

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Elettronica e delle Tecnologie Digitali

---

Studio e Analisi dell'Algoritmo di decodifica Guessing Random Additive Noise Decoding (GRAND)

Candidato:
Michele Di Chiara

Relatore:
Dr. Massimo Battaglioni

Anno Accademico 2025-2026



UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Elettronica e delle Tecnologie Digitali

---

Studio e Analisi dell'Algoritmo di decodifica Guessing Random Additive Noise Decoding (GRAND)

Candidato:
Michele Di Chiara

Relatore:
Dr. Massimo Battaglioni

Anno Accademico 2025-2026

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Elettronica e delle Tecnologie Digitali
Via Brecce Bianche – 60131 Ancona (AN), Italy

Ringraziamenti

Desidero innanzitutto ringraziare il Dott. Massimo Battaglioni per la grande disponibilità dimostrata durante tutto lo svolgimento di questo lavoro di tesi a partire dal tirocinio, per il confronto costante e per l'attenzione con cui mi ha seguito nelle diverse fasi del lavoro e non solo. Le sue indicazioni sono state fondamentali per affrontare con maggiore consapevolezza gli aspetti teorici e implementativi trattati e per portare a termine questo percorso.

Un ringraziamento particolare va inoltre al Prof. Franco Chiaraluce, per quanto mi ha trasmesso attraverso gli insegnamenti di Segnali Determinati e Aleatori e Telecomunicazioni Digitali. Le sue lezioni hanno contribuito in modo importante a far crescere il mio interesse per il mondo delle telecomunicazioni, trasformando progressivamente quello che inizialmente era soltanto un interesse, in una vera passione.

Desidero poi ringraziare l'Università Politecnica delle Marche e tutti i docenti che hanno contribuito alla mia formazione durante questi tre anni. Quando ho iniziato questo percorso non avevo ancora ben chiaro quale strada avrei voluto seguire e non ero nemmeno certo che l'ingegneria sarebbe stata la scelta giusta per me. Nel corso della laurea triennale ho avuto invece la possibilità non soltanto di costruire le basi della mia formazione ingegneristica, ma anche di imparare ad apprezzare sempre di più lo studio dell'ingegneria e il modo di affrontare i problemi che essa richiede.

Con il passare del tempo ho potuto comprendere quali fossero gli ambiti più vicini ai miei interessi e, tra questi, le comunicazioni hanno rappresentato senza dubbio quello che più mi ha coinvolto. Concludo, quindi, questo percorso con la consapevolezza di aver trovato, durante questi anni, non soltanto una formazione, ma anche una direzione verso cui desidero continuare a crescere e approfondire i miei studi.

Ancona, Ottobre 2026

Michele Di Chiara

Indice

Introduzione **1**

1 GRAND **5**
1.1 Richiami sui codici lineari 5
1.2 GRAND con decisione hard e canale BSC 6
1.3 Ordinamento dei pattern mediante peso di Hamming 7
1.4 Test di appartenenza tramite sindrome 7
1.5 Esempio su codice toy 7
1.6 Generalizzazione della generazione dei pattern 8
1.7 Struttura dell'implementazione in C 11
1.8 Codice LDPC utilizzato 13
1.9 Collegamento con la valutazione numerica 14

2 ORBGRAND **15**
2.1 Dal GRAND hard all'informazione soft 15
2.2 Modulazione BPSK e canale AWGN 15
2.3 Generazione del rumore mediante Box-Muller 16
2.4 Decisione hard, LLR e affidabilità 17
2.5 Permutazione di affidabilità e distinzione tra rango e posizione fisica 18
2.6 Logistic Weight 19
2.7 Generazione dei pattern secondo Logistic Weight 20
2.8 Dal dominio dei ranghi alle posizioni fisiche 22
2.9 Esempio numerico su codice toy 23
2.10 Struttura dell'implementazione in C 24
2.11 Applicazione al codice LDPC 26

3 CDF-ORBGRAND **27**
3.1 Rank Companding 28
3.2 Peso CDF dei pattern 29
3.3 CDF del modulo dell'LLR nel canale AWGN 29
3.4 Significato del risultato teorico 30
3.5 Esempio di verifica dell'ordinamento CDF 30
3.6 Il ruolo della min-heap 32
3.7 Rappresentazione compatta dei pattern per N=128 33
3.8 Struttura del decoder CDF-ORBGRAND 34
3.9 Validazione dell'ordine a $N = 128$ 37

3.10 Collegamento con il confronto numerico 38

4 Risultati numerici **39**

4.1 Impostazione delle simulazioni 39
4.2 Risultati di GRAND hard sul canale BSC 40
4.3 Confronto ORBGRAND e CDF-ORBGRAND 43
4.3.1 Confronto del BLER 43
4.3.2 Confronto del numero medio di query 44
4.3.3 Analisi sui medesimi blocchi 45
4.4 Sintesi dei risultati 45

Conclusioni **47**

Elenco delle figure

1.1 Flowchart della generazione ricorsiva dei pattern di errore mediante backtracking. 10

2.1 Flowchart del procedimento di decodifica ORBGRAND. 25

3.1 Flowchart del procedimento di decodifica CDF-ORBGRAND. 35

4.1 Confronto tra il BLER simulato di GRAND hard e i riferimenti teorici sul canale BSC: probabilità di abbandono P_{ab}, bound RCU, converse e approssimazione normale, al variare della probabilità di crossover p. 40

4.2 Numero medio di query per blocco di GRAND hard al variare della probabilità di crossover p. 42

4.3 Confronto del BLER tra ORBGRAND e CDF-ORBGRAND al variare di E_b/N_0. 43

4.4 Confronto del numero medio di query per blocco tra ORBGRAND e CDF-ORBGRAND al variare di E_b/N_0. 44

Elenco delle tabelle

4.1 Numero di blocchi recuperati esclusivamente da uno dei due decoder. 45

Introduzione

Nell'ambito delle comunicazioni digitali, uno dei problemi fondamentali è rappresentato dall'affidabilità dell'informazione. A partire dai lavori svolti da Claude Shannon, sono state poste le basi dell'"Information Theory", mostrando come tale informazione può essere trasmessa con ottimi gradi di affidabilità, in presenza di disturbi, quali rumore [1]. Nella pratica, la codifica di canale è l'operazione mediante la quale una trasmissione digitale viene protetta dai disturbi del canale attraverso l'introduzione di ridondanza nel messaggio trasmesso. In ricezione, il decoder sfrutta tale ridondanza per individuare e, quando possibile, correggere gli errori introdotti dal canale, ricostruendo l'informazione trasmessa nella maniera più affidabile possibile [2].

