Facoltà di Ingegneria
Corso di laurea triennale in ingegneria elettronica

Progetto di codici LDPC per comunicazioni a bassa complessità, basato su tecniche di punturazione e accorciamento

Design of LDPC codes for low-complexity communications, based on puncturing and shortening techniques

Candidato:
Greganti Gianmarco

Relatore:
Dr. Battaglioni Massimo

Correlatore:
Chiar.mo Prof. Chiaraluce Franco

Anno Accademico 2023-2024

Facoltà di Ingegneria
Corso di laurea triennale in ingegneria elettronica

Progetto di codici LDPC per comunicazioni a bassa complessità, basato su tecniche di punturazione e accorciamento

Design of LDPC codes for low-complexity communications, based on puncturing and shortening techniques

Candidato:
Greganti Gianmarco

Relatore:
Dr. Battaglioni Massimo

Correlatore:
Chiar.mo Prof. Chiaraluce Franco

Anno Accademico 2023-2024

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di laurea triennale in ingegneria elettronica
Via Brecce Bianche – 60131 Ancona (AN), Italy

A tutti coloro che non smettono di credere in me.

Ringraziamenti

Una menzione particolare va al prof. Massimo Battaglioni che ha reso possibile il miglioramento delle mie competenze nell'ambito della programmazione software e la conoscenza di nuovi contenuti riguardanti la codifica di canale, al prof. Chiaraluce che mi ha permesso di iniziare questo percorso e alla mia famiglia che da sempre mi supporta in ogni scelta che faccio.

Ancona, Settembre 2024

Greganti Gianmarco

Sommario

Un importante vantaggio che si ottiene dalla rappresentazione binaria di un segnale analogico è la possibilità di sfruttare criteri di codifica per compensare l'effetto dei disturbi che si sommano al segnale modulato durante la sua trasmissione attraverso un canale di comunicazione; la compromissione dell'informazione contenuta nel messaggio che si intende inviare, si manifesta in uscita dal demodulatore sotto forma di inversione di uno o più bits. Il ruolo del codice in questo contesto è quello di mettere il ricevitore in condizione di ricevere correttamente la sequenza inizialmente trasmessa; un punto di forza dei codici analizzati in questa tesi è la capacità di conseguire una buona distanza tra le parole di codice generando sequenze pseudo-random mediante un registro a scorrimento e applicando su di esse operazioni di punturazione e accorciamento per ricavare le parole di codice. Dopo un'introduzione teorica ai codici lineari, rivolgendo particolare attenzione ai codici a blocco, verrà analizzata una loro classe detta Low-Density Parity-Check Primitive Rate Compatible (PRC-LDPC), basata sulla teoria dei codici ciclici e sulla definizione di una matrice di parità $\mathbf{H}$ derivata dai coefficienti di un polinomio primitivo. Si identificheranno dei parametri utili alla stima della distanza minima dei codici PRC-LDPC e, a partire dallo studio delle loro caratteristiche, verrà svolta una ricerca di metodi volti all'ottimizzazione delle prestazioni di alcuni di essi a partire da certe specifiche. La valutazione dell'impatto che questi approcci hanno sul codice punturato verrà svolta tramite simulazioni Monte Carlo di trasmissioni modulate e codificate, volte a stimare la probabilità d'errore.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

In un generico sistema di comunicazione, schematizzato in Figura 1, una funzione di rilievo è svolta dal canale, che è il mezzo fisico utilizzato per inviare il segnale di interesse dal trasmettitore al ricevitore (nelle trasmissioni wireless il canale è solitamente lo spazio libero).

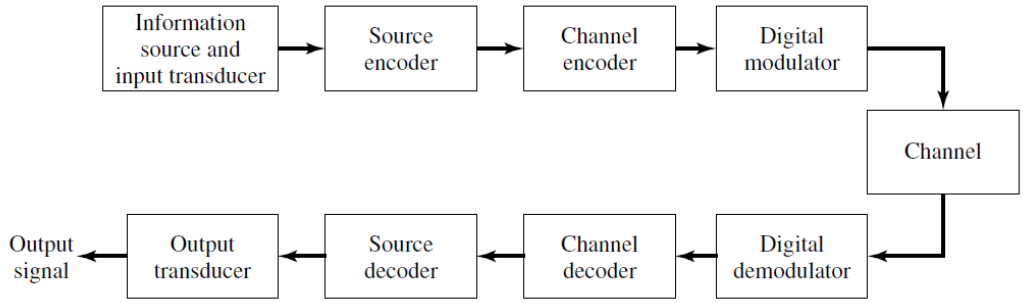


Figura 1.: Schema a blocchi di un sistema di comunicazione digitale [1].

Definizione 0.1. *Per canale continuo si intende un canale che ha come ingressi e uscite dei simboli che possono assumere valori reali, mentre per canale discreto si intende un mezzo i cui simboli in ingresso e in uscita sono finiti o in infinità numerabile.*

Definizione 0.2. *Per canale senza memoria si intende un canale la cui uscita in un certo istante è influenzata solo dall'ingresso in quell'istante e non da eventuali ingressi precedenti.*

Definizione 0.3. *Per canale simmetrico si intende un canale in cui la probabilità di errore sul simbolo è la stessa per tutti i simboli da trasmettere.*

Qualunque sia il mezzo fisico per la trasmissione del segnale, il problema principale è che il segnale trasmesso viene corrotto in modo casuale da vari eventi possibili. La forma più comune di degradazione del segnale si presenta sotto forma di rumore additivo, che viene generato nella parte antecedente il ricevitore. Nella trasmissione wireless, ad esempio, i disturbi additivi aggiuntivi sono il rumore prodotto dall'uomo e il rumore atmosferico captati da un'antenna ricevente. In generale l'obiettivo principale della trasmissione di informazioni su qualsiasi canale di comunicazione è l'affidabilità, misurata come la probabilità di ricezione corretta di un simbolo trasmesso attraverso il canale: supponendo che in ingresso ad esso ci sia un simbolo m e considerando di trasmetterlo attraverso un canale discreto, $p(m$ ricevuto $\mid m$ trasmesso) corrisponde a tale probabilità. Un risultato fondamentale della teoria dell'informazione ricavato da C. Shannon è che una trasmissione affidabile (ovvero una trasmissione con una probabilità di errore inferiore a un certo valore, piccolo a

piacere) è possibile anche su canali affetti da rumore purché il rate di trasmissione abbia come limite superiore la capacità del canale [2].

Esempio 1. *Per fare un semplice esempio, si supponga di considerare la situazione in Figura 2 che rappresenta un canale discreto senza memoria che ha 3 simboli in ingresso e 3 in uscita: se viene ricevuto il simbolo 1 il ricevitore non sa se è stato trasmesso 1 o 2 perché 2 è uno dei possibili simboli trasmessi e la stessa cosa accade se viene ricevuto il simbolo 2, quindi esiste una certa probabilità di commettere un errore sulla determinazione del simbolo trasmesso. Se però tra trasmettitore e ricevitore viene stabilito che il trasmettitore utilizza solo i simboli 0 e 1 (oppure 0 e 2) allora, se viene ricevuto il simbolo 1 o 2, il ricevitore ha la certezza che è stato trasmesso 1 perché 2 non è più un possibile ingresso.*

In altri termini, maggiore è la "distanza" tra gli ingressi maggiore è la probabilità di ricezione corretta. Si accenna ora ad un caso particolare di canale discreto senza memoria detto canale binario simmetrico per vedere in quali condizioni è possibile applicare il risultato visto precedentemente. Essendo definito da un alfabeto di simboli binari, come si evince dalla Figura 3 non è possibile considerare il caso in cui si riesce a determinare con alta probabilità il simbolo trasmesso, quindi è necessario considerare l'estensione del canale binario simmetrico in cui si hanno sequenze binarie di ingresso e di uscita di lunghezza n. A partire dal calcolo delle possibili uscite che differiscono da un certo ingresso in $n \cdot \epsilon$ posizioni (dove ϵ è la probabilità di errore in ricezione e dipende dal canale) si dimostra che il rate di trasmissione R risulta inversamente proporzionale a n e maggiore è questo parametro, minore è la probabilità di errore; la capacità C rappresenta invece il rate massimo di trasmissione per una comunicazione affidabile. Aumentando n fino a un valore tale da raggiungere la condizione $R < C$, è possibile trovare un codice con lunghezza di blocco pari a n capace di portare la probabilità di errore in ricezione a un valore arbitrariamente piccolo, come spiegato in [1].

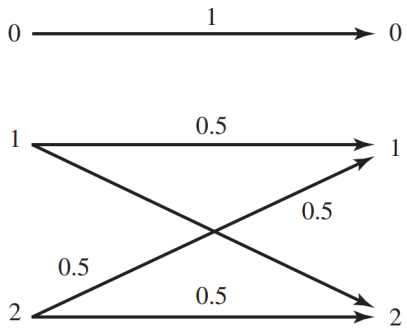


Figura 2.: Esempio di canale discreto [1].

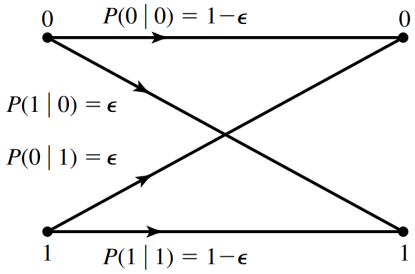


Figura 3.: Canale binario simmetrico [1].

Canale AWGN

Il modello matematico di canale che verrà assunto come riferimento è il canale continuo Additive White Gaussian Noise (AWGN). La relazione che esiste tra la k-esima componente dell'm-esimo segnale trasmesso s_{mk} e la variabile aleatoria r_k in uscita dal demodulatore è $r_k = s_{mk} + w_k$ con $k = 1, \ldots, N_t$ dove w_k è una variabile aleatoria a valore medio nullo e varianza $\sigma_w^2 = \frac{N_0}{2}$.

Esempio 2. *Si immagini di dover trasmettere una sequenza binaria* $\mathbf{t}$*, i cui simboli sono mappati secondo il criterio di modulazione BPSK (0 viene modulato come* $s_1 = -\sqrt{E_b}$ *e 1 come* $s_2 = \sqrt{E_b}$*), allora* $r = \pm\sqrt{E_b} + w$*. La situazione è illustrata in Figura 4.*

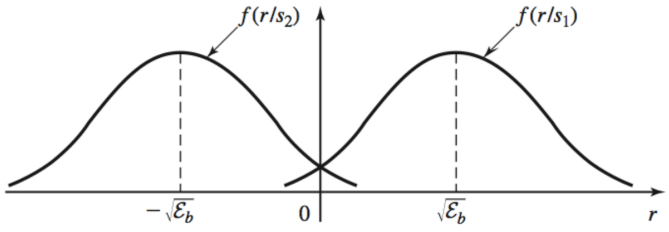


Figura 4.: Probabilità condizionate di due segnali antipodali.

Assumendo come soglia di decisione lo 0, è possibile calcolare la probabilità di errore sul simbolo (che in questo caso coincide con la probabilità di errore sul bit). La legge probabilistica che descrive come il rumore bianco additivo gaussiano interferisce con una trasmissione è la pdf normale, quindi la probabilità di errore sul simbolo risulta essere:

$$P(e \mid s_1) = \int_{-\infty}^{0} f(r|s_1)\, dr = \frac{1}{\sqrt{\pi N_0}} \int_{-\infty}^{0} e^{-\frac{(r-\sqrt{E_b})^2}{N_0}}\, dr = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) = P(e \mid s_2)$$

dove $\sqrt{E_b}$ *e* $\sigma_r^2 = \frac{N_0}{2}$ *sono rispettivamente valore medio e varianza della gaussiana e* N_0 *rappresenta la densità spettrale del rumore. Sotto l'ipotesi di simboli equiprobabili,*

la probabilità di errore sul bit vale:

$$P_b = P(0)P(e \mid s_1) + P(1)P(e \mid s_2) = \frac{1}{2}P(e \mid s_1) + \frac{1}{2}P(e \mid s_2) = Q\left(\sqrt{\frac{2E_b}{N_0}}\right)$$

Caratteristiche e applicazioni dei codici LDPC

Una famiglia di codici utilizzata per la correzione degli errori è quella dei Low-Density Parity-Check, progettati con lo scopo di ottenere un rate di trasmissione che tende alla capacità del canale rumoroso. Essi garantiscono una bassa complessità di co/decodifica, un basso costo in termini di memoria occupata e la loro struttura garantisce l'esistenza di tecniche basate sul miglioramento della loro distanza minima. Questo tipo di codici, tra le altre applicazioni, trovano impiego nelle missioni spaziali in cui sono sfruttati per proteggere ad esempio dei dati scientifici da eventuali interferenze che si verificano nello spazio libero e quindi garantire l'integrità delle informazioni che devono essere ricevute da un ricevitore sulla terra; sono impiegati anche nella navigazione satellitare, in cui offrono un'ottima affidabilità nelle informazioni di posizionamento trasmesse.

Ai fini di questa tesi si sfrutterà una classe di codici LDPC detti PRC [9], ottenuti a partire da codici simplessi. I codici PRC-LDPC sono progettati sulla base di regoli di Golomb e polinomi primitivi, da cui dipende la struttura della matrice di parità. Il design dei codici si baserà sull'utilizzo di strategie chiamate punturazione (che consiste nell'eliminare una colonna per ogni riga tolta nella matrice di parità) e accorciamento (che consiste nel togliere colonne alla stessa), aventi come obiettivi l'aumento del rate del codice e, laddove possibile, la riduzione della probabilità di errore dei bit di informazione in un messaggio. Nel caso del progetto, si lavora a valle della punturazione e si agisce sulla matrice **H** eseguendo combinazioni di accorciamenti diverse; il punto di forza di questa operazione sta nel togliere colonne in determinate posizioni e risulterà evidente che, tramite un accorciamento in posizioni random, non si potrebbero avere gli stessi vantaggi nelle prestazioni.

