riportato il grafico dei confronti, così come spiegato per gli altri codici.

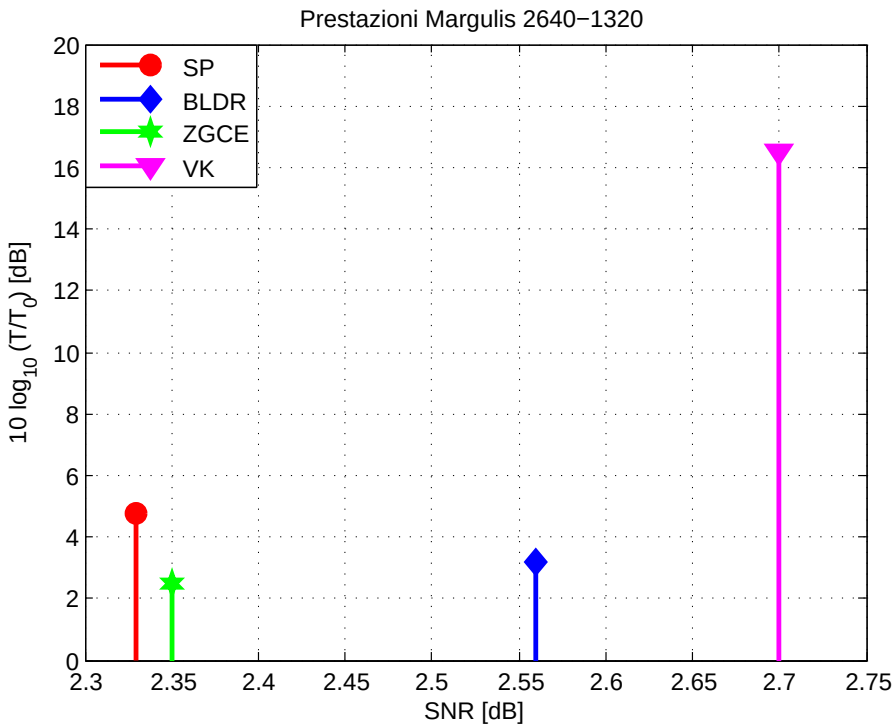


Figura 6.12: Confronto prestazioni per il codice di Margulis (2640,1320)

Si nota subito che, per il codice di Margulis, le prestazioni dei vari codici sono molto vicine tra loro. In particolare:

- L'algoritmo di Vontobel-Kotter è a meno di 0.4 dB dal SP, ma rimane sempre molto lento, anche se meno rispetto ai codici precedenti.

- Il BLDR risulta più veloce del SP ed, inoltre, gli è molto vicino come BER
- Il ZGCE ha prestazioni migliori del SP, in quanto raggiunge circa allo stesso SNR il valore di 10^{-6}, ma ha il vantaggio di impiegare quasi la metà del tempo ad effettuare la decodifica

Quindi per questo particolare tipo di codice non solo gli algoritmi ADMM risultano competitivi con il Sum Product, ma il ZGCE risulta avere prestazioni migliori.

Capitolo 7

Conclusioni

Si sono analizzati vari algoritmi per la decodifica di codici LDPC, con l'obiettivo di vedere se l'utilizzo di tecniche di programmazione lineare (PL) è una valida alternativa all'algoritmo Sum Product.

Nelle simulazioni effettuate ci si è concentrati fondamentalmente su due aspetti: La probabilità d'errore sul bit, al variare del rapporto segnale rumore (SNR), e i tempi di decodifica. Si riassumono ora i risultati ottenuti da entrambi i punti di vista.

- Probabilità d'errore sul bit (BER):
  Si è visto che i risultati dipendono molto dal tipo di codice LDPC usato, infatti, i due codici IEEE 802.11n hanno prestazioni diverse dal Margulis. Per i primi le curve di BER, dei tre algoritmi di tipo PL, sono a $1,5$ dB dal Sum-Product, mentre per il secondo sono molto più vicine. Si è però notato che la differenza tra le prestazioni non dipende tanto dalle dimensioni del codice, infatti i due codici IEEE 802.11n, aventi dimensioni diverse, presentano la stessa distanza tra i vari algoritmi. Invece, per quanto riguarda i codici di Margulis, che sono regolari e hanno una parity check matrix molto meno densa, la decodifica con metodo PL è più vicina a quella SP. Ne consegue che le prestazioni dipendono molto dalla struttura della parity check matrix del codice. In ogni caso, tra tutti i codici analizzati, si è trovato che dal punto di vista della BER tra gli algoritmi PL il più performante sembra il ZGCE, seguito dal BLDR e per ultimo il Vontobel-Koetter. Bisogna comunque specificare, che sotto questo aspetto la differenza è sempre molto contenuta $(0,2 - 0,4)$ dB.