I decoder tradizionali vengono generalmente progettati per specifiche famiglie di codici, sfruttando la stessa struttura del codice considerato. Ovviamente, questa impostazione può portare a soluzioni con elevatissima efficienza ma, d'altra parte, rende il procedimento di decodifica strettamente vincolato alle caratteristiche del codice. Il paradigma *Guessing Random Additive Noise Decoding* (GRAND) propone, invece, un diverso punto di vista. Infatti, invece di cercare direttamente la codeword trasmessa, il decoder tenta di ricostruire il rumore che ha alterato la parola ricevuta [3]. In quanto processo stocastico, le diverse possibili realizzazioni del rumore vengono considerate in ordine di probabilità, dalla più probabile alla meno probabile, e per ogni ipotesi viene verificato se la sequenza candidata appartiene o meno al codebook. Dunque, con un ordinamento coerente con la distribuzione di probabilità del rumore, GRAND realizza una decodifica di massima verosimiglianza, mantenendo un funzionamento indipendente dalla struttura del codice.

Pertanto, GRAND risulta molto interessante dal punto di vista della sua natura e soprattutto dal punto di vista implementativo. Nel caso hard-decision considerato nel presente lavoro, il canale viene modellato come *Binary Symmetric Channel* e i pattern di errore vengono ordinati secondo peso di Hamming crescente. Infatti, per una probabilità di crossover $p < 0.5$, la probabilità di una specifica realizzazione del vettore di errore dipende dal suo peso di Hamming e diminuisce all'aumentare di quest'ultimo. Per un codice lineare, ogni parola candidata può essere verificata attraverso il controllo della relativa sindrome [3].

L'informazione hard, tuttavia, non conserva il grado di affidabilità delle singole decisioni. Pertanto, per sfruttare l'informazione soft disponibile in uscita da un canale, è stato proposto *Ordered Reliability Bits GRAND* (ORBGRAND) [4]. In questo caso, le posizioni della sequenza ricevuta vengono invece ordinate in base

alla loro affidabilità, assegnando a ognuna di esse una posizione in classifica, a cui viene associato un rango. I pattern di rumore vengono, dunque, esplorati guidati da un'indicazione basata sui ranghi delle posizioni coinvolte, ovvero il *Logistic Weight*, dando maggiore importanza alle ipotesi di errore che interessano i bit meno affidabili. ORBGRAND, in questo modo, mantiene il principio di universalità, utilizzando, però, l'informazione soft, per modificare l'ordine con cui vengono testate le ipotesi di rumore [4].

Tuttavia, l'ordinamento basato sui ranghi di affidabilità pone dei limiti; in particolare, si può osservare che esso conserva l'ordine relativo di affidabilità, ma non la distanza effettiva tra i valori. Dunque, la formulazione *CDF-ORBGRAND* affronta tale aspetto, attraverso una trasformazione dei ranghi normalizzati, basata sull'inversa della funzione di ripartizione di affidabilità, secondo il principio del *rank companding* [5]. La nuova metrica assegna, quindi, ai pattern un costo che tiene conto della distribuzione statistica dell'affidabilità e non soltanto della posizione nell'ordinamento. A partire da questa formulazione, dunque, seguirà l'analisi e l'implementazione di CDF-ORBGRAND.

Il contributo di questa tesi è principalmente algoritmico e implementativo. Tale lavoro è partito dallo studio e dall'implementazione in linguaggio C di GRAND hard-decision, con particolare attenzione alla generazione dei pattern di rumore, al controllo della sindrome e alla gestione finale del numero massimo di query di lavoro. L'implementazione è stata successivamente estesa a ORBGRAND, con l'introduzione dell'ordinamento delle affidabilità e la ricerca dei pattern a partire dal valore del *Logistic Weight*. Infine, il lavoro ha dato importanza all'implementazione di CDF-ORBGRAND, dove il costo associato ai pattern viene calcolato mediante la trasformazione CDF, gestendo poi l'ordine di esplorazione attraverso una struttura *min-heap*.

Le implementazioni realizzate sono state applicate a un codice LDPC regolare, la cui matrice di controllo di parità è stata costruita secondo il metodo introdotto da Gallager [6], con lunghezza $N = 128$, dimensione $K = 67$, peso di colonna pari a 4 e peso di riga pari a 8. La stessa struttura di codice è stata impiegata per le diverse versioni del decoder, così da confrontare le strategie di guessing mantenendo invariato il codice considerato.

La valutazione numerica è stata organizzata in due diversi scenari. GRAND hard è stato simulato su BSC, poiché in tale modello l'informazione ricevuta è binaria e l'ordine delle ipotesi di errore può essere determinato direttamente dal peso di Hamming. ORBGRAND e CDF-ORBGRAND sono stati invece confrontati su canale AWGN con modulazione BPSK, poiché i campioni ricevuti conservano un'informazione soft che permette di associare ai bit un diverso grado di affidabilità. Nei due casi sono stati quindi fatti variare, rispettivamente, la probabilità di crossover p e il rapporto E_b/N_0 [2]. Oltre alla metrica del *Block Error Rate*, che quantifica la performance in termini di tasso d'errore, è stato registrato il numero di query (ovvero di pattern d'errore testati). Così facendo, si affianca all'analisi prestazionale

un'indicazione degli oneri computazionali sostenuti dai vari decoder.