La discussione si compone della presente introduzione ai fondamenti teorici della codifica di canale, in cui si definiscono le caratteristiche di una comunicazione affidabile con alcuni esempi e un accenno al tipo di canale utilizzato per lo svolgimento del progetto. Nel primo capitolo si analizzano le proprietà dei codici lineari e le caratteristiche di due tipi di codice appartenenti a questa classe: i codici a blocco e i codici convoluzionali; nel secondo capitolo si introducono le caratteristiche dei codici LDPC, il metodo utilizzato per rappresentarli graficamente e le tecniche di decodifica utilizzate per la ricostruzione della parola di codice originaria. L'ultimo capitolo si apre con la definizione delle proprietà dei codici PRC-LDPC e la descrizione delle tecniche di punturazione e accorciamento, prosegue con la ricerca di un modo per valutare la distanza minima di questi codici e si conclude con la progettazione di codici accorciati secondo due criteri e l'analisi delle loro prestazioni, in base a cui verranno tratte alcune considerazioni notevoli.

Capitolo 1.

Codici lineari

1.1. Introduzione ai codici a blocco binari

Un codice a blocco binario (n, k) è completamente definito da $M = 2^k$ sequenze binarie, dette parole di codice, dove k è il numero di cifre di informazione da trasmettere e n la lunghezza del blocco in cui è codificata la corrispondente sequenza di informazione. Un codice C è quindi formato da M parole di codice c_i con $i = 1, \ldots, 2^k$:

$$C = \{c_1, \ldots, c_M\}$$

Definizione 1.1. *Un codice a blocco è lineare se combinando linearmente due parole di codice, la parola risultante è ancora una parola di codice. Nel caso binario, ciò richiede che se c_i e c_j sono due parole di codice allora anche $c_i \oplus c_j$ è una parola $c_l \in C$.*

Sulla base di quanto detto, un codice a blocco lineare è un sottospazio k-dimensionale di uno n-dimensionale. Una possibile tecnica di progetto può essere quella di scegliere le M sequenze binarie in modo da massimizzare la distanza tra loro. A tale proposito, di seguito si definiscono i parametri fondamentali che caratterizzano un generico codice.

Definizione 1.2. *La distanza di Hamming tra due parole di codice c_i e c_j è il numero di posizioni in cui esse differiscono ed è indicata come $d(c_i, c_j)$. Si definisce peso di Hamming (o semplicemente peso) di una parola c_i la sua distanza da quella nulla.*

Definizione 1.3. *La distanza minima di un codice è la minima distanza di Hamming tra due parole di codice diverse:*

$$d_{min} = \min\ d(c_i, c_j)$$

con $i \neq j$. Il peso minimo di un codice è il minimo dei pesi delle parole senza considerare quella formata da tutti zeri:

$$w_{min} = \min\ w(c_i)$$

con $c_i \neq 0$.

N.B. Nel caso dei codici lineari si può dimostrare che $d_{min} = w_{min}$.

1.1.1. Matrice G e matrice H

Queste due rappresentazioni matriciali di un codice lineare sono utili per la codifica e decodifica di un messaggio. In particolare la matrice **G** può essere utilizzata per codificare la sequenza di informazione, mentre la matrice **H** può essere sfruttata in decodifica per rilevare/correggere errori. Si può definire **G** come:

$$\mathbf{G}=\begin{pmatrix}\mathbf{g}_1\\ \mathbf{g}_2\\ \vdots\\ \mathbf{g}_k\end{pmatrix}=\begin{pmatrix}g_{11} & g_{12} & \cdots & g_{1n}\\ g_{21} & g_{22} & \cdots & g_{2n}\\ \vdots & \vdots & \ddots & \vdots\\ g_{k1} & g_{k2} & \cdots & g_{kn}\end{pmatrix}$$

dove $\mathbf{g}_1, \ldots, \mathbf{g}_k$ sono sequenze binarie di lunghezza n linearmente indipendenti e **G** è una matrice $k \times n$. Data una qualunque sequenza di informazione $\mathbf{x}$ tra le 2^k possibili, è possibile ricavare la corrispondente parola di codice $\mathbf{c}$ di lunghezza n mediante la relazione:

$$\mathbf{c} = \mathbf{x} \cdot \mathbf{G}.$$

Esempio 3. *Supponendo di avere un codice lineare (5,2) con la seguente struttura:*

$$C = \{00000, 01110, 10101, 11011\}$$

e di prendere come riferimento le sequenze di informazione 01 e 10, si può rappresentare la matrice **G** *come:*

$$\mathbf{G}=\begin{pmatrix}10101\\ 01110\end{pmatrix}.$$

Sfruttando la relazione precedente si ottiene:

$$\begin{pmatrix}c_1 & c_2 & c_3 & c_4 & c_5\end{pmatrix} = \begin{pmatrix}x_1 & x_2\end{pmatrix} \cdot \mathbf{G}.$$

Risolvendo l'espressione si ricavano le seguenti equazioni che descrivono completamente il codice:
$c_1 = x_1$
$c_2 = x_2$
$c_3 = x_1 \oplus x_2$
$c_4 = x_2$
$c_5 = x_1$

Scegliendo una qualsiasi delle 4 sequenze di informazione, è possibile ricavare la parola di codice a essa corrispondente.

Un noto teorema dell'algebra lineare afferma che, considerando solo le parole dello spazio n-dimensionale ortogonali a quelle del codice, si può individuare un sottospazio lineare $C^\top$ (con $\top$ indicante l'ortogonalità) di quello k-dimensionale con dimensione

$n-k$ tale che:

$$\mathbf{c} \cdot \mathbf{H}^t = \mathbf{0}$$

dove t indica l'operazione di trasposizione e $\mathbf{H}$ è la matrice di parità del codice originale di dimensione $(n-k) \times n$ in cui ogni riga è ortogonale a qualunque $\mathbf{c} \in C$. Se questa relazione non è soddisfatta significa che si è verificato un errore durante la trasmissione, ciò equivale infatti a rilevare la ricezione di una parola non appartenente al codice.

1.2. Codifica di codici a blocco

L'operazione di codifica è svolta a monte del canale e, seguendo la strategia FEC (forward error correction), permette di aggiungere ai k simboli di informazione ulteriori r simboli che non portano informazione ma che sono necessari per differenziare maggiormente le sequenze che possono essere trasmesse; infatti, una sequenza di k bit trasmessa attraverso il canale potrebbe diventare una delle altre sequenze trasmissibili e quindi sarebbe comunque accettata dal ricevitore.

1.2.1. Occupazione spettrale

Oltre a un limite inferiore sulla lunghezza del blocco di codice, è da considerare anche l'aumento al crescere di n della larghezza di banda necessaria a trasmettere una sequenza infatti, dovendo ora considerare n bit nello stesso intervallo di tempo che era riservato ai k bit in assenza di codifica, è necessario che $k \cdot T_b = n \cdot T_b'$ dove T_b era il tempo di bit in assenza di codifica e T_b' è il nuovo intervallo di tempo assegnato a ciascun bit (minore di T_b). Ciò comporta che $F_s = \frac{n}{k} \cdot F_c$ dove F_c era la frequenza di campionamento del segnale analogico da trasmettere (dal quale si ricava la sequenza di informazione) e F_s è la frequenza di simbolo.

Osservazione È evidente che all'aumentare dei bit di ridondanza può aumentare la distanza minima tra le sequenze da trasmettere e quindi solitamente migliora la capacità di correzione degli errori, a costo però di inviare un maggior numero di bit in un'unità di tempo.

1.2.2. Esempi di codici binari a rilevazione e correzione d'errore

Esempio 4. *Un esempio base di codice binario è il codice a bit di parità in cui, data una certa sequenza di informazione di lunghezza k in ingresso al codificatore, in uscita viene fornita una sequenza lunga k+1 in modo che il bit aggiunto renda pari (o dispari, dipende dalla convenzione) il numero di bit della sequenza trasmessa. In questo modo se avviene la transizione di un bit, il blocco in uscita dal canale conterrà un numero di uni dispari e quindi il ricevitore rileva la presenza di un errore poiché riconosce l'assenza di questo blocco tra quelli che potevano essere trasmessi. Un*

problema importante che riguarda questo tipo di codici è la possibilità di rilevare solo un numero dispari di errori (perché se fosse pari si ricadrebbe in una delle sequenze trasmissibili) e inoltre non è in grado di correggerli. Supponendo di essere nel seguente caso:

Informazione	*Ridondanza*
000	*0*
001	*1*
010	*1*
011	*0*
...	...

Identificando con p la probabilità di errore sul singolo bit, è possibile calcolare ad esempio la probabilità di errore non rilevato. Considerando come sequenza trasmessa t=1001 ad esempio: $P_{nr} = P(2\ errori) + P(4\ errori)$
$= P\{1010 \mid t\} + P\{1100 \mid t\} + P\{0000 \mid t\} + ... + P\{0110 \mid t\} = 6 \cdot p^2 \cdot (1-p)^2 + p^4$
essendo 2 bit sbagliati in 6 casi e 4 bit sbagliati in un caso. La probabilità di rilevazione corretta corrisponde invece a $P_r = 1 - P_{nr}$.

Esempio 5. *Un altro esempio di codice binario è il codice a ripetizione, che decide "a maggioranza" il bit trasmesso. Qui il codificatore opera su un bit alla volta e, fissato* $k = 1$*, ripete ogni simbolo della sequenza da trasmettere n volte. Supponendo di inviare la sequenza 0 1 1 0 al codificatore che ripete ogni bit* $n = 4$ *volte, in uscita si avrebbe 0000 1111 1111 0000. Questo codice dispone di una capacità correttiva a differenza di quello precedente infatti nel caso in esame, se si verificasse un solo errore in una qualsiasi delle sequenze, il ricevitore sarebbe in grado di correggerlo perché prende come riferimento i simboli che occorrono più spesso; se invece si verificassero 2 errori, il ricevitore non sarebbe in grado di correggerli ma solo di rilevarne la presenza perché non può decidere a maggioranza in questo caso. Infine se si verificassero 4 errori, il codice non riuscirebbe neanche a rivelarne la presenza in quanto la sequenza ricevuta si confonderebbe con una di quelle che si sarebbe potuta trasmettere. L'inconvenienza nell'utilizzo di questi codici sta nell'evidente richiesta di una banda notevole per la trasmissione e nell'impossibilità di correggere un numero di errori pari o superiore alla metà di n. Si calcola ora la probabilità di errore sulla parola di codice P(r* $\mid t = 1111$*) (analogamente si può calcolare P(r* $\mid t = 0000$*)) :* $P_e = P\{0000 \mid t\} + P\{0001 \mid t\} + ... + P\{1000 \mid t\} = p^4 + 4 \cdot p^3 \cdot (1-p)$ *essendo 3 bit sbagliati in 4 casi e 4 bit sbagliati in un caso. La probabilità di ricezione corretta si calcola invece come* $P_{rc} = 1 - P_e$.

Esempio 6. *L'ultimo esempio riguarda il codice di Hamming (7,4). Questo tipo di codice fornisce la conoscenza della posizione del bit errato e dunque la possibilità di correggerlo attraverso un controllo di parità. Si supponga che il dato da trasmettere sia una sequenza di 4 bit, ad esempio 1011 e si consideri un numero di bit di ridondanza pari a 3, allora i 7 bit che sono effettivamente trasmessi sono* $PP1P011$ *dove le P indicano i bit di parità e sono inserite nelle posizioni 1,2 e 4 dei bit della parola di codice (numerati dal bit più significativo al meno significativo). La scelta di questa disposizione per i bit di controllo risiede nel metodo di codifica utilizzato: il bit di parità nella posizione 1 controlla la parità dei bit che si trovano nelle posizioni le cui rappresentazioni binarie terminano con un 1 (in binario XXX1, con X don't care oppure in numeri interi: 1,3,5 e 7) e quindi nel caso in esame la P in posizione 1 dovrà assumere valore 0; il bit di controllo nella posizione 2 controlla la parità dei bit che si trovano nelle posizioni 2,3,6 e 7 (XX1X), di conseguenza il bit in posizione 2 dovrà assumere valore 1; il bit di Hamming nella quarta posizione controlla la parità dei bit nelle posizioni 4,5,6 e 7 (X1XX), in base a ciò la P nella quarta posizione assumerà 0 come valore. La parola codificata è dunque* $\mathbf{01}1\mathbf{0}011$*; immaginando che si verifichi un errore nella quinta posizione, il ricevitore esegue l'operazione EX-OR sui bit corrispondenti a ciascun set di posizioni: il primo bit di parità esegue l'EX-OR sulle posizioni dispari ottenendo come risultato 1, il secondo bit di controllo lo esegue sulle posizioni 2,3,6 e 7 ottenendo uno 0 che viene posto alla sinistra del bit ricavato precedentemente e analogamente viene ricavato il bit 1 da affiancare alla sequenza 01. In questo modo si ottiene la sequenza 101 che in numero intero corrisponde alla posizione dove si è verificato l'errore, ma il problema legato a questo tipo di codice riguarda l'impossibilità di correggere più di un errore.*

1.2.3. Codici ciclici

Definizione 1.4. *Un codice ciclico è un codice a blocco lineare con la seguente proprietà: data una qualsiasi parola di codice* $\mathbf{c}$*, un suo shift ciclico produce un'altra parola di codice.*