- Tempo di decodifica:
  Per quanto riguarda i codici IEEE 802.11n, i tempi di decodifica sono simili per gli algoritmi SP, BLDR e ZGCE, mentre il VK è poco competitivo in quanto ha tempi di decodifica molto più elevati rispetto agli altri. Per il codice di Margulis, invece, i due algoritmi ADMM sono

più rapidi anche rispetto al Sum Product. Il Vontobel-Koetter è un po' più veloce rispetto al caso dell'IEEE 802.11n, ma rimane ancora poco competitivo.

In generale per tutti i codici analizzati si è visto che la decodifica tramite la programmazione lineare usando la tecnica ADMM, sia per il BLDR che per il ZGCE, risulta avere prestazioni abbastanza vicine all'algoritmo Sum-Product.

I codici IEEE 802.11n sono stati progettati per decodificatori di tipo Sum-Product e quindi era normale attendersi che le performance di tale algoritmo fossero leggermente superiori.

I risultati ottenuti con il codice di Margulis suggeriscono un interessante sviluppo per quanto riguarda l'algoritmo ZGCE, in quanto non solo ha una BER inferiore per SNR bassi a quella del SP, ma soprattutto impiega quasi la metà del tempo per effettuare la decodifica.

Questo suggerisce quindi che, con una parity check matrix adeguata, si potrebbero costruire dei codici LDPC decodificabili con algoritmi veloci, con buone prestazioni e che, essendo basati su tecniche di programmazione lineare, non soffrono la saturazione.

Bibliografia

[1] "IEEE Standard 802.11 n". *IEEE Computer Society*, 2009.

[2] Siddhart Barman, Xishuo Liu, Stark C. Draper, and Benjamin Recht. "Decomposition Methods for Large Scale LP Decoding". *ArXiv e-prints*, April 2012.

[3] D.P. Bersekas and J.N. Tsitsikilis. *"Parallel and Distributed Computation: Numerical Methods"*. Belmont, MA: Athena Scientific, 1997.

[4] Stephen Boyd and Lieven Vandenberghe. *"Convex Optimization"*. Cambridge University Press, 2004.

[5] David Burshtein. "Iterative Approximate Linear Programming Decoding of LDPC Codes With Linear Complexity". *IEEE Transaction on information theory*, 55(11):4835–4859, November 2009.

[6] Tomaso Erseghe. "Channel codes and capacity". URL:moodle.dei.unipd.it/course/view.php?id=1507/, 2012.

[7] Jon Feldman, Martin J. Wainwright, and David R. Karger. "Using Linear Programming to Decode Binaary Linear Codes". *IEEE Transaction on information theory*, 51(3):954–972, March 2005.

[8] Robert G. Gallager. "Low-Density Parity-Check Codes". *Cambridge MA:MIT Press*, 1963.

[9] Thomas J.Richardson, M.Amin Shokrollahi, and Rüdiger L. Urbanke. "Design of Capacity-Approaching Irregular Low-Density Parity-Check Codes". *IEEE Transaction on information theory*, 47(2):619–637, February 2001.

[10] Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. "Factor Graphs and the Sum-Product Algorithm". *IEEE Transaction on information theory*, 47(2):498–519, February 2001.

[11] David J.C. MacKay and Michael S.Postik. "Weaknesses of Margulis and Ramanujan-Margulis Low-Density Parity-Check Codes". *Eletronic Notes in Theorical Computer Science*, 74, 2003.

[12] G.A Margulis. "Explicit costruction of graph without short cycles and low density codes". *Combinatorica*, 2:71–78, 1982.

[13] R.M.Tanner. "A recursive approach to low complexity codes". *IEEE Transaction on information theory*, 27(9):533–547, Semptember 1981.

[14] T.Erseghe, D.Zennaro, E. Dall'Agnese, and L.Vangelista. "Fast consensus by the alternating directions multipliers method". *IEEE Transaction on Signal Processing*, 59(11):5523–5537, November 2011.

[15] Pascal O. Vontobel and Ralf Koetter. "On low-complexity linear-programming decoding of LDPC codes". *European Transaction on Telecommunications*, 2006.

[16] H. Zhu, G.B.Giannakis, and A. Cano. "Distributed in-network channel decoding". *IEEE Transaction on Signal Processing*, 57(10):3970–3983, October 2009.