La tesi è organizzata come segue. Il primo capitolo è dedicato a GRAND e ne presenta il principio di funzionamento, così come l'ordinamento dei pattern nel caso di hard-decision e le varie scelte implementative. Il secondo capitolo introduce ORBGRAND e ne descrive l'utilizzo in relazione all'affidabilità dell'informazione e al *Logistic Weight*. Il terzo capitolo è dedicato invece a CDF-ORBGRAND, alla trasformazione dei ranghi e alla relativa implementazione. Il quarto capitolo raccoglie i risultati numerici di tutti i decoder. Le conclusioni riassumono, infine, i risultati ottenuti e le osservazioni emerse dai confronti.

Capitolo 1

GRAND

Il *Guessing Random Additive Noise Decoding* (GRAND) affronta il problema della decodifica a partire da un punto di vista differente rispetto ai tradizionali decoder, costruiti attorno alla struttura delle famiglie di codice su cui operano. Alla base di GRAND, vi è l'idea di formulare ipotesi sul rumore che può aver modificato la parola di codice trasmessa, ordinando tali ipotesi in ordine di probabilità. Per ogni realizzazione ipotizzata, il decoder costruisce un'eventuale parola trasmessa, a partire dalla quale verifica la sua appartenenza al codebook. In questo modo, GRAND viene rappresentato come un algoritmo *noise-centric*, tale per cui il codice viene utilizzato, principalmente, come test di appartenenza, mentre il fulcro della ricerca giunge ad essere l'ordine con cui vengono esplorate le possibili realizzazioni del rumore [3].

Così, questa impostazione, nel suo funzionamento generale, rende GRAND indipendente da una qualsiasi specifica costruzione del codice. Inoltre, per un codice lineare, è sufficiente disporre di un unico metodo per poter stabilire se per una parola candidata vengono soddisfatti i vincoli di parità, ossia il controllo della sindrome. Data questa circostanza, è possibile mantenere separati il meccanismo di generazione delle ipotesi di rumore e la verifica della validità della codeword.

1.1 Richiami sui codici lineari

Un codice lineare binario $\mathcal{C}$ di lunghezza N e dimensione K, indicato come codice (N, K), è un sottospazio vettoriale di $\mathrm{GF}(2)^N$ avente dimensione K. Di conseguenza, il codice contiene 2^K parole di codice, ciascuna costituita da N simboli binari.

Una matrice generatrice $G \in \mathrm{GF}(2)^{K \times N}$ è una matrice di rango K le cui righe costituiscono una base del codice $\mathcal{C}$. Dunque, indicando con $M \in \mathrm{GF}(2)^K$ il vettore informativo, ogni parola di codice $X \in \mathcal{C}$ può essere ottenuta, operando in $\mathrm{GF}(2)$, come

$$X = MG. \tag{1.1}$$

Una matrice di controllo di parità $H \in \mathrm{GF}(2)^{m \times N}$ è una matrice di rango $N - K$ tale che le parole del codice $\mathcal{C}$ siano esattamente le soluzioni del sistema omogeneo

$$HX^T = 0. \tag{1.2}$$

Inoltre, la matrice generatrice e la matrice di controllo di parità soddisfano la relazione

$$GH^T = 0. \tag{1.3}$$

Tali proprietà costituiscono i richiami fondamentali sui codici lineari utilizzati nel seguito [2].

Per una generica sequenza Z, viene definita sindrome il vettore

$$S = ZH^T. \tag{1.4}$$

Pertanto, Z appartiene al codice se e solo se la relativa sindrome è nulla. Tale proprietà costituisce, dunque, il criterio di appartenenza utilizzato dal decoder GRAND nel seguito.

1.2 GRAND con decisione hard e canale BSC

Nella prima implementazione, l'algoritmo viene analizzato in versione hard-decision. Quindi, il decoder riceve una sequenza binaria qualsiasi, senza disporre di un'informazione relativa al grado di affidabilità dei singoli bit. Come già detto, il canale implementato, su cui transita l'informazione, è il *Binary Symmetric Channel* (BSC), in cui ogni bit può essere invertito indipendentemente, con probabilità di crossover p [2].

Indicando con X la parola trasmessa, con E il pattern binario degli errori del canale (ottenuto come modello delle possibili realizzazioni di rumore) e con Y la sequenza ricevuta, la relazione simbolica tra parola trasmessa e conseguente codeword ricevuta è

$$Y = X \oplus E. \tag{1.5}$$

Poiché in GF(2) somma e sottrazione coincidono, il decoder ipotizza un pattern di errore candidato $\hat{E}$ e costruisce la parola candidata

$$Z = Y \oplus \hat{E}. \tag{1.6}$$

Successivamente, per stabilire se Z appartiene al codebook, al decoder non resta che verificare tramite controllo di sindrome se quest'ultima risulta nulla. Se la verifica ha esito positivo, GRAND può arrestare la ricerca e restituire la candidata trovata.

Tuttavia, è ammissibile che possa verificarsi un caso limite, ovvero il rilevamento di una parola a sindrome nulla, che però non coincide con la sequenza trasmessa.

1.3 Ordinamento dei pattern mediante peso di Hamming

Nel BSC, la probabilità di una specifica realizzazione e del vettore di errore E, caratterizzata da peso di Hamming w, è

$$P(E = e) = p^w(1-p)^{N-w}. \tag{1.7}$$

Dunque, a parità di lunghezza, per $p < 0.5$, un pattern caratterizzato da un numero minore di bit invertiti risulta più probabile di un pattern con peso maggiore. Oltre a ciò, tutti i pattern aventi lo stesso peso di Hamming sono equiprobabili. Ne consegue che, nel caso hard considerato, l'ordine di guessing richiesto da GRAND coincide con l'esplorazione dei pattern secondo peso crescente. Si parte, pertanto, con il pattern nullo, per poi passare a tutti i pattern di peso 1, quindi quelli di peso 2 e così via fino a N [3].

Poiché nel BSC tutti i pattern aventi lo stesso peso di Hamming sono equiprobabili, l'ordine con cui essi vengono esplorati all'interno dello stesso livello di peso è irrilevante ai fini dell'ordinamento probabilistico. Motivo per il quale, dal punto di vista implementativo, la generazione di pattern aventi stesso w risulta completamente arbitraria. Ovviamente, nessun pattern di peso superiore sarà testato prima di aver esaurito tutti quelli appartenenti a pesi inferiori.