Per semplificare l'analisi dei codici ciclici, si può considerare ciascuna parola di codice $\mathbf{c} = [c_1, c_2, \ldots, c_{n-1}, c_n]$ come un polinomio:

$$c(q) = \sum_{i=1}^{n} c_i q^{n-i}$$

dove i è il numero di shifts. Considerando la parola di codice traslata di una posizione a sinistra $\mathbf{c}^1 = [c_2, c_3, \ldots, c_n, c_1]$ la si può scrivere in termini polinomiali come:

$$c^1(q) = c_2 q^{n-1} + c_3 q^{n-2} + \cdots + c_n q + c_1$$

oppure, sfruttando l'operazione EX-OR, come:

$$c^1(q) = qc(q) + c_1(q^n + 1) = qc(q) \pmod{(q^n + 1)}$$

Se si trasla di una posizione a sinistra $\mathbf{c}^1$, l'espressione di $\mathbf{c}^2$ è analoga a quella scritta precedentemente e così via fino a che $i = n$:

$$c^n(q) = q^n c(q) \pmod{(q^n + 1)} = (q^n + 1)c(q) + c(q) \pmod{(q^n + 1)} = c(q)$$

Teorema 1.2.1. *In un qualunque codice ciclico* (n, k) *ogni polinomio relativo a una certa parola di codice è multiplo di un polinomio generatore di grado* $n - k$*:*

$$g(q) = q^{n-k} + g_2 q^{n-k-1} + g_3 q^{n-k-2} + \cdots + g_{n-k} q + 1$$

dove $q^n + 1$ *deve essere divisibile per* $g(q)$*. Una qualsiasi sequenza di informazione* $\mathbf{x}$ *può essere espressa in modo polinomiale come:*

$$X(q) = x_1 q^{k-1} + x_2 q^{k-2} + \cdots + x_{k-1} q + x_k$$

dove $x_1, \ldots, x_k$ *possono assumere valore 0 o 1. Infine il polinomio della parola di codice associata a una sequenza* $\mathbf{x}$ *è esprimibile nel seguente modo:*

$$c(q) = X(q) \cdot g(q)$$

Esempio 7. *Supponendo di voler generare un codice ciclico (6,2) è necessario considerare un polinomio generatore di grado 4. Poiché* $q^6 + 1 = (q^2 + 1)(q^4 + q^2 + 1)$ *l'unico polinomio di quarto grado che divide* $q^6 + 1$ *è* $q^4 + q^2 + 1$*. Quindi in questo caso* $g(q) = q^4 + q^2 + 1$ *e lo si moltiplica per tutti i polinomi* $X(q)$ *della forma* $X(q) = x_1 q + x_2$ *in modo da poter ricavare le parole codificate in Tabella 1.1.*

Ingresso	$X(q)$	$c(q) = X(q) \cdot g(q)$	Parola codificata
00	0	0	000000
01	1	$q^4 + q^2 + 1$	010101
10	q	$q^5 + q^3 + q$	101010
11	$q + 1$	$q^5 + q^4 + q^3 + q^2 + q + 1$	111111

Tabella 1.1.: Parole codificate del codice ciclico $(6, 2)$.

1.3. Principi di decodifica di codici a blocco

L'operazione di decodifica è realizzata a valle del canale in base a due possibili criteri: la decodifica a decisione soft (che fornisce una stima più dettagliata della probabilità di errore in quanto opera sui numeri reali in uscita dal canale e dal demodulatore) e la decodifica a decisione hard; in via di principio, entrambi possono essere metodi "a massima verosimiglianza", poiché il loro scopo è minimizzare la probabilità di decodifica di una parola di codice errata quando tutte le parole di codice hanno la stessa probabilità di essere trasmesse. Si supponga di voler trasmettere il messaggio codificato con la modulazione BPSK attraverso il canale AWGN. Nelle Sezioni 1.3.1 e 1.3.2 si presentano i due metodi di decodifica.

1.3.1. Decodifica soft

Considerando il vettore $\mathbf{r}$ in uscita dal demodulatore, esso viene confrontato con tutte le possibili parole di codice e viene scelta quella che esibisce la distanza euclidea più piccola rispetto al vettore ricevuto.

Esempio 8. *[1] Si analizzi il codice a ripetizione (3,1) che consiste nelle due parole di codice 000 e 111. Si supponga che esse siano trasmesse utilizzando la modulazione PSK binaria con energia per simbolo unitaria. Il vettore ricevuto è* $\mathbf{r}$ *= (0.5, 0.5,-3). Se viene implementata la decodifica a decisione soft, si deve confrontare la distanza euclidea tra* $\mathbf{r}$ *e i due punti di coordinate (-1, -1, -1) e (1,1,1) e scegliere quello più vicino al vettore ricevuto prendendo come soglia di decisione lo 0. Si ha quindi:*
$d^E(\mathbf{r}, (1, 1, 1)) = 0.5^2 + 0.5^2 + 4^2 = 16.5$
$d^E(\mathbf{r}, (-1, -1, -1)) = 1,5^2 + 1,5^2 + (-2)^2 = 8.5$

di conseguenza un decodificatore di questo tipo decodificherà **r** *come (-1,-1,-1) (cioè la parola 000).*

1.3.2. Decodifica hard

In questo caso invece viene selezionata la parola di codice che esibisce la minima distanza di Hamming rispetto a un vettore **y** ricavato a partire dal vettore ricevuto.

Esempio 9. *[1] Ogni componente di* **r** *viene confrontata con la soglia 0: se essa è maggiore di zero assumerà valore 1, se è minore di zero assumerà valore 0. Nel caso in esame il vettore che deve essere confrontato con le sequenze (0,0,0) e (1,1,1) è quindi* **y** *= (1,1,0). Sulla base di ciò, è evidente che il ricevitore stimerà come messaggio trasmesso la parola (1,1,1) poiché questa e* **y** *differiscono per una sola posizione.*

Osservazione In generale per un qualsiasi codice a blocco si può dimostrare che, sfruttando la decodifica hard, il numero di errori correggibili è:

$$t = \begin{cases} \frac{d_{\min}-1}{2} & \text{se } d_{\min} \text{ è dispari.} \\ \frac{d_{\min}-2}{2} & \text{se } d_{\min} \text{ è pari.} \end{cases}$$

Il numero di errori rilevabili è invece $t = d_{min} - 1$.

1.4. Cenni ai codici convoluzionali

Sebbene la conoscenza di questa famiglia di codici lineari non sia strettamente essenziale per il lavoro che seguirà, si introducono per completezza i codici convoluzionali, che presentano alcune caratteristiche comuni ai codici a blocco. Una di queste riguarda, almeno in parte, la codifica. Infatti, si ha che anche in questo caso ai k bits di informazione si aggiungono r bits di ridondanza in modo che la parola di codice risultante contenga n bits, ma mentre nei codici a blocco la codifica di ogni blocco avveniva indipendentemente da quella di tutti gli altri, nei codici convoluzionali i bit di questo blocco non sono determinati solo dai bit di informazione attuali ma anche da quelli precedenti. In Figura 1.1 è riportato il diagramma a blocchi di un codificatore convoluzionale, che consiste in un registro a scorrimento con kL stadi dove L è detta constraint length. In ogni unità temporale, k bits di informazione entrano nel registro e il contenuto degli ultimi k stadi viene perso. Successivamente vengono calcolate n combinazioni lineari in base alla scelta degli stadi per generare la parola codificata. Da questo schema è facile notare che le uscite dell'n-encoder non dipendono solo dai k bit più recenti che sono entrati nell'encoder, ma anche dai $(L-1)k$ contenuti dei primi $(L-1)k$ stadi prima dell'arrivo dei k bit attuali. Sulla base di quanto detto, è possibile considerare questo registro come una macchina a stati finiti con $2^{(L-1)k}$ stati. Un'altra caratteristica che accomuna i codici a blocco e i codici convoluzionali è il rate, infatti per ogni sequenza di informazione si ha una sequenza codificata e quindi esso risulta coincidere con il rapporto tra k e n.

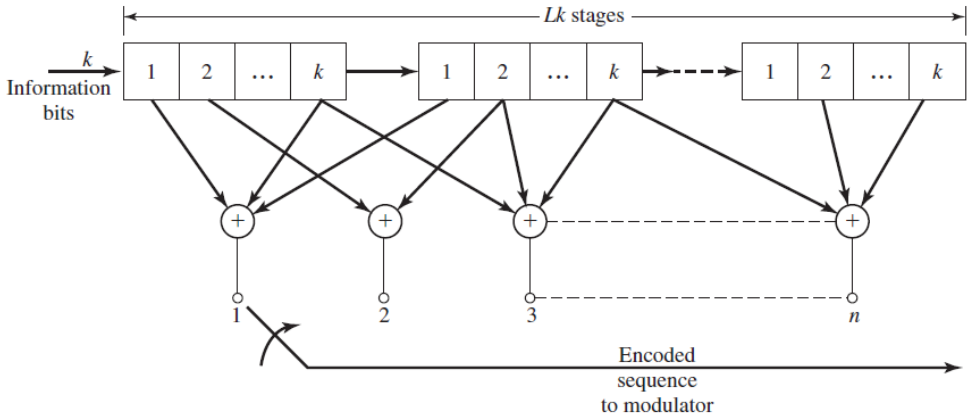


Figura 1.1.: Codificatore convoluzionale.

Come evidenziato in [6], una classe di codici convoluzionali detta SC-LDPC (Spatially Coupled) è particolarmente affine a una classe di codici a blocco descritta nel capitolo 3, detta PRC-LDPC. Considerando ad esempio la seguente matrice di

parità di un codice a blocco:

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

si può ricavare la sottomatrice di un codice convoluzionale tempo-variante periodico:

$$\mathbf{H}_t^{(m)} = \begin{pmatrix} 1 & & & \\ 0 & 1 & & \\ 0 & 1 & 0 & \\ 1 & 0 & 0 & 1 \\ & 0 & 1 & 0 \\ & & 1 & 0 \\ & & & 1 \end{pmatrix}$$

dove $t = 0, 1, \dots$, $m = 0, \dots, M$ e il periodo è $T = 4$. La matrice complessiva $\mathbf{H}^T$ si trova disponendo in diagonale un set di sottomatrici $\mathbf{H}_t^{(m)}$ variabili a ogni istante t e in numero pari a $M + 1$ (dove M è memoria del codice); questa struttura è molto simile a quella dei codici PRC rappresentata in (3.1) e, come nei codici a blocco ci si aspetta una crescita della distanza minima all'aumentare di n, nei codici convoluzionali a un aumento di M deve corrispondere lo stesso risultato.

Capitolo 2.

Codici LDPC

I codici LDPC sono una classe di codici sviluppati per la prima volta da R. Gallager nel 1962 [4], utilizzati nelle comunicazioni moderne per le seguenti caratteristiche: determinano una buona distanza minima tra le parole di codice e grazie alla loro struttura si possono usare schemi di decodifica iterativa relativamente semplici, che comunque garantiscono prestazioni vicine ai limiti teorici.

Definizione 2.1. *Un codice LDPC è un codice a blocco lineare definito da una matrice di parità sparsa, ovvero il numero di zeri all'interno di essa deve essere molto maggiore del numero di uni.*

Definizione 2.2. *Un codice LDPC è detto regolare se la matrice di parità che lo descrive contiene esattamente w_c uni in ciascuna colonna e $w_r = \frac{n}{n-k} \cdot w_c$ uni in ciascuna riga.*

Definizione 2.3. *Un codice LDPC è detto irregolare se il numero di uni in ciascuna riga o colonna della matrice $\boldsymbol{H}$ non è costante.*

La differenza principale tra i codici a blocco classici e i codici LDPC sta nel metodo usato per la decodifica dell'informazione. Per i codici a blocco classici con lunghezza di blocco piccola si è già visto che si utilizzano criteri di decodifica a massima verosimiglianza (ML), per i codici LDPC si sfruttano invece algoritmi di decodifica iterativa che lavorano sulla base della rappresentazione grafica della loro matrice $\mathbf{H}$.

Osservazione Siccome la complessità dell'algoritmo di decodifica è proporzionale al numero di uni per colonna, avere un basso $\langle w_c \rangle$ permette di considerare parole di codice con n grande, fattore che generalmente aiuta nel raggiungimento di prestazioni ottime.

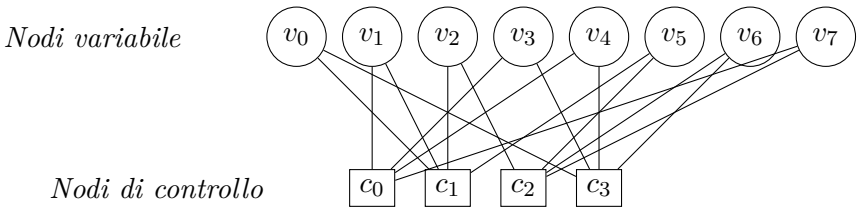
2.1. Rappresentazione grafica

La rappresentazione grafica di un codice LDPC può essere determinata tramite un grafo detto grafo di Tanner [5] che è costruito a partire da due tipi di nodi: i nodi di controllo (c-nodes) che sono tanti quanti sono i bit di ridondanza e i nodi variabile (v-nodes) che sono in numero pari alla lunghezza della parola di codice. La regola che stabilisce i collegamenti tra c-nodes e v-nodes è la seguente: un c-node in posizione j è collegato a un v-node in posizione i se $h_{ji} = 1$; non ci sono invece collegamenti tra nodi della stessa classe.