1.4 Test di appartenenza tramite sindrome

Il criterio di appartenenza introdotto all'inizio del capitolo viene applicato da GRAND a ogni parola candidata Z. Il decoder calcola la relativa sindrome secondo l'equazione (1.4) e accetta la candidata quando questa risulta nulla.

Tale controllo risulta coerente alla natura di GRAND, perché evita la necessità di confrontare la candidata con un vasto elenco di codeword al fine di valutarne la struttura. Per ogni pattern di errore, dunque, il decoder esegue sempre lo stesso set di istruzioni: a partire dalla costruzione di Z tramite operatore XOR, seguendo poi il calcolo della sindrome, la verifica della sua nullità e, in caso negativo, il passaggio al pattern successivo.

1.5 Esempio su codice toy

Per illustrare il comportamento del decoder, è utile utilizzare un codice di piccola lunghezza. Si consideri un codice $(5, 2)$, definito dalla matrice generatrice

$$G = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{1.8}$$

e dalla matrice di controllo di parità

$$H = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{1.9}$$

Assumendo il messaggio informativo

$$M = [1\ 1], \tag{1.10}$$

la corrispondente parola di codice è

$$X = 11011. \tag{1.11}$$

Se, dunque, il pattern di errore che modella la realizzazione di rumore è

$$E = 01100, \tag{1.12}$$

la sequenza ricevuta risulta

$$Y = X \oplus E = 10111. \tag{1.13}$$

Il decoder non conosce, ovviamente, né X né il pattern realmente introdotto dal canale. Il primo tentativo è, stando a quanto detto,

$$\hat{E}_0 = 00000. \tag{1.14}$$

La candidata coincide naturalmente con Y e viene scartata, essendo la sindrome non nulla. La ricerca prosegue con i pattern di peso 1 e, dal momento in cui tutte le relative sindromi sono non nulle, con quelli di peso 2. Nel momento in cui viene testato

$$\hat{E} = 01100, \tag{1.15}$$

si verifica la nullità della sindrome e la candidata viene, di conseguenza, accettata. Tale esempio mostra come GRAND non identifica il rumore confrontandolo con quello realmente generato dal canale, ma si arresta alla prima ipotesi che conduce a una parola valida del codice.

1.6 Generalizzazione della generazione dei pattern

Per un pattern di errore di lunghezza N, fissato il peso di Hamming w, le possibili configurazioni corrispondono a tutte le scelte di w posizioni tra le N disponibili. Il loro numero è pertanto pari al coefficiente binomiale

$$\binom{N}{w}, \tag{1.16}$$

e la procedura di generazione deve quindi operare in funzione dei parametri N e w, producendo tutte le configurazioni associate al peso considerato senza omissioni e senza duplicazioni.

Per la generazione dei pattern, fissato un peso, è stata adottata una procedura ricorsiva, in termini tecnici *backtracking*. L'idea consiste nello scegliere gradualmente le posizioni in cui collocare il simbolo 1 nei pattern di errore. Dopo aver selezionato una specifica posizione, la ricerca continua per le posizioni successive e soltanto per esse, impedendo di generare la stessa combinazione più volte in ordini differenti. Dal punto di vista implementativo, da programma occorre utilizzare un contatore che segnala il numero di 1 ancora da inserire. Pertanto, quando questo numero è nullo, il pattern risulta completo e può essere sottoposto al test GRAND.

Per $N = 5$ e $w = 2$, ad esempio, la procedura genera dieci pattern, nell'ordine:

$$\begin{aligned} &11000,\ 10100,\ 10010,\ 10001,\ 01100,\\ &01010,\ 01001,\ 00110,\ 00101,\ 00011. \end{aligned} \tag{1.17}$$

Il meccanismo di backtracking permette, dunque, di esplorare tutte le combinazioni mantenendo una struttura indipendente dal valore specifico di N.

La logica della procedura ricorsiva adottata è rappresentata nel flowchart in Figura 1.1.

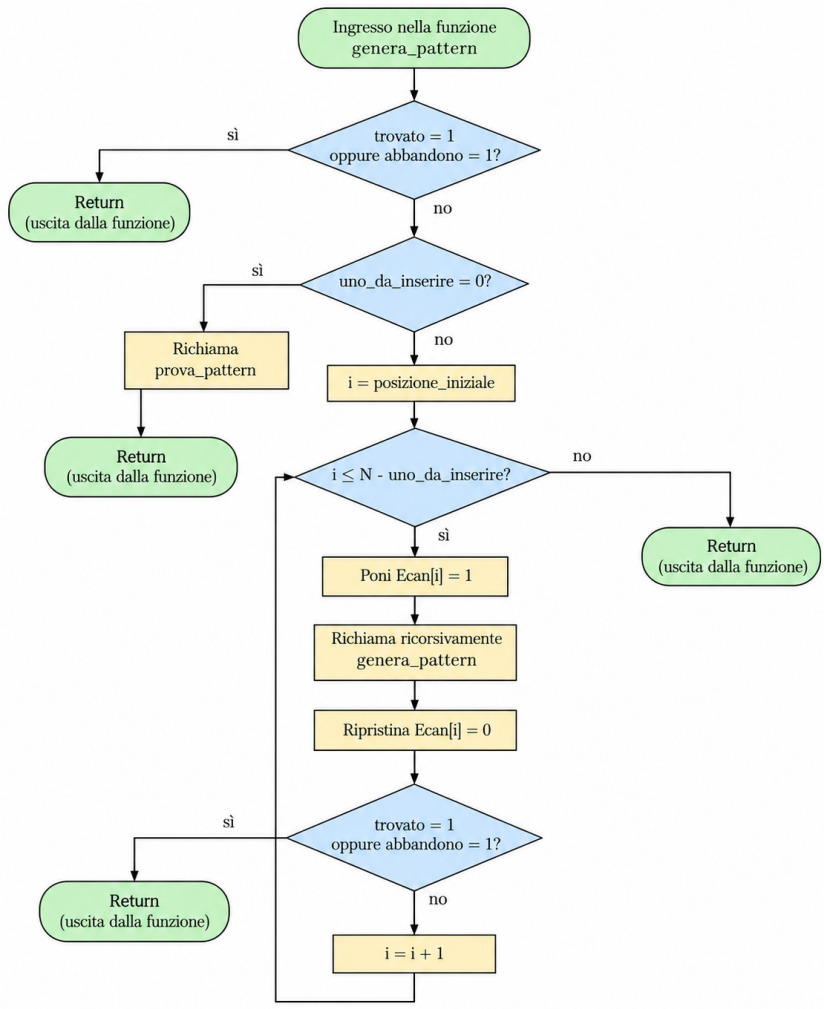


Figura 1.1: Flowchart della generazione ricorsiva dei pattern di errore mediante backtracking.

1.7 Struttura dell'implementazione in C

A partire dal fatto che il test di validità si verifica successivamente alla generazione dei pattern, la realizzazione in C è stata organizzata separando queste due diverse operazioni. La funzione di generazione costruisce i pattern di un determinato peso e una seconda funzione riceve il pattern completo per poter calcolare

$$Z = Y \oplus E_{\text{can}} \tag{1.18}$$

e ne calcola la relativa sindrome, mentre un flag di controllo consente di interrompere sia la ricorsione sia il ciclo sui pesi.

La funzione principale utilizzata per la generazione ricorsiva dei pattern è riportata nel Listato 1.1.

Listato 1.1: Generazione ricorsiva dei pattern di errore a peso di Hamming fissato.

```
// Genera ricorsivamente tutti i pattern a peso di Hamming fissato
void genera_pattern(int Y[], int H[NH][N], int Ecan[], int Z[], int SZ[NH],
    int Xcan[], int Efin[], int* trovato, int uno_da_inserire,
    int posizione_iniziale) {

    int i;

    // Arresto in caso di successo o raggiungimento del limite di query
    if (*trovato == 1 || abbandono == 1) {
        return;
    }

    // Pattern completo => viene sottoposto al controllo
    if (uno_da_inserire == 0) {
        prova_pattern(Y, H, Ecan, Z, SZ, Xcan, Efin, trovato);
        return;
    }

    // Scelta ricorsiva della posizione del prossimo 1
    for (i = posizione_iniziale; i <= N - uno_da_inserire; i++) {

        Ecan[i] = 1;

        genera_pattern(Y, H, Ecan, Z, SZ, Xcan, Efin, trovato,
            uno_da_inserire - 1, i + 1);

        // Backtracking
        Ecan[i] = 0;

        if (*trovato == 1 || abbandono == 1) {
            return;
        }
```

```
33      }
34  }
```

Questa separazione, per di più, permette di mantenere il codice leggibile e flessibile. Le operazioni ausiliarie, come l'azzeramento o la copia dei vettori, restano, invece, distinte dalla logica di decodifica. Quindi, il ciclo principale ha l'unico fine di far variare il peso di Hamming da 0 a N e richiamare il generatore dei corrispondenti pattern.

Una volta completato il pattern candidato, questo viene passato alla funzione `prova_pattern`, riportata nel Listato 1.2. La funzione costruisce la candidata tramite operatore XOR, aggiorna il numero di query effettuate e ne verifica l'appartenenza al codice attraverso il controllo di sindrome.

Listato 1.2: Verifica di un pattern candidato mediante controllo di sindrome.

```
1  // Verifica un pattern candidato gia' costruito
2  void prova_pattern(int Y[], int H[NH][N], int Ecan[], int Z[],
3      int SZ[NH], int Xcan[], int Efin[], int* trovato) {
4
5      int i;
6
7      // Costruzione della candidata Z = Y XOR Ecan
8      for (i = 0; i < N; i++) {
9          Z[i] = Y[i] ^ Ecan[i];
10     }
11
12     // Arresto se e' gia' stato raggiunto il numero massimo di query
13     if (numero_query >= MAX_QUERY) {
14         abbandono = 1;
15         return;
16     }
17
18     // Ogni controllo di sindrome corrisponde a una nuova query
19     numero_query++;
20
21     // Calcolo della sindrome della candidata
22     calcola_sindrome_sparsa(Z, SZ);
23
24     // Stampe utilizzate soltanto per debug e validazione
25     if (STAMPE_DETTAGLIATE) {
26         stampa_vettore(Ecan, N, "Ecan␣=␣");
27         stampa_vettore(Z, N, "Z␣␣␣=␣");
28         stampa_vettore(SZ, NH, "SZ␣␣␣=␣");
29     }
30
31     // Sindrome nulla => la candidata appartiene al codice
32     if (sindrome_nulla(SZ) == 1) {
33         *trovato = 1;
```

```
34          copia_vettore(Xcan, Z, N); // Salvataggio della codeword stimata
35          copia_vettore(Efin, Ecan, N); // Salvataggio del pattern finale
36      }
37  }
```

1.8 Codice LDPC utilizzato

Per le simulazioni è stato utilizzato un codice LDPC regolare, la cui matrice di controllo di parità H è stata costruita secondo il metodo introdotto da Gallager in [6]. In tale costruzione, la matrice è caratterizzata da un numero fissato di elementi non nulli per ogni colonna e per ogni riga, indicati rispettivamente con w_c e w_r. Nel lavoro svolto, la lunghezza delle codeword risulta

$$N = 128 \tag{1.19}$$

e la matrice di controllo di parità contiene 64 righe, con peso di riga

$$w_r = 8 \tag{1.20}$$

e peso di colonna

$$w_c = 4. \tag{1.21}$$

Il rango della matrice è

$$\operatorname{rank}(H) = 61 \tag{1.22}$$

e la dimensione del codice risulta, quindi,

$$K = N - \operatorname{rank}(H) = 128 - 61 = 67. \tag{1.23}$$

Il programma costruisce la matrice generatrice G e utilizza messaggi informativi casuali, per enfatizzare la logica secondo la quale dal lato ricevitore non si conosce l'informazione trasmessa. In questo modo, risulta possibile simulare blocchi differenti senza fissare a priori una singola codeword.