Esempio 10. *[7] Si consideri la seguente matrice di parità che descrive un codice LDPC regolare:*

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

Il suo grafo è:



Definizione 2.4. *Un percorso che inizia da un nodo e termina sullo stesso nodo è detto ciclo, la cui lunghezza è pari al numero di connessioni. La lunghezza del ciclo più breve è detto girth.*

Nell'Esempio 10 il girth del grafo è pari a 4. Un esempio di ciclo di lunghezza 4 è: $v_3 \to c_3 \to v_4 \to c_0 \to v_3$. Esiste un vincolo che deve essere soddisfatto dalla matrice di parità per ottenere un buon codice LDPC (cioè il cui algoritmo di decodifica a decisione soft non dia problemi di convergenza), chiamato RCC (Row-coloumn constraint) il quale stabilisce che all'interno di **H** non devono presentarsi degli uni disposti ai vertici di una "figura rettangolare", ciò si traduce nell'evitare cicli di lunghezza 4 nel grafo di Tanner.

2.2. Decodifica di codici LDPC

L'operazione di decodifica è un processo iterativo che prevede l'utilizzo di una delle seguenti teniche: decodifica a decisione hard o a decisione soft; un esempio di algoritmo a decisione hard è il "bit-flipping" mentre un algoritmo che sfrutta la decisione soft è il "sum-product". La differenza principale tra i due è che nel "sum-product" a ogni bit è associata una probabilità di essere 0 o 1, mentre nel "bit-flipping" si decide il valore di ogni bit a maggioranza.

2.2.1. Decodifica a decisione hard: algoritmo bit-flipping

Nel caso del bit flipping, ogni nodo di controllo collegato a un certo insieme di nodi variabile riceve l'informazione da essi inviata e, in base a questa, esegue un test di parità: per ogni gruppo di v-nodes, se la somma dei bits inviati da ciascuno di essi fornisce come risultato un numero dispari (equazione di parità non soddisfatta), viene incrementato di un'unità il numero di occorrenze dei v-nodes coinvolti in quell'equazione. Dopo aver calcolato tutte le equazioni di parità, il bit che presenta il maggior numero di occorrenze viene invertito. L'algoritmo si ripete fino a che tutte le equazioni non sono soddisfatte o quando viene raggiunto un numero massimo di iterazioni.

Esempio 11. *Prendendo come riferimento l'Esempio 10, si supponga di voler trasmettere la parola di codice 11001000 e che in uscita dal demodulatore si presenti invece la parola* $\mathbf{y} = 10001000$.

Nodi variabile	Occorrenze nelle equazioni errate	Decisione
v_0	$1 \xrightarrow{1} 0$	1
v_1	$2 \xrightarrow{0\to1} 0$	1
v_2	$1 \xrightarrow{0} 0$	0
v_3	$1 \xrightarrow{0} 0$	0
v_4	$1 \xrightarrow{1} 0$	1
v_5	$1 \xrightarrow{0} 0$	0
v_6	$0 \xrightarrow{0} 0$	0
v_7	$1 \xrightarrow{0} 0$	0

Tabella 2.1.: Il valore a sinistra di $\to$ indica il numero di occorrenze contate in base ai controlli di parità svolti su $\mathbf{y}$; il valore a destra indica invece il numero di occorrenze ottenuto dopo l'inversione del bit in posizione v_1 su $\mathbf{y}$. In questo caso, subito dopo la prima inversione tutte le equazioni di parità sono soddisfatte, quindi l'algoritmo termina fornendo la parola trasmessa.

2.2.2. Decodifica a decisione soft: algoritmo SPA-LLR

L'LLR-SPA, descritto in [8], calcola la probabilità a posteriori (indicante la probabilità che un bit assuma valore 0 o 1) per ciascun bit della parola di codice trasmessa, a partire dalla sequenza ricevuta e dai vincoli imposti dal codice.

Definizione 2.5. *Si indica con $N(i)$ il set di connessioni del nodo variabile v_i, che contiene le label c-nodes connessi al nodo v_i:*

$$N(i) = \{j \in \{1, ..., n-k\} : h_{ji} = 1\}$$

Si indica con $M(j)$ il set di connessioni del nodo di controllo c_j, contenente le label dei nodi variabile connessi al nodo c_j:

$$M(j) = \{i \in \{1, ..., n\} : h_{ji} = 1\}$$

Nel caso dell'Esempio 10 si ha quindi: $N(0) = \{1,3\}$, $N(1) = \{0,1\}$, $N(2) = \{1,2\}$,.., $N(7) = \{0,2\}$; $M(0) = \{1,3,4,7\}$, $M(1) = \{0,1,2,5\}$, $M(2) = \{2,5,6,7\}$, $M(3) = \{0,3,4,6\}$.

Definizione 2.6. *Si denoti con $q^{(v)}_{i\to j}(x)$, $x \in \{0,1\}$ l'informazione che v_i invia a c_j indicante la probabilità che il nodo v_i assuma 0 o 1 in base a tutti i controlli di parità che riguardano v_i escluso c_j. Con $q^{(c)}_{i\leftarrow j}(x)$, $x \in \{0,1\}$ si denoti invece l'informazione che c_j manda a v_i indicante la probabilità che il nodo v_i assuma 0 o 1 in base ai controlli di parità che effettua c_j su tutti i nodi escluso v_i.*

N.B. La lettera tra parentesi si riferisce a chi computa l'informazione mentre la freccia indica il flusso del messaggio.

L'informazione probabilistica LLR (Log-Likelihood Ratio) di una variabile aleatoria binaria è definita come:

$$L(X) = \log \frac{P(X=0)}{P(X=1)}$$

Denotando con $\mathbf{y}$ la parola ricevuta corrispondente alla parola di codice $\mathbf{u}$, si definiscano: $L^{(v)}_{i\to j}(u_i) = \log \frac{q^{(v)}_{i\to j}(0)}{q^{(v)}_{i\to j}(1)}$ e $L^{(c)}_{i\leftarrow j}(u_i) = \log \frac{q^{(c)}_{i\leftarrow j}(0)}{q^{(c)}_{i\leftarrow j}(1)}$.

Di seguito si descrivono i passi dell'algoritmo:

1. **Inizializzazione:** A ogni nodo variabile è associata una probabilità *a posteriori* LLR $L(u_i) = \log\{P(u_i = 0|y_i)/P(u_i = 1|y_i)\}$. Per ogni posizione (j, i) tale $h_{ji} = 1$, vengono calcolati i messaggi che vanno dai nodi variabile ai nodi di controllo
$$L^{(v)}_{i\to j}(u_i) = L(u_i)$$
$$L^{(c)}_{i\leftarrow j}(u_i) = 0$$

2. **Aggiornamento dei nodi di controllo:** Per ogni c_j e per ogni $v_i \in M(j)$, vengono calcolati i messaggi da inviare a ogni nodo variabile escludendo di volta in volta l'informazione che era stata portata dal nodo i-esimo, il quale ora riceve il seguente dato
$$L^{(c)}_{i\leftarrow j}(u_i) = 2\tanh^{-1}\left\{\prod_{i'\in M(j)\setminus i}\tanh(L^{(v)}_{i'\to j}(u_{i'})/2)\right\}$$

3. **Aggiornamento dei nodi variabile:** Per ogni v_i e per ogni $c_j \in N(i)$, vengono calcolati i messaggi da inviare a ogni nodo di controllo senza considerare ogni volta l'informazione che era stata portata dal nodo j-esimo (nello step 2), il quale ora riceve il dato nella seguente forma
$$L^{(v)}_{i\to j}(u_i) = L(u_i) + \sum_{j'\in N(i)\setminus j} L^{(c)}_{i\leftarrow j'}(u_i)$$

4. **Informazione LLR totale:** Infine per ogni v_i, si esegue il calcolo
$$L^{(v)}_{i}(u_i) = L(u_i) + \sum_{j\in N(i)} L^{(c)}_{i\leftarrow j}(u_i)$$

5. **Decisione:** Utilizzando una BPSK in cui 0 è mappato come $+1$ e 1 come -1, si quantizza $\mathbf{u}$ come segue: $\hat{u}_i = 0$ se $L^{(v)}_i(u_i) \geq 0$, and $\hat{u}_i = 1$ if $L^{(v)}_i(u_i) < 0$. Se è verificata l'espressione $\hat{\mathbf{u}} \cdot \mathbf{H}^t = 0$, dove ^{t} denota la trasposizione, l'algoritmo si arresta e si assume $\hat{\mathbf{u}}$ come parola decodificata; altrimenti si ritorna al passo 2. Se, dopo un numero massimo prefissato di iterazioni il processo di decodifica non termina, ne viene dichiarato il fallimento.

Capitolo 3.

Codici PRC-LDPC

3.1. Introduzione ai codici PRC-LDPC

Questa classe di codici, introdotta in [9], è descritta da una matrice di parità la cui struttura dipende da un polinomio primitivo (da qui l'aggettivo Primitive) che, per definire un codice LDPC, deve rispettare alcuni vincoli, uno dei quali è il vincolo RCC accennato nel capitolo precedente. L'aggettivo Rate-Compatible indica la possibilità di adattare il codice a diverse esigenze di trasmissione, cioè si può scegliere di avere un buon rate di trasmissione oppure una buona capacità di correzione degli errori in base al tipo di canale considerato, senza riprogettare completamente il codice. Per raggiungere il rate desiderato, si può combinare l'utilizzo di due tecniche che verranno descritte in seguito: punturazione e accorciamento. In questo capitolo si riprendono molte delle nozioni introdotte nel capitolo 1 e se ne definiscono altre ai fini del progetto che sarà presentato nell'ultima sezione di questo capitolo.

Definizione 3.1. *Si definisce vettore di coefficienti* $\mathbf{h}$ *un vettore binario di lunghezza* $k+1$ *che ha come elementi i coefficienti* $h_i \in \{0,1\}$ *di un polinomio di parità* $h(x) = \sum_{i=0}^{k} h_i x^i$ *in cui* w_h *uni sono disposti in corrispondenza delle posizioni* $\{i \in [0,k] \mid h_i = 1\}$. *Si definisce supporto del vettore* $\mathbf{h}$ *un vettore che ha come elementi le posizioni* i *e come numero di elementi* w_h.

Definizione 3.2. *Si definisce regolo di Golomb un vettore di numeri interi non negativi in cui la differenza tra due elementi qualsiasi è diversa per ogni altra coppia di valori considerata.*

Definizione 3.3. *Si definisce* $\mathbf{H}$ *in questo caso una matrice* $r \times N$ *a rango pieno composta da* $N-k$ *ripetizioni dello stesso vettore* $\mathbf{h}$ *(associato al vettore di coefficienti di un polinomio primitivo) ognuna shiftata a destra di una posizione a ogni riga, con numero medio di uni per colonna pari a* $\langle w_c \rangle$.

La struttura della matrice di parità di un codice PRC-LDPC è rappresentata nel sequente modo:

$$\mathbf{H} = \begin{pmatrix} h_0 & h_1 & \cdots & h_k & & & \\ & h_0 & h_1 & \cdots & h_k & & \\ & & \ddots & \ddots & \ddots & & \\ & & & h_0 & h_1 & \cdots & h_k \end{pmatrix} \tag{3.1}$$

Questo tipo di codici sono progettati a partire da codici ciclici il cui polinomio generatore $g(x)$ è ottenuto da un polinomio di parità $h(x)$ dividendo $x^N + 1$ per il reciproco del polinomio, definito come $h^*(x) = x^k h(x^{-1})$. Se $h(x)$ è primitivo, esso descrive un codice simplesso ciclico che verrà denominato "codice padre" (indicato con $C_{\mathbf{p}}$) con lunghezza di ciascun blocco pari a $N = 2^k - 1$, rate $R = \frac{k}{2^k-1}$, distanza minima $d_{min} = 2^{k-1}$ e le cui parole di codice non nulle sono sequenze pseudo-random ottenute da N shifts ciclici della sequenza $\mathbf{p} = [\mathbf{g}, \mathbf{0}_{k-1}]$ associata a $g(x)$, avente periodo massimo N.

Osservazione Condizione necessaria e sufficiente al soddisfacimento del vincolo RCC in un codice PRC-LDPC è che il supporto del vettore **h** sia un regolo di Golomb, perché questo rende impossibile la formazione di cicli di lunghezza 4 nella matrice di parità, come dimostrato in [9].

3.1.1. Punturazione

La punturazione può essere utilizzata per il design di una matrice **H** che descrive un codice a rate più alto rispetto a quello di $C_{\mathbf{p}}$, a costo di una riduzione della distanza minima (essendo tolti bit di controllo). Supposto un certo polinomio $h(x)$, esso descrive una famiglia di un codice PRC caratterizzato da N parole di codice la cui lunghezza del blocco è compresa nell'intervallo $[k+1, 2^k+1]$; con la punturazione si va a considerare una finestra di lunghezza $n = N - \rho$ (dove ρ è il numero di punturazioni effettuate), che idealmente scorre ciclicamente di una posizione su **p**, individuando a ogni shift una nuova parola di codice di lunghezza $n < N$. Il codice risultante, talvolta chiamato "codice figlio" (indicato con $C_{\mathbf{p}}(n)$), conterrà quindi N parole lunghe n e, ovviamente, la parola nulla. In Figura 3.1 si mostra l'effetto della punturazione sulla matrice **H**.