Poiché il numero di pattern da esplorare cresce combinatoriamente con il peso di Hamming, la ricerca viene limitata mediante l'introduzione di un numero massimo di query pari a

$$Q_{\max} = 2.5 \cdot 10^6, \tag{1.24}$$

dove per query si intende il controllo di sindrome a cui si sottopone ogni parola candidata. Se il limite viene raggiunto senza aver trovato una candidata valida, il blocco viene classificato come abbandono.

La struttura sparsa della matrice H del codice LDPC adottato è stata sfruttata per rendere più efficiente il calcolo della sindrome e rendere meno complessa l'architettura implementativa. Infatti, poiché ciascuna riga contiene soltanto 8 elementi pari a 1, il controllo può utilizzare direttamente le posizioni non nulle, invece di scorrere tutte le 128 colonne a ogni query. Questa ottimizzazione non modifica di per sé la logica del decoder, ma riduce ulteriormente il lavoro ripetuto milioni di volte durante le simulazioni.

1.9 Collegamento con la valutazione numerica

La versione finale di GRAND hard viene, infine, inserita in una simulazione multi-blocco su canale BSC. Per ogni valore di probabilità di crossover p, vengono infatti generati nuovi messaggi informativi che vengono codificati e poi trasmessi lungo il canale. Da qui il decoder opera sulla sola sequenza ricevuta, alterata o meno in base al valore di p, e registra l'esito della ricerca e il numero di query utilizzate.

La simulazione prosegue fino al raggiungimento di 100 errori, dove, con questa dicitura, si intendono sia gli abbandoni che le undetection, ovvero la rilevazione di una parola valida, ma non coincidente con quella trasmessa. Le relative curve di *Block Error Rate* e le misure di complessità ottenute con questo procedimento verranno presentate separatamente nel capitolo dedicato ai risultati numerici.

Capitolo 2

ORBGRAND

L'algoritmo *Ordered Reliability Bits Guessing Random Additive Noise Decoding* (ORBGRAND) nasce come estensione soft del paradigma GRAND. Nel caso hard, analizzato nel precedente capitolo, il decoder, in base al peso di Hamming, svolge la decodifica lavorando su sequenze binarie ipotizzando, a sua volta, le ipotesi di rumore. Nel BSC, infatti, non è necessario distinguere l'affidabilità delle singole posizioni, poiché ogni bit è soggetto alla stessa probabilità di crossover e il canale non fornisce un'informazione soft differenziata tra i bit ricevuti. Considerando, invece, modelli di canale come l'AWGN, i campioni ricevuti assumono valori reali e possono fornire un'indicazione del diverso grado di affidabilità associato alle singole decisioni.

In questo contesto, per sua natura, GRAND hard non sfrutta tale informazione aggiuntiva; si introduce allora ORBGRAND, che utilizza le affidabilità per modificare l'ordine con cui vengono formulate le ipotesi di rumore. Rimane, invece, invariato il principio di verifica delle parole candidate tramite controllo di sindrome [4].

2.1 Dal GRAND hard all'informazione soft

Nel caso di trasmissione su canale AWGN con modulazione BPSK, il ricevitore non osserva direttamente una sequenza binaria, ma campioni reali ottenuti dai simboli trasmessi perturbati dal rumore additivo. Indicando con s_i il simbolo trasmesso e con n_i il campione di rumore, il valore ricevuto è

$$y_i = s_i + n_i, \tag{2.1}$$

dove per y_i si intende il simbolo ricevuto e per n_i il rispettivo rumore gaussiano.

Pertanto, tali valori reali contengono sia l'informazione necessaria per prendere una decisione sul bit, tramite *decision boundaries*, sia un'indicazione di quanto tale decisione sia affidabile. Un campione molto distante dalla soglia è generalmente più affidabile di un campione che, invece, ne è prossimo.

2.2 Modulazione BPSK e canale AWGN

Nel lavoro svolto viene adottata una modulazione BPSK con convenzione

$$0 \to +\sqrt{E_s}, \qquad 1 \to -\sqrt{E_s}. \tag{2.2}$$

Con energia normalizzata $E_s = 1$, la mappatura si riduce a

$$0 \to +1, \qquad 1 \to -1. \tag{2.3}$$

Il canale AWGN aggiunge ad ogni simbolo un campione di rumore gaussiano indipendente

$$n_i \sim \mathcal{N}(0, \sigma^2), \tag{2.4}$$

ottenendo, così, esattamente

$$y_i = s_i + n_i. \tag{2.5}$$

Per un codice con rate

$$R_c = \frac{K}{N} \tag{2.6}$$

ed una modulazione BPSK ad energia del simbolo unitaria, si osserva che la varianza del rumore viene ricavata dal rapporto E_b/N_0 secondo [2]

$$\sigma^2 = \frac{E_s}{2R_c\gamma_b}, \qquad \gamma_b = \left(\frac{E_b}{N_0}\right)_{\text{lin}}. \tag{2.7}$$

Tale rapporto viene solitamente espresso in dB, e per il calcolo di (2.7) viene prima convertito in scala lineare attraverso

$$\gamma_b = 10^{\frac{(E_b/N_0)_{\text{dB}}}{10}}. \tag{2.8}$$

In questo modo, il livello di rumore della simulazione può essere modificato a partire dalla scelta di E_b/N_0, parametro che verrà utilizzato, in seguito, nello sweep numerico.

2.3 Generazione del rumore mediante Box-Muller

Dal punto di vista implementativo, la libreria standard del linguaggio C fornisce la possibilità di lavorare con campioni pseudo-casuali, derivanti, tuttavia, da una distribuzione di probabilità uniforme. Per questo motivo, al fine di ottenere campioni gaussiani di rumore, è stata utilizzata la trasformazione di Box-Muller [7]. Quest'ultima permette di costruire una variabile normale standard, esattamente come quella desiderata, a partire da due variabili uniformi indipendenti. Date, infatti,

$$U_1, U_2 \sim \mathcal{U}(0, 1), \tag{2.9}$$

si può ricavare

$$Z = \sqrt{-2\ln(U_1)}\cos(2\pi U_2), \qquad Z \sim \mathcal{N}(0,1). \tag{2.10}$$

Il campione standard, di conseguenza, viene scalato mediante la deviazione standard del canale, σ, ottenendo così

$$n = \sigma Z \sim \mathcal{N}(0, \sigma^2). \tag{2.11}$$

2.4 Decisione hard, LLR e affidabilità

ORBGRAND, come già detto, lavora con informazione soft, ottenuta come vettore reale

$$Y_{\text{soft}} = (y_1, y_2, \ldots, y_N), \tag{2.12}$$

costituito dagli elementi y_i a partire dal quale il ricevitore costruisce la decisione hard e la misura dell'affidabilità. Con la convenzione di modulazione BPSK adottata, per ogni valore ricevuto, la *decision boundary* della costellazione unidimensionale è rappresentata dalla soglia nulla, per cui:

$$Y_{\text{hard},i} = \begin{cases} 0, & y_i \geq 0, \\ 1, & y_i < 0. \end{cases} \tag{2.13}$$

Dal punto di vista della ricezione della sequenza in uscita dal canale, sembra perdersi l'effettivo contributo dell'informazione soft, in quanto il decoder tramuta il vettore Y_{soft} in Y_{hard}. Tuttavia, il contributo principale viene apportato dalla misura dell'affidabilità.

Infatti, nel caso BPSK, con simboli equiprobabili, il *Log-Likelihood Ratio* può essere scritto come

$$L_i = \frac{2\sqrt{E_s}}{\sigma^2} y_i. \tag{2.14}$$

Risalta all'occhio la proporzionalità diretta con i valori y_i. Pertanto, il segno dell'LLR è coerente con la decisione del bit, mentre il modulo dell'LLR, $|L_i|$, rappresenta il grado di affidabilità, vale a dire che valori piccoli indicano maggiore incertezza, mentre valori maggiori corrispondono a decisioni più affidabili.

Data, inoltre, la proporzionalità tra LLR e il valore ricevuto, ordinare i bit secondo

$$|L_i| \tag{2.15}$$

equivale, ai soli fini di ordinamento, a ordinarli secondo

$$|y_i|. \tag{2.16}$$

2.5 Permutazione di affidabilità e distinzione tra rango e posizione fisica

Dopo aver ordinato i valori di $|L_i|$, per poterli tramutare in una gerarchia di affidabilità nel vettore Y_{hard}, occorre una permutazione. ORBGRAND introduce una permutazione π tale che

$$|L_{\pi(1)}| \leq |L_{\pi(2)}| \leq \cdots \leq |L_{\pi(N)}|. \tag{2.17}$$

Si può osservare come $\pi(1)$ indica la posizione fisica del bit meno affidabile, $\pi(2)$ quella del secondo bit meno affidabile e così via. Pertanto, risulta rilevante poter distinguere il rango di affidabilità dalla posizione fisica nella codeword. Grazie a tale permutazione si può caratterizzare un rango $r = 1$ tale per cui si intende il bit meno affidabile, con quest'ultimo che può trovarsi in una qualunque posizione della sequenza ricevuta.

Nell'implementazione in C, l'ordinamento viene applicato agli indici delle posizioni e non direttamente ai valori di affidabilità. Così facendo, la permutazione π mantiene l'associazione tra ciascun rango di affidabilità e la corrispondente posizione fisica nella codeword.

I pattern di errore vengono inizialmente costruiti nel dominio dei ranghi. Successivamente, mediante la permutazione π, ciascun rango viene ricondotto alla relativa posizione fisica, ottenendo così il pattern da applicare a Y_{hard}.

Ad esempio, con

$$\pi(1) = 4 \tag{2.18}$$

si intende che il primo bit meno affidabile ha posizione fisica 4 da cui, nel dominio dei ranghi, verrà posto un simbolo 1 in prima posizione, ma il pattern di errore effettivo da testare presenterà un simbolo 1 in posizione 4.

L'ordinamento delle affidabilità viene realizzato senza modificare direttamente il vettore dei valori $|L_i|$. La funzione `ordina_affidabilita` riordina gli indici contenuti nella permutazione π, associando i primi ranghi ai bit meno affidabili. L'implementazione è riportata nel Listato 2.1.

Listato 2.1: Ordinamento crescente delle affidabilita' e costruzione della permutazione pi.

```
1  void ordina_affidabilita(double Affidabilita[], int pi[]) {
2
3     int i;
4     int j;
5     int indice_min;
6     int temp;
7
8     // Selection sort sugli indici contenuti in pi
```

```
9     for (i = 0; i < N - 1; i++) {
10
11        indice_min = i;
12
13        // Ricerca del bit meno affidabile tra quelli rimanenti
14        for (j = i + 1; j < N; j++) {
15
16            if (Affidabilita[pi[j]] <
17                Affidabilita[pi[indice_min]]) {
18
19                indice_min = j;
20            }
21        }
22
23        // Scambio degli indici fisici nella permutazione
24        if (indice_min != i) {
25
26            temp = pi[i];
27            pi[i] = pi[indice_min];
28            pi[indice_min] = temp;
29        }
30    }
31 }
```

2.6 Logistic Weight

Nel dominio dei ranghi, dunque, ORBGRAND assegna a un pattern di errore il Logistic Weight

$$w_L(E) = \sum_{r:e_r=1} r. \tag{2.19}$$

Questa misura non considera solamente il numero di bit ipotizzati errati, ma anche, secondo la loro posizione nel dominio dei ranghi, il loro grado di affidabilità.

Un errore sul bit meno affidabile ha costo 1, un errore sul terzo bit meno affidabile avrà, di conseguenza, costo 3. Tuttavia, anche due errori sui primi due bit meno affidabili hanno costo $1 + 2 = 3$.

ORBGRAND esplora, così, i pattern secondo i valori crescenti di w_L, privilegiando le ipotesi che coinvolgono le posizioni meno affidabili [4]. Ovviamente, si segue la stessa filosofia adottata da GRAND, secondo cui il generatore di pattern da testare non può ammettere generazioni di valori crescenti di Logistic Weight senza aver prima terminato l'esplorazione e il conseguente test di tutti i pattern associati ai valori inferiori di Logistic Weight.