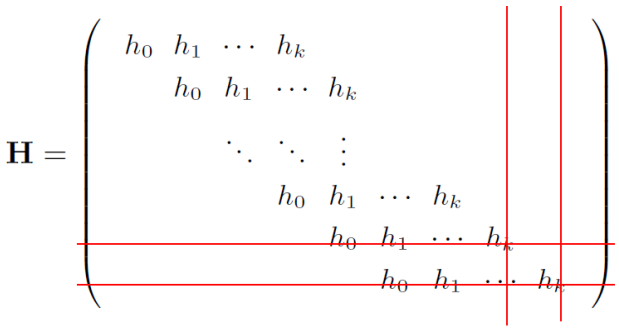


Figura 3.1.: Eliminazione delle ultime due righe e ultime due colonne sulla matrice di parità ($\rho = 2$) [9].

3.1.2. Accorciamento

Questa operazione si utilizza con l'obiettivo di progettare un codice a rate più basso rispetto a quello di $C_{\mathbf{p}}(n)$ in modo da aumentare potenzialmente la distanza minima del codice risultante, accettando un minor rate di trasmissione (essendo eliminati bit di informazione). Di conseguenza a valle dell'accorciamento si ottiene un codice con lunghezza di blocco pari a $n_s = n - s$, dove s è il numero di accorciamenti effettuati. In Figura 3.2 si mostra un esempio di accorciamento di due posizioni su $\mathbf{H}$.

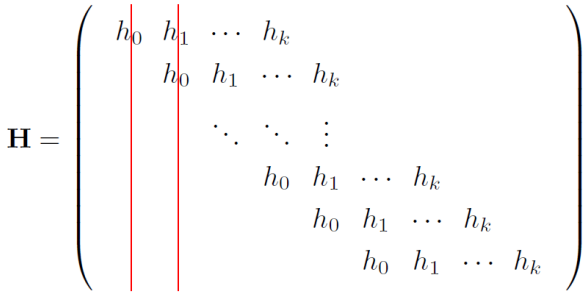


Figura 3.2.: Eliminazione delle prime due colonne sulla matrice di parità ($s = 2$) [9].

3.2. Studio della distanza minima dei codici simplessi punturati

In questa sezione si introducono degli strumenti utili alla stima della distanza minima di un codice $C_{\mathbf{p}}$ in cui k ha un valore talmente grande da rendere impossibile l'analisi della sequenza $\mathbf{p}$ completa, come mostrato anche in [11].

Con $A(w, n)$ si indica il numero di codewords di peso w e lunghezza n, mentre per distanza minima $d_{min}(n)$ si intende il più piccolo valore positivo di w tale che $A(w, n) > 0$. Si ponga ora l'attenzione su una sottosequenza di $\mathbf{p}$ di lunghezza $n < L + Z_r + Z_l + 2 < N$, avente la seguente struttura:

$$\tilde{p} = [1, \mathbf{0}_{Z_l}, 1 \cdots 1_L, \mathbf{0}_{Z_r}, 1]$$

dove L è la lunghezza della sottosequenza centrale che ha come estremi i due uni interni, entro i quali possono trovarsi sia uni che zeri; Z_l e Z_r sono invece rispettivamente il numero di zeri a sinistra e a destra della sottosequenza centrale.

Definizione 3.4. *Si definisce famiglia di parole di codice un insieme di codeword di lunghezza $n \in [L, L + Z_r + Z_l]$ che possono essere ottenute le une dalle altre mediante uno shift non ciclico su $\tilde{p}$.*

Si indichi con $A_i(w, n)$ il numero di parole di codice appartenenti alla famiglia i-esima.

Esempio 12. *Si supponga di considerare come sequenza $\tilde{p}$ la seguente:*

$$\cdots 1|0000|101|00|1 \cdots$$

con $L = 3$, $Z_l = 4$ e $Z_r = 2$. Assumendo $n = L + Z_r + 1 = 6$ ad esempio, si può valutare il parametro $A_1(2, 6)$ intercettando prima di tutto la prima parola di codice appartenente alla famiglia 1 (evidenziata in verde)

$$\cdots 1|0000|101|00|1 \cdots \tag{3.2}$$

e poi si passa all'esecuzione degli shifts non ciclici per trovare le altre codewords della stessa. Se si prova a traslare la finestra lunga 6 di una posizione a destra, si osserva facilmente che la parola trovata (in rosso) non appartiene alla famiglia 1 non avendo lo stesso pattern della (3.2).

$$\cdots 1|0000|101|00|1 \cdots$$

Considerando invece traslazioni a sinistra della stessa finestra, si possono identificare altre due parole di codice della famiglia 1 e una appartenente ad una famiglia diversa:

$$\cdots 1|0000|101|00|1\cdots$$

$$\cdots 1|0000|101|00|1\cdots$$

$$\cdots 1|0000|101|00|1\cdots$$

Dunque si può concludere che $A_1(2,6) = 3$.

In generale a seconda della lunghezza e della struttura delle parole di codice considerate, si definisce $A_i(w,n)$ come:

$$A_i(w,n) = \begin{cases} n - L + 1 & \text{per } L \leq n \leq L + Z_r \\ 1 + Z_r & \text{per } L + Z_r < n \leq L + Z_l \\ 1 + Z + L - n & \text{per } L + Z_l < n \leq L + Z \\ 0 & \text{altrove} \end{cases}$$

con $Z_l \geq Z_r$ e $Z = Z_r + Z_l$. Il caso trattato nell'Esempio 12 rientra nella situazione $L + Z_r < n \leq L + Z_l$.

N.B. $A(w,n) = \sum_i A_i(w,n)$

Indicando con $d_{max}(n)$ il valore più grande di w per cui $A(w,n) > 0$, si possono introdurre alcuni parametri adatti a svolgere un'analisi della distanza minima del codice $C_{\mathbf{p}}(n)$ a partire da una certa lunghezza di blocco. Si definiscono a tal proposito:

$$\Delta(n) = \frac{d_{max}(n) - d_{min}(n)}{2};\ \langle d(n)\rangle = \frac{d_{max}(n) + d_{min}(n)}{2}$$

Quindi si può scrivere che:

$$d_{min}(n) = \langle d(n)\rangle - \Delta(n);\ d_{max}(n) = \langle d(n)\rangle + \Delta(n) \tag{3.3}$$

dove $\Delta(n)$ indica lo scostamento tra la distanza media e la distanza minima. Si definisce infine:

$$\delta(n) = \langle d(n)\rangle - w(n) \tag{3.4}$$

lo scostamento tra la distanza media e il peso $w(n)$ di una codeword, dove

$$w(n) = \begin{cases} \frac{n}{2} & \text{per } n \leq \frac{N-1}{2} \\ \frac{n}{2} + \frac{1}{2} & \text{per } \frac{N-1}{2} < n \leq N \end{cases}$$

In altre parole punturando minimo $\frac{N+1}{2}$ bits nel codice $C_{\mathbf{p}}$, si ottengono codewords di peso $\frac{n}{2}$ in media; punturando massimo $\frac{N-1}{2}$ bits oppure non eseguendo la punturazione, si ricavano parole di codice con peso $\frac{n}{2} + \frac{1}{2}$ mediamente. Combinando le relazioni (3.3) con la (3.4), si ottengono le seguenti espressioni:

$$d_{min}(n) = w(n) + \delta(n) - \Delta(n)$$

$$d_{max}(n) = w(n) + \delta(n) + \Delta(n)$$

da cui si può intuire che nel caso ideale la distanza minima sarebbe pari a $w(n)$; nel caso reale sono presenti anche le quantità $\Delta(n)$ e $\delta(n)$, la cui valutazione per certi valori di n favorisce una buona stima della distanza minima. Per fare un'analisi approssimativa, si osservi la tabella in Figura A.1 nell'Appendice 1 che mostra i valori di $d_{min}(n)$, $d_{max}(n)$, $\Delta(n)$ e $\delta(n)$ calcolati per $n_1 = \frac{N-1}{2}$ e $n_2 = \frac{N+1}{2}$, sulla base di alcuni polinomi di parità con k e w_h diversi. Come si evince dalla tabella, all'aumentare di k si assiste ad un aumento di $\Delta(n)$, mentre il valore di $\delta(n)$ oscilla poco; a ogni incremento di k di un'unità la distanza minima assume un valore maggiore del doppio di quella ottenuta con il valore di k precedente essendo predominante il fattore 2^k in $w(n)$. Considerando un numero molto più grande di campioni di n, si può concludere che $\Delta(n)$ è una funzione pari rispetto a $\frac{N}{2}$ e rappresenta uno scostamento grande rispetto a $\delta(n)$, quindi tra i due è il parametro che impatta maggiormente sulla diminuzione della distanza minima. Per quanto riguarda $\delta(n)$, essa è invece una funzione dispari rispetto a $\frac{N}{2}$ e il suo valore può crescere o decrescere nell'intorno di questo punto, quindi può contribuire o meno all'aumento della distanza minima per certi valori di n.

3.3. Metodi per il miglioramento delle prestazioni di codici simplessi punturati e simulazioni numeriche

In questa sezione si mostra come l'azione dell'accorciamento combinata a quella della punturazione nei codici simplessi, produca risultati vantaggiosi soprattutto laddove si sfruttino codifiche con elevati rate (in cui la distanza minima è piccola). La possibilità di eliminare tutte le parole di codice a distanza $d_{min}(n)$ e dunque aumentare la nuova distanza minima di una o più unità è un obiettivo particolarmente interessante; questa operazione infatti ha come potenziale risultato la riduzione del cosiddetto BER floor (lenta discesa del BER in un certo intervallo di E_b/N_0). In particolare, grazie a dei codici sviluppati in ambiente matlab, si ricaveranno matrici di parità attraverso due strategie di accorciamento, delle quali poi si analizzeranno i pro e i contro. Il punto di partenza di questo lavoro è la scelta del fattore di accorciamento s, del Golomb iniziale (di cui si devono definire w_h e k) e la ricerca di un polinomio primitivo ad esso associato per la costruzione della matrice $\mathbf{H}$ di partenza.

3.3.1. Accorciamento casuale

Il metodo descritto in questa sottosezione consiste nel togliere colonne in maniera casuale dalla matrice di parità, verificando a ogni accorciamento che non ci siano righe o colonne nulle (ciò infatti causerebbe rispettivamente la mancanza di alcune equazioni di parità oppure una valutazione errata della parità da parte di certi nodi di controllo). Nel caso si ottenga una matrice composta da vettori riga o colonna nulli, va cercato un altro set di valori in base al quale accorciare finchè non si trova quello per cui viene rispettata la condizione. L'obiettivo di questo metodo è quello di analizzare gli effetti sulle prestazioni prodotti da un accorciamento approssimativo, che comunque toglie in modo randomico alcune parole di codice anche se di peso qualsiasi. La porzione di codice nella sezione B.2 dell'Appendice B mostra gli effetti di un accorciamento casuale multiplo sulla matrice di parità.

3.3.2. Accorciamento selettivo

Per applicare questo approccio è necessaria la conoscenza del numero di parole di codice di basso peso, oltre che del peso e del supporto di ognuna di esse. È quindi opportuno usare il programma in [13], che trova per ogni distanza il numero di parole di peso basso e i relativi supporti (che il software salva in un file di formato .cw). L'obiettivo dell'accorciamento selettivo è quello di eliminare idealmente tutte le parole di codice del codice simplesso punturato (caratterizzato dal coefficiente $A(w, n)$, introdotto nella sezione 3.2) con peso pari alla distanza minima togliendo colonne da $\mathbf{H}$ in corrispondenza delle posizioni degli uni che compongono le parole di peso minimo, le quali possono potenzialmente generare un BER floor. Data una codeword $\mathbf{c}$ a distanza minima, l'accorciamento selettivo consiste nel rimuovere una colonna in posizione $i \in \mathit{Supporto}(\mathbf{c})$, facendo sì che la nuova matrice di parità assuma dimensione $(n-k) \times (n-1)$ e che il polinomio $h(x)$ perda un coefficiente in diverse righe: ciò equivale a scartare tutte le parole del codice punturato per le quali $i \in \mathit{Supporto}(\mathbf{c})$. La Figura 3.3 mostra l'effetto dell'accorciamento selettivo sulla matrice di parità iniziale.

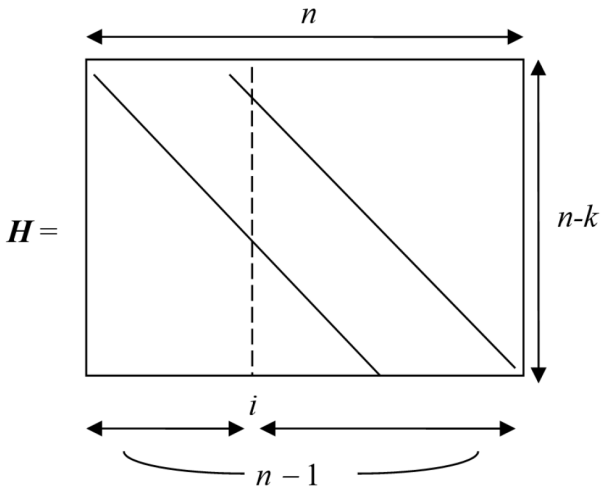


Figura 3.3.: Matrice risultante da un accorciamento selettivo [12].

Il risultato finale è il dimezzamento del numero totale di parole di codice (si passa da 2^k a 2^{k-1}), riducendo con buona probabilità il BER floor. Per fornire dei fondamenti teorici dell'algoritmo che verrà descritto successivamente, si introducono alcune notazioni: sia $i \in [0, n-1]$ e si considerino tutte le parole di codice di peso d_{min} che contribuiscono ad $A(d_{min}, n)$. Si distinguano tre casi: intersezione dei loro supporti vuota, contenente un singolo elemento o contenente più elementi; dato un set di codewords $\mathcal{D} = \mathbf{c}_0, \ldots, \mathbf{c}_{D-1}$, si indichi con $b_i^{(\mathcal{D})}$ il coefficiente avente come valore il numero di occorrenze dell'elemento $i \in Supporto(\mathcal{D})$ e si definisca $I_j^{(\mathcal{D})} = \{i \in [0, k-1] \mid b_i^{(\mathcal{D})} = j\}$; ad esempio $I_0^{(\mathcal{D})}$ corrisponde all'insieme delle posizioni che non corrispondono a elementi di un supporto, $I_1^{(\mathcal{D})}$ all'insieme delle posizioni degli uni presenti in un solo supporto e così via.

Per conseguire lo scopo stabilito precedentemente, si propone ora una soluzione di accorciamento ottima che permette di eliminare il maggior numero di parole di un peso specificato. A partire dal numero di posizioni da accorciare desiderato s, i passi dell'algoritmo in [12] sono i seguenti:

1. Si imposta $d = d_{\min}$.

2. Si cerca di trovare il maggior numero di parole di codice possibile di peso d (in questo caso si usa il software in [13]) e si separano poi le codewords in famiglie secondo il criterio definito nella sezione 3.2.

3. Per ogni famiglia, si trasla ciclicamente a destra la sequenza che la caratterizza, controllando a ogni shift se il vettore $\mathbf{v}$ ottenuto è una codeword attraverso l'espressione $\mathbf{v} \cdot \mathbf{H}^t = 0$; se questa non viene rispettata, la parola viene scartata. Il processo si ripete finchè uno shift non causa la condizione $v_{n-1} = 1$.

4. Per ogni famiglia, si trasla ciclicamente a sinistra la sequenza che la caratterizza, controllando a ogni shift se il vettore $\mathbf{v}$ ottenuto è una codeword attraverso l'espressione $\mathbf{v} \cdot \mathbf{H}^t = 0$; se questa non viene rispettata, la parola non viene considerata. Il procedimento deve essere ripetuto finchè uno shift non causa la condizione $v_0 = 1$.

5. Si costruisce il set $\mathcal{D}$, contenente tutte le parole di codice trovate negli steps 2, 3 e 4.

6. Si suddivide l'intervallo $[0, k-1]$ come $\bigcup_{j=0}^{M} I_j^{(\mathcal{D})}$, dove $j \in [0, M]$ tale che $I_j^{(\mathcal{D})} \notin \{\emptyset\}$.

7. Si accorcia di s posizioni il codice considerato prendendole prima in $I_M^{(\mathcal{D})}$, poi in $I_{M-1}^{(\mathcal{D})}$, fino a $I_1^{(\mathcal{D})}$. Se per qualche $j \in [1, M]$ il numero residuo di posizioni da accorciare s_r è più piccolo del numero di elementi in $I_j^{(\mathcal{D})}$, si scelgono casualmente s_r elementi di $I_j^{(\mathcal{D})}$ e si eliminano colonne da $\mathbf{H}$ in base a quelli. Se $s > \left|\bigcup_{j=0}^{M} I_j^{(\mathcal{D})}\right|$ (dove $|\cdot|$ indica il numero di elementi dell'insieme) dopo aver accorciato tutte le posizioni presenti nell'insieme $\bigcup_{j=1}^{M} I_j^{(\mathcal{D})}$, si fissano $d = d_{\min} + 1$, $s = s - \left|\bigcup_{j=0}^{M} I_j^{(\mathcal{D})}\right|$ e si riparte dallo step 2.

N.B. Lo step 2 non garantisce di trovare tutte le parole di codice di una certa famiglia, di conseguenza si introducono gli step 3 e 4 che sono necessari per una ricerca esaustiva delle codewords all'interno di ogni famiglia, ma che potrebbero comunque non essere sufficienti a trovare tutte quelle del peso (basso) desiderato.

Riguardo l'implementazione dell'algoritmo descritto in questa sottosezione si faccia riferimento al codice nella sezione B.3 dell'Appendice B.

3.3.3. Risultati

In questa sottosezione vengono forniti dei risultati nella forma di un confronto tra le prestazioni di alcuni codici prima e dopo l'applicazione dei due tipi di accorciamento descritti precedentemente al variare di alcuni parametri. Per ottenere i dati in termini di BER (in funzione di E_b/N_0) da paragonare, si eseguono simulazioni Monte Carlo di trasmissioni modulate con BPSK sul canale AWGN (modello riconducibile con buona approssimazione ad un canale BSC, vedi Figura 3, con bit error rate pari a $\epsilon = P_b$) con decodificatore basato sull'algoritmo LLR-SPA introdotto in 2.2.2 che esegua, al massimo, 100 iterazioni.

A parità di $k = 553$ e considerando il seguente regolo di Golomb con $w_h = 27$, come punto di partenza del progetto per ottenere polinomi di parità di pesi anche minori:

$$\begin{bmatrix} 0, 3, 15, 41, 66, 95, 97, 106, 142, 152, 220, 221, 225, 242, \\ 295, 330, 338, 354, 382, 388, 402, 415, 486, 504, 523, 546, 553 \end{bmatrix}$$

si supponga di voler progettare tre codici con i seguenti rate di trasmissione R_s: 6/7, 5/6 e 3/4; si parte dai codici punturati rispettivamente a rate 553/639, 553/656 e 553/724 e si accorciano poi di un fattore $s = 41$. Le specifiche per la determinazione della lunghezza dei blocchi di codice originali sono: $k = 553$, $s = 41$ e R_s; siccome $R_s = k_s/n_s$, si può calcolare $n_s = (k - s)/R_s$ per ricavare $n = n_s + s$. Per evitare di avere colonne troppo pesanti (e quindi prevenire un innalzamento della complessità di decodifica), si considerano per ogni rate valori di w_h decrescenti al diminuire di n in base alla relazione $\langle w_c \rangle = (1 - R) \cdot w_h$. I risultati prestazionali offerti dai codici punturati e dai relativi codici accorciati sono riportati nelle Figure 3.4, 3.5 e 3.6.

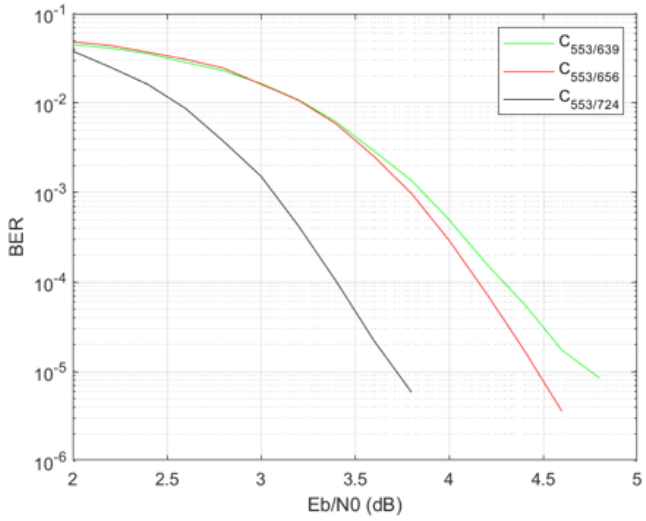


Figura 3.4.: Confronto tra le prestazioni dei codici punturati.

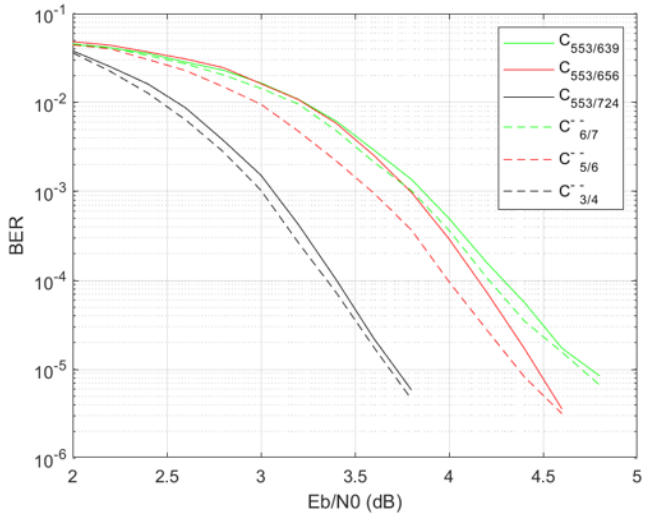


Figura 3.5.: Confronto tra le prestazioni dei codici originali (punturati) e quelle dei codici accorciati in base all'approccio presentato in 3.3.1 con fattore di accorciamento $s = 41$.

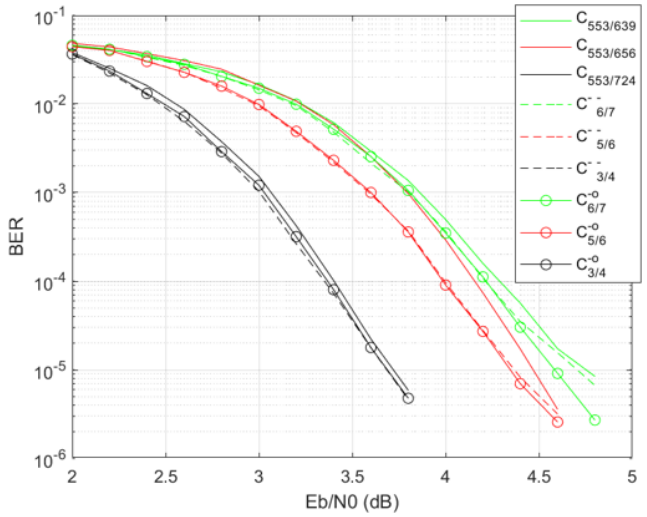


Figura 3.6.: Confronto tra le prestazioni dei codici originali e quelle relative ai codici dopo l'applicazione dei due tipi di accorciamento descritti in 3.3.1 e 3.3.2, considerando come fattore di accorciamento $s = 41$.

Le curve del grafico in Figura 3.6 evidenziano che un accorciamento mirato alle parole di codice ha la possibilità di ridurre notevolmente un BER floor rispetto a un accorciamento casuale soprattutto nei codici a rate alto, ciò è confermato dalle prestazioni della curva verde —○— relativa al codice $C_{6/7}$.

Codice punturato						
Codice	n	R	Regolo di Golomb primitivo	s	d_{min}	$A(d_{min})$
$C_{553/639}$	639	553/639	0 3 41 66 95 97 106 142 152 220 221 225 242 295 330 354 382 388 402 415 486 504 523 546 553	-	3	16
$C_{553/656}$	656	553/656	0 3 15 41 97 106 142 152 220 242 295 338 382 388 402 415 486 504 523 546 553	-	4	9
$C_{553/724}$	724	553/724	0 3 66 97 142 220 221 295 330 354 382 402 486 546 553	-	8	29
Codice accorciato						
Codice	n_s	R_s	Regolo di Golomb primitivo	s	d^s_{min}	$A(d^s_{min})$
$C_{6/7}$	598	6/7	"	41	4	37
$C_{5/6}$	615	5/6	"	41	5	13
$C_{3/4}$	683	3/4	"	41	9	40

Tabella 3.1.: Ogni blocco di ciascun codice punturato presenta $k = 553$ cifre di informazione, mentre variano i parametri n e w_h.

Come sperato, dopo aver accorciato il numero di colonne a n_s mediante la strategia ottima, la nuova distanza minima di ogni codice preso in esame è diventata $d^s_{min} = d_{min} + 1$, come testimonia la Tabella 3.1. Infine in Figura 3.7 si mostrano altri risultati che testimoniano l'efficienza dell'eliminazione ottima delle colonne pur non aumentando in certi casi la distanza minima del codice. In questo caso le specifiche sono le stesse, salvo uno dei due fattori di accorciamento e il valore w_h è fissato; la determinazione di n è analoga a quella precedente.

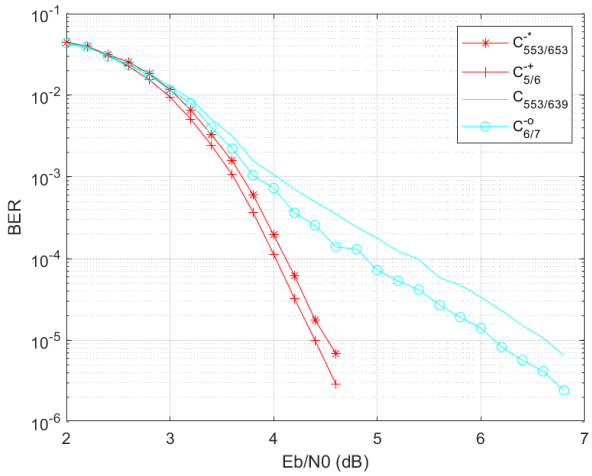


Figura 3.7.: Confronto tra le prestazioni dei codici originali e quelle dei codici a rate 5/6 e 6/7, che hanno subito un accorciamento rispettivamente di 57 e 41 posizioni.

Codice punturato							
Codice	k	n	R	Regolo di Golomb primitivo	s	d_{min}	$A(d_{min})$
$C_{553/653}$	553	653	553/653	0 3 15 41 97 106 142 152 220 242 295 338 382 388 402 415 486 504 523 546 553	-	3	2
$C_{553/639}$	553	639	553/639	0 3 15 41 66 95 97 106 142 152 220 221 225 242 295 330 338 382 415 486 553	-	2	84
Codice accorciato							
Codice	k_s	n_s	R_s	Regolo di Golomb primitivo	s	d^s_{min}	$A(d^s_{min})$
$C_{5/6}$	496	596	5/6	"	57	5	10
$C_{6/7}$	512	598	6/7	"	41	2	27

Tabella 3.2.: Ogni blocco di ciascun codice punturato presenta $k = 553$ cifre di informazione, mentre variano i parametri n ed s.

Accorciando di 41 posizioni il codice $C_{6/7}$, il coefficiente $A(d_{min})$ passa dal valore 84 a 27. Accorciando invece $C_{5/6}$ in 57 posizioni, si ottiene un miglioramento della distanza minima di due unità.

Capitolo 4.

Conclusioni

L'elaborato ha trattato il design di codici PRC-LDPC con l'obiettivo di diminuire il fenomento del BER floor, che si presentava soprattutto in codici simplessi punturati dotati di un rate elevato. Chiaramente, si è posta l'attenzione sull'eliminazione delle parole di basso peso, così da massimizzare la distanza minima del codice, quando possibile; ciò si traduce in un'eliminazione strategica delle colonne della matrice di parità. Una tecnica valida che permette di perseguire tale obiettivo (diminuendo il rate del codice di partenza) è l'accorciamento, tuttora oggetto di ricerca; in questa tesi ne sono state proposte due versioni: accorciamento casuale e accorciamento selettivo. In Tabella 4 vengono riassunti gli aspetti positivi e negativi di questi due metodi, nell'ottica di valutarli nel contesto dell'affidabilità delle comunicazioni.

Accorciamento casuale	**Accorciamento selettivo**
Vantaggi	**Vantaggi**
1. Lieve miglioramento delle prestazioni rispetto al codice di partenza. 2. Non necessita di tempo per la ricerca delle parole di codice, è prevista solo l'attesa di un breve tempo di elaborazione della matrice di parità accorciata grazie alla semplicità dell'algoritmo di accorciamento.	1. Alta probabilità di avere un notevole miglioramento delle prestazioni soprattutto nei codici con rate alti, grazie ad una sufficiente conoscenza della struttura, del numero e del peso delle parole di peso più piccolo.
Svantaggi	**Svantaggi**
1. Nessuna informazione sul supporto, sul numero e sul peso delle codewords eliminate.	1. Attesa di un tempo pari alla somma di quello necessario a trovare le parole di codice e di quello previsto per la sintesi della nuova matrice di parità; il primo è dovuto alla ricerca delle codewords, il secondo è causato invece dalla complessità dell'algoritmo di accorciamento.

Tabella 4.1.: Confronto tra accorciamento casuale e accorciamento mirato all'eliminazione delle parole di codice.

È evidente quindi che un accorciamento random privilegia semplicità e velocità di elaborazione, mentre un accorciamento mirato privilegia il guadagno prestazionale; come ci si poteva aspettare, condizione necessaria all'incremento delle prestazioni è quindi un maggiore impiego di risorse computazionali. Sulla base di ciò, si può affermare dunque che i valori di BER in funzione di E_b/N_0 conseguiti attraverso l'accorciamento selettivo rispecchiano maggiormente le finalità del progetto, pur dovendo accettare un tempo di calcolo dei risultati maggiore.

Appendice A.

Distanza minima

Polinomio di parità	d_min(n1)	d_max(n1)	Delta(n1)	delta(n1)	d_min(n2)	d_max(n2)	Delta(n2)	delta(n2)
[0, 1, 3]	1	3	1	0.5	1	3	1	-0.5
[0, 1, 4]	2	5	1.5	0 descrescente	3	6	1.5	0 descrescente
[0, 2, 5]	5	10	2.5	0 descrescente	6	11	2.5	0 descrescente
[0, 1, 2, 4, 5]	5	11	3	0.5	5	11	3	-0.5
[0, 2, 3, 4, 5]	5	10	2.5	0 decrescente	6	11	2.5	0 decrescente
[0, 1, 6]	12	19	3.5	0 decrescente	13	20	3.5	0 decrescente
[0, 1, 2, 5, 6]	12	20	4	0.5	12	20	4	-0.5
[0, 2, 3, 5, 6]	12	19	3.5	0 crescente	13	20	3.5	0 crescente
[0, 1, 7]	27	37	5	0.5	27	37	5	-0.5
[0, 3, 7]	27	37	5	0.5	27	37	5	-0.5
[0, 1, 2, 3, 7]	27	37	5	0.5	27	37	5	-0.5
[0, 2, 3, 4, 7]	27	36	4.5	0 decrescente	28	37	4.5	0 decrescente
[0, 1, 2, 3, 4, 5, 7]	27	37	5	0.5	27	37	5	-0.5
[0, 1, 3, 6, 7]	27	37	5	0.5	27	37	5	-0.5
[0, 2, 4, 6, 7]	27	37	5	0.5	27	37	5	-0.5
[0, 2, 5, 6, 7]	26	38	6	0.5	26	38	6	-0.5
[0, 1, 2, 4, 5, 6, 7]	26	38	6	0.5	26	38	6	-0.5
[0, 2, 3, 4, 8]	56	71	7.5	0 decrescente	57	72	7.5	0 decrescente
[0, 1, 3, 5, 8]	56	72	8	0.5	56	72	8	-0.5
[0, 1, 2, 3, 4, 6, 8]	56	71	7.5	0 decrescente	57	72	7.5	0 decrescente
[0, 1, 5, 6, 8]	56	71	7.5	0 crescente	57	72	7.5	0 crescente
[0, 2, 5, 6, 8]	56	71	7.5	0 crescente	57	72	7.5	0 crescente
[0, 3, 5, 6, 8]	54	73	9.5	0 decrescente	55	74	9.5	0 decrescente
[0, 1, 6, 7, 8]	56	72	8	0.5	56	72	8	-0.5
[0, 1, 2, 5, 6, 7, 8]	56	72	8	0.5	56	72	8	-0.5
[0, 4, 9]	116	140	12	0.5	116	140	12	-0.5
[0, 2, 3, 5, 9]	118	138	10	0.5	118	138	10	-0.5
[0, 3, 4, 6, 9]	117	139	11	0.5	117	139	11	-0.5
[0, 1, 2, 3, 5, 6, 9]	117	139	11	0.5	117	139	11	-0.5
[0, 1, 2, 4, 5, 6, 9]	117	139	11	0.5	117	139	11	-0.5
[0, 1, 3, 4, 6, 7, 9]	115	141	13	0.5	115	141	13	-0.5
[0, 1, 4, 8, 9]	116	139	11.5	0 crescente	117	140	11.5	0 crescente
[0, 4, 5, 8, 9]	118	138	10	0.5	118	138	10	-0.5
[0, 5, 6, 8, 9]	118	138	10	0.5	118	138	10	-0.5
[0, 1, 3, 5, 6, 8, 9]	117	139	11	0.5	117	139	11	-0.5
[0, 2, 7, 8, 9]	117	138	10.5	0 decrescente	118	139	10.5	0 decrescente
[0, 1, 2, 3, 7, 8, 9]	114	141	13.5	0 decrescente	115	142	13.5	0 decrescente
[0, 1, 5, 6, 7, 8, 9]	117	139	11	0.5	117	139	11	-0.5
[0, 3, 5, 6, 7, 8, 9]	117	139	11	0.5	117	139	11	-0.5
[0, 3, 10]	239	272	16.5	0 crescente	240	273	16.5	0 crescente
[0, 1, 3, 4, 10]	240	271	15.5	0 crescente	241	272	15.5	0 crescente
[0, 1, 2, 3, 5, 6, 10]	242	269	13.5	0 decrescente	243	270	13.5	0 decrescente
[0, 2, 3, 8, 10]	240	271	15.5	0 decrescente	241	272	15.5	0 decrescente
[0, 3, 4, 8, 10]	240	272	16	0.5	240	272	16	-0.5
[0, 1, 5, 8, 10]	241	270	14.5	0 decrescente	242	271	14.5	0 decrescente
[0, 4, 5, 8, 10]	240	271	15.5	0 decrescente	241	272	15.5	0 decrescente
[0, 1, 3, 4, 5, 6, 7, 8, 10]	239	272	16.5	0 crescente	240	273	16.5	0 crescente
[0, 1, 4, 9, 10]	241	271	15	0.5	241	271	15	-0.5
[0, 1, 2, 3, 4, 5, 6, 9, 10]	242	270	14	0.5	242	270	14	-0.5
[0, 2, 3, 6, 8, 9, 10]	239	272	16.5	0 crescente	240	273	16.5	0 crescente
[0, 1, 5, 6, 8, 9, 10]	240	272	16	0.5	240	272	16	-0.5
[0, 3, 4, 5, 6, 7, 8, 9, 10]	238	274	18	0.5	238	274	18	-0.5
[0, 2, 11]	490	534	22	0.5	490	534	22	-0.5
[0, 1, 3, 5, 11]	487	537	25	0.5	487	537	25	-0.5
[0, 2, 3, 5, 11]	489	535	23	0.5	489	535	23	-0.5
[0, 1, 5, 6, 11]	491	532	20.5	0 decrescente	492	533	20.5	0 decrescente
[0, 2, 3, 7, 11]	492	532	20	0.5	492	532	20	-0.5
[0, 2, 5, 8, 11]	489	534	22.5	0 decrescente	490	535	22.5	0 decrescente
[0, 1, 4, 5, 6, 8, 11]	484	539	27.5	0 crescente	485	540	27.5	0 crescente
[0, 1, 2, 3, 4, 5, 6, 8, 11]	489	534	22.5	0 decrescente	490	535	22.5	0 decrescente
[0, 1, 4, 9, 11]	490	534	22	0.5	490	534	22	-0.5
[0, 1, 4, 7, 8, 9, 11]	489	535	23	0.5	489	535	23	-0.5
[0, 2, 3, 10, 11]	491	533	21	0.5	491	533	21	-0.5
[0, 1, 3, 4, 7, 10, 11]	491	533	21	0.5	491	533	21	-0.5
[0, 1, 3, 4, 5, 7, 8, 10, 11]	487	537	25	0.5	487	537	25	-0.5
[0, 2, 3, 9, 12]	994	1054	30	0.5	994	1054	30	-0.5
[0, 2, 4, 5, 6, 8, 9, 10, 12]	993	1055	31	0.5	993	1055	31	-0.5
[0, 1, 2, 4, 6, 11, 12]	990	1058	34	0.5	990	1058	34	-0.5
[0, 1, 3, 5, 9, 11, 12]	989	1059	35	0.5	989	1059	35	-0.5
[0, 1, 3, 4, 6, 8, 10, 11, 12]	995	1053	29	0.5	995	1053	29	-0.5
[0, 1, 3, 4, 13]	1994	2102	54	0.5	1994	2102	54	-0.5
[0, 1, 2, 3, 4, 5, 7, 9, 13]	2009	2087	39	0.5	2009	2087	39	-0.5
[0, 1, 5, 7, 8, 9, 13]	2006	2090	42	0.5	2006	2090	42	-0.5
[0, 4, 5, 7, 8, 10, 13]	2002	2094	46	0.5	2002	2094	46	-0.5
[0, 1, 4, 7, 8, 11, 13]	2002	2094	46	0.5	2002	2094	46	-0.5
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13]	1993	2103	55	0.5	1993	2103	55	-0.5
[0, 2, 3, 4, 6, 8, 10, 12, 13]	2005	2091	43	0.5	2005	2091	43	-0.5
[0, 1, 4, 6, 7, 8, 11, 12, 13]	2010	2086	38	0.5	2010	2086	38	-0.5
[0, 1, 3, 4, 6, 7, 9, 10, 14]	4028	4164	68	0.5	4028	4164	68	-0.5
[0, 1, 6, 11, 14]	4012	4180	84	0.5	4012	4180	84	-0.5
[0, 2, 5, 6, 9, 11, 14]	4018	4174	78	0.5	4018	4174	78	-0.5
[0, 1, 3, 5, 6, 7, 8, 9, 11, 12, 14]	4032	4160	64	0.5	4032	4160	64	-0.5
[0, 1, 3, 5, 6, 13, 14]	4023	4169	73	0.5	4023	4169	73	-0.5
[0, 1, 6, 7, 10, 11, 12, 13, 14]	4031	4161	65	0.5	4031	4161	65	-0.5

Figura A.1.

Appendice B.

Programmi Matlab

B.1. Codice per la sintesi della matrice di parità iniziale

```
1
2  function H=poly_scorre(poly,n)
3
4  %1) poly=zeros(1,k+1)
5  %2) poly([Golomb primitivo]+1)=1
6
7  primariga=[poly zeros(1,n-length(poly))];
8  posizione_elementi=1:length(primariga);
9  posizione=posizione_elementi(primariga==1);
10 H=primariga;
11 while H(:,end)~=1
12     primariga=zeros(1,length(primariga));
13     posizione=posizione + ones(1,length(posizione));
14     primariga(posizione)=1;
15     H=[H;primariga];
16 end
```

Listing B.1: In ingresso vengono forniti il parametro poly (equivalente al vettore **h**) e il numero delle colonne n prima dell'accorciamento.

B.2. Codice per la simulazione di un accorciamento casuale multiplo

```
1
2  function H_accorc = accorciamento_r(golomb, n_tolte,
       R_des)
3
4  k = max(golomb);
5  k_des = k - n_tolte;
```

```
n_des = ceil(k_des / R_des);
polinomio = zeros(1, n_des + n_tolte);
polinomio(golomb + 1) = 1;
H = poly_scorre(polinomio, n_des + n_tolte);
H_accorc=H;

flag=1;
while flag
    n_pos_selez=randperm(size(H,2),n_tolte);
    H_accorc(:,n_pos_selez)=[];

    if sum(sum(H_accorc,2)>0)==size(H_accorc,1) &&
       sum(sum(H_accorc,1)>0)==size(H_accorc,2)
        flag=0;
    end
end
```

Listing B.2: Vengono presi come parametri il golomb primitivo, il numero di colonne da togliere desiderato n_tolte e il rate R_des che deve assumere il codice dopo l'accorciamento. In primis vengono calcolati: il nuovo numero di cifre di informazione k_des e quello delle colonne della matrice; in seguito viene generato un vettore lungo n_tolte che ha come elementi quelli presi casualmente nell'intervallo $[1, n]$. Successivamente vengono soppresse n_tolte colonne in base alle posizioni date dagli elementi del vettore costruito precedentemente. Infine viene effettuato un controllo iterativo del vincolo sulle righe e colonne non nulle.

B.3. Codice per la simulazione di un accorciamento selettivo multiplo

```

function Matrice_parole = Parole_codice(n)

file = fopen('nome del file.txt','r');
dati = [];
flag=1;
while ~feof(file) && flag
    riga_file = fgetl(file);
    if ~contains(riga_file,"W")
        st_to_nu=str2num(riga_file);
        numeri =[st_to_nu 0.5*ones(1,n-length(st_to_nu))];
```

```
12          dati=[dati;numeri];
13      end
14      if contains(riga_file,"Weight=%...%")
15              flag=0;
16      end
17  end
18
19  fclose(file);
20
21  N_parole=size(dati,1);
22
23  Matrice_zeri=zeros(N_parole,n);
24  Matrice_uni=ones(N_parole,n);
25  Matrice_exp_parole=dati;
26  Matriceexpparole=Matrice_exp_parole + Matrice_uni;
27  for j=1:N_parole
28      for i_matriceexpparole=1:n
29          for i_matricezeri=1:n
30               if Matriceexpparole(j,i_matriceexpparole)
                    == i_matricezeri
31                  Matrice_zeri(j,i_matricezeri) = 1;
32              end
33          end
34      end
35  end
36  Matrice_parole=Matrice_zeri;
37  for k=1:N_parole
38      if dati(k,1)~=0
39          Matrice_parole(k,1)=0;
40      end
41  end
```

Listing B.3: Si costruisce una matrice binaria di parole di codice a partire dai supporti forniti dallo strumento in [13] e presenti nel file .cw (ognuno di questi file corrisponde a un codice con un rate specifico).

Come suggeriscono gli step 3 e 4, l'idea è quella di verificare la presenza di zeri all'esterno di ogni parola trovata (che comincia e termina con un 1) e in caso affermativo considerare tutti i suoi possibili shift non ciclici, verificando che le parole risultanti siano a loro volta parole di codice attraverso il prodotto scalare $\mathbf{v} \cdot \mathbf{H}^t = 0$. In questo modo, ci si aspetta di trovare altre parole di codice oltre a quelle trovate dal programma.

```
1
2  function Matrice_circshift =
       verifica_circshift_parole(golomb,n,peso_scelto)
3
4   Matrice_parole_scelta =
       Parole_peso_scelto(peso_scelto,n);
5
6   polinomio=zeros(1,n);
7   polinomio(golomb+1)=1;
8   H=poly_scorre(polinomio,n);
9   posizione_elem_Matpsc=1:size(Matrice_parole_scelta,2);
10  Matrice_circshift_lessctr=[];
11 for i_righe=1:size(Matrice_parole_scelta,1)
12
13     parola= Matrice_parole_scelta(i_righe,:);
14     posizione_uni=posizione_elem_Matpsc(parola==1);
15     pos_min=min(posizione_uni);
16     pos_max=max(posizione_uni);
17     num_zerisx=sum(parola(posizione_elem_Matpsc(1:
18     (pos_min-1)))==0);
19
20   num_zeridx=sum(parola(posizione_elem_Matpsc((pos_max+1):
21   length(parola)))==0);
22
23         if num_zerisx>0 || num_zeridx>0
24            Matrice_circshiftdx=parola;
25            riga_dx=1;
26            while Matrice_circshiftdx(riga_dx,end)~=1
27                parola=zeros(1,length(parola));
28                posizione_uni=posizione_uni+ones(1,
29                length(posizione_uni));
30
31                parola(posizione_uni)=1;
32                Matrice_circshiftdx=[Matrice_circshiftdx;
33                parola];
34
35                riga_dx=riga_dx+1;
36            end
37            parola_sx=Matrice_parole_scelta(i_righe,:);
38            posizione_unisx=posizione_elem_Matp
```

```
39            sc(parola_sx==1);
40
41            Matrice_circshiftsx=parola_sx;
42            riga_sx=1;
43            while Matrice_circshiftsx(riga_sx,1)~=1
44                parola_sx=zeros(1,length(parola));
45                posizione_unisx=posizione_unisx-ones(1,
46                length(posizione_uni));
47
48                parola_sx(posizione_unisx)=1;
49                Matrice_circshiftsx=[Matrice_circshiftsx;
50                parola_sx];
51
52                riga_sx=riga_sx+1;
53            end
54          Matrice_circshift_dxsx=[Matrice_circshiftdx;
55          Matrice_circshiftsx];
56
57          Matrice_circshift_lessctr=[Matrice_circshift_les
58          sctr; Matrice_circshift_dxsx];
59
60        end
61          if num_zerisx==0 && num_zeridx==0
62            Matrice_circshift_lessctr=[Matrice_circshift_les
63            sctr; parola];
64
65          end
66 end
67
68 Matrice_circshift_lessctr=unique(Matrice_circshift_lessctr
69 ,'rows');
70
71 for i_circsh=size(Matrice_circshift_lessctr,1):-1:1
72     if
          any(mod(Matrice_circshift_lessctr(i_circsh,:)*H',2))
73          Matrice_circshift_lessctr(i_circsh,:)=[];
74     end
75 end
76 Matrice_circshift=Matrice_circshift_lessctr;
```

Listing B.4: Per ogni parola considerata si verifica che al di fuori di essa ci siano degli zeri, se la condizione è soddisfatta ne opera lo shift non ciclico destro e sinistro di una posizione alla volta fino a che gli uni più esterni non coprono sia la posizione 0 che la posizione $n-1$.

dove Matrice_parole_scelta è una matrice ottenuta dalla funzione B.5.

```
1
2 function Matrice_parole_scelta =
     Parole_peso_scelto(peso_scelto,n)
3
4   Matrice_parole = Parole_codice(n);
5   peso=sum(Matrice_parole,2);
6   Matrice_parole_scelta=
7   Matrice_parole(peso==peso_scelto,:);
```

Listing B.5: Viene generata una matrice per un gruppo di parole con un peso specificato.

Infine, la funzione che svolge l'accorciamento di **H** è quella descritta di seguito (in cui viene chiamata la B.4):

```
1
2 function H_accorc_nr =
     accorciamento_nr(golomb,n_tolte,R_des)
3
4   k=max(golomb);
5   k_des=k-n_tolte;
6   n_des=ceil(k_des/R_des);
7   n=n_des+n_tolte;
8
9   polinomio=zeros(1,n);
10  polinomio(golomb+1)=1;
11  H=poly_scorre(polinomio,n);
12  H_accorc_nr=H;
13
14  Matrice_parole_MINDIST = Parole_codice(n);
15  peso_minimo=min(unique(sum(Matrice_parole_MINDIST,2)));
16
17  supporto_min = unique(supp_func(golomb,n,peso_minimo));
18  flag_0=1;
```

```
19   s=1;
20 while flag_0
21     supporto_2 =
          unique(supp_func(golomb,n,peso_minimo+s));
22     if ~isempty(supporto_2)
23         flag_0=0;
24     end
25     s=s+1;
26 end
27
28   Matrice_circshift=
        verifica_circshift_parole(golomb,n,peso_minimo) ;
29
30   posizione_elem_Matcirc=1:size(Matrice_circshift,2);
31   Matrice_interi=[];
32  for i_supp=1:size(Matrice_circshift,1)
33      riga_interi=posizione_elem_Matcirc(Matrice_circshi
34      ft(i_supp,:)==1);
35
36      Matrice_interi=[Matrice_interi;riga_interi];
37  end
38
39  unelem_perriga_vec=[];
40
41 if n_tolte>=length(supporto_min)
42     colonne_aggiuntive=n_tolte-length(supporto_min);
43
44     flag=1;
45         while flag
46             pos_uni_casuale2=randperm(length(supporto_2),
                   colonne_aggiuntive);
47             peso_aggiuntivo=supporto_2(pos_uni_casuale2);
48             if length([supporto_min peso_aggiuntivo])==
49             length(unique([supporto_min
                   peso_aggiuntivo]))
50
51                 flag=0;
52             end
53         end
54
55     H_accorc_nr(:,[supporto_min peso_aggiuntivo])=[];
56
```

```
else
    flag=1;
    while flag
       for j=1:size(Matrice_interi,1)
        random_pos= randperm(peso_minimo,1);
        dati_random=Matrice_interi(j,random_pos);
        unelem_perriga_vec=[unelem_perriga_vec;dati_ran
        dom];
        if length(unelem_perriga_vec)==
        length(unique(unelem_perriga_vec))

            flag=0;
        end
       end
    end

    colonne_aggiunte=n_tolte-length(unelem_perriga_vec);
        if colonne_aggiunte>0

            flag_2=1;
            while flag_2
                pos_uni_cas2=randperm(length(supporto_2),
                   colonne_aggiunte);
                peso_aggiunto=supporto_2(pos_uni_cas2);
                if length([unelem_perriga_vec'
                   peso_aggiunto])== length
                   (unique([unelem_perriga_vec'
                   peso_aggiunto]))

                    flag_2=0;
                end
            end

            H_accorc_nr(:,[unelem_perriga_vec'
               peso_aggiunto] )=[];
        else
           supporto_mintolto=supporto_min(1:n_tolte);
           H_accorc_nr(:,supporto_mintolto)=[];
        end

 end
```

Listing B.6: Si effettua l'accorciamento selettivo in base al parametro n_tolte (corrispondente a s).

dove supporto è ottenuto mediante la B.7.

```
1
2 function supporto=supp_func(golomb,n,peso_scelto)
3   Matrice_circshift=
       verifica_circshift_parole(golomb,n,peso_scelto) ;
4   posizione_elem_Matcirc=1:size(Matrice_circshift,2);
5   Matrice_interi=[];
6  for i_supp=1:size(Matrice_circshift,1)
7       riga_interi=posizione_elem_Matcirc(Matrice_circshi
8       ft(i_supp,:)==1);
9
10      Matrice_interi=[Matrice_interi;riga_interi];
11  end
12   supporto = unique(reshape(Matrice_interi',1,[]));
```

Listing B.7: definisce il supporto di una codeword di un certo peso.

Se ora si da come ingresso al programma in [13] la matrice accorciata ottenuta dalla funzione B.6, si ha come risultato la decimazione di tutte le parole di codice di peso più basso.

Bibliografia

[1] John G. Proakis and Masoud Salehi. Communication System Engineering. Prentice-Hall (Pearson), 2nd Ed, 2001.

[2] C. E. Shannon, "A mathematical theory of communication," in The Bell System Technical Journal, vol. 27, no. 3, pp. 379-423, July 1948.

[3] R. W. Hamming, "Error detecting and error correcting codes," in The Bell System Technical Journal, vol. 29, no. 2, pp. 147-160, April 1950.

[4] R. Gallager, "Low-density parity-check codes," in IRE Transactions on Information Theory, vol. 8, no. 1, pp. 21-28, January 1962.

[5] R. Tanner, "A recursive approach to low complexity codes," in IEEE Transactions on Information Theory, vol. 27, no. 5, pp. 533-547, September 1981.

[6] A. Jimenez Felstrom and K. S. Zigangirov, "Time-varying periodic convolutional codes with low-density parity-check matrix," in IEEE Transactions on Information Theory, vol. 45, no. 6, pp. 2181-2191, Sept. 1999, doi: 10.1109/18.782171. keywords: Convolutional codes,

[7] R. Jose and A. Pe, "Analysis of hard decision and soft decision decoding algorithms of LDPC codes in AWGN," 2015 IEEE International Advance Computing Conference (IACC), Banglore, India, 2015, pp. 430-435, doi: 10.1109/IADCC.2015.7154744. keywords: Decoding;Iterative decoding;Block codes;Bit error rate;Sparse matrices;Forward error correction;LDPC;FEC;LBC;iterative decoding;BER;Sum product decoding;Message passing algorithm,

[8] Xiao-Yu Hu, E. Eleftheriou, D.-M. Arnold and A. Dholakia, "Efficient implementations of the sum-product algorithm for decoding LDPC codes," GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No.01CH37270), San Antonio, TX, USA, 2001, pp. 1036-1036E vol.2

[9] M. Battaglioni, M. Baldi, F. Chiaraluce and G. Cancellieri, "Rate-compatible LDPC Codes based on Primitive Polynomials and Golomb Rulers," in IEEE Transactions on Communications.

[10] M. Battaglioni and G. Cancellieri, "Punctured Binary Simplex Codes as LDPC codes," 2022 61st FITCE International Congress Future Telecommunications: Infrastructure and Sustainability (FITCE), Rome, Italy, 2022, pp. 1-6.

[11] M. Battaglioni, M. Amagliani, M. Baldi, F. Chiaraluce and G. Cancellieri, "Design and Analysis of a Family of Complexity-Constrained LDPC Codes," 2024 IEEE International Symposium on Information Theory (ISIT), Athens, Greece, 2024, pp. 428-433.

[12] M. Battaglioni, G. Greganti and G. Cancellieri, "Optimized Rate-Adaptive Error Correction through Puncturing and Shortening of Simplex Codes," AEIT 2024, pp. 1-6.

[13] D. J. C. MacKay. (2008) Source code for approximating the MinDist problem of LDPC codes. [Online]. Available: http://www.inference.eng.cam.ac.uk/mackay/MINDIST_ECC.html