Il Logistic Weight rappresenta, in questo modo, un'approssimazione efficiente dell'ordinamento probabilistico. Utilizza la posizione dei bit nella classifica di af-

fidabilità, ma non conserva le differenze numeriche tra i corrispondenti valori di affidabilità. In particolare, due bit associati a ranghi consecutivi ricevono costi, a loro volta, consecutivi indipendentemente dal fatto che i rispettivi valori di $|L_i|$ siano molto vicini oppure molto distanti tra loro. Questa perdita delle distanze di affidabilità costituisce la motivazione per una variante di ORBGRAND, quale CDF-ORBGRAND, che verrà presentato nel capitolo successivo.

2.7 Generazione dei pattern secondo Logistic Weight

Tra GRAND e ORBGRAND, cambiando la metrica, cambia anche la generazione dei pattern. Fissato un valore w_L, devono essere individuate tutte le combinazioni di ranghi distinti, la cui somma sia uguale al Logistic Weight richiesto, come esplicitato analiticamente. Il problema, dunque, può essere ricondotto alle partizioni intere in parti distinte, con ranghi compresi tra 1 e N.

Per esempio, per

$$w_L = 4, \tag{2.20}$$

sono ammesse le combinazioni

$$4 \qquad \text{e} \qquad 1+3, \tag{2.21}$$

mentre

$$2+2 \tag{2.22}$$

non è valida, perché lo stesso rango verrebbe utilizzato doppiamente. Selezionando tutti i ranghi, il Logistic Weight massimo si ottiene come:

$$w_{L,\max} = 1+2+\cdots+N = \frac{N(N+1)}{2}. \tag{2.23}$$

Anche in questo caso, è stata utilizzata una procedura ricorsiva di backtracking. L'utilizzo di una funzione che mantiene il peso rimanente ancora da distribuire e il primo rango ancora disponibile rappresenta l'analogo di GRAND hard corrispondente ai soli uni da inserire.

Nel momento in cui viene scelto uno specifico rango r, il peso rimanente viene ridotto di r

$$\text{peso_rimanente} \leftarrow \text{peso_rimanente} - r \tag{2.24}$$

e la chiamata successiva parte da

$$r+1. \tag{2.25}$$

Implementativamente, questa scelta impedisce il riutilizzo dello stesso rango, evitando duplicazioni.

La procedura descritta è implementata mediante la funzione ricorsiva `genera_pattern_logistici`, riportata nel Listato 2.2. Il parametro `peso_rimanente` rappresenta la parte del Logistic Weight ancora da costruire, mentre `rango_iniziale` impedisce il riutilizzo dei ranghi già selezionati.

Listato 2.2: Generazione ricorsiva dei pattern secondo il Logistic Weight.

```
void genera_pattern_logistici(int peso_rimanente,
    int rango_iniziale, int Erank[], int Ecan[], int pi[],
    int Yhard[], int H[NH][N], int Z[], int SZ[NH],
    int Xcan[], int Efin[], int* trovato) {

    int rango;

    // Arresto in caso di successo o raggiungimento del limite di query
    if (*trovato == 1 || abbandono == 1) {
        return;
    }

    // Logistic Weight completamente costruito
    if (peso_rimanente == 0) {

        // Conversione dal dominio dei ranghi alle posizioni fisiche
        mappa_pattern(Erank, Ecan, pi);

        // Verifica del pattern candidato
        prova_pattern(Yhard, H, Ecan, Z, SZ,
            Xcan, Efin, trovato);

        return;
    }

    // Scansione dei ranghi ancora utilizzabili
    for (rango = rango_iniziale; rango <= N; rango++) {

        // Il rango non deve superare il peso ancora da costruire
        if (rango <= peso_rimanente) {

            Erank[rango - 1] = 1;

            // Ricorsione utilizzando soltanto ranghi successivi
            genera_pattern_logistici(
                peso_rimanente - rango,
                rango + 1,
                Erank, Ecan, pi, Yhard, H,
                Z, SZ, Xcan, Efin, trovato);
```

```
40
41              // Backtracking
42              Erank[rango - 1] = 0;
43
44              if (*trovato == 1 || abbandono == 1) {
45                  return;
46              }
47          }
48      }
49  }
```

2.8 Dal dominio dei ranghi alle posizioni fisiche

Operativamente, i pattern generati sono inizialmente nel dominio dei ranghi. Ciò significa che, ad esempio, con

$$N = 5, \qquad w_L = 3, \qquad \pi = (3, 2, 4, 5, 1), \tag{2.26}$$

il pattern prodotto dal generatore sarà, per $1 + 2 = 3$,

$$E_{\text{rank}} = 11000. \tag{2.27}$$

Poiché i ranghi attivi sono 1 e 2, attraverso la permutazione π essi corrispondono rispettivamente alle posizioni fisiche $\pi(1) = 3$ e $\pi(2) = 2$. Il corrispondente pattern fisico risulta, quindi,

$$E_{\text{can}} = 01100. \tag{2.28}$$

I pedici *rank* e *can* indicano proprio il dominio di definizione dei pattern stessi.

Dunque, prima del test della sindrome, i pattern E_{rank} vanno trasformati nei pattern fisici E_{can}, utilizzando la permutazione π.

Per un rango r attivo in E_{rank}, viene posto a 1 il bit situato nella posizione fisica $\pi(r)$.

La relazione concettuale può essere rappresentata come

$$E_{\text{rank}} \xrightarrow{\pi} E_{\text{can}}. \tag{2.29}$$

La corrispondente operazione viene realizzata in C mediante la funzione `mappa_pattern`, riportata nel Listato 2.3. Per ogni rango attivo nel pattern E_{rank}, la permutazione π permette di individuare la corrispondente posizione fisica da porre a 1 in E_{can}.

Listato 2.3: Mappatura del pattern dal dominio dei ranghi alle posizioni fisiche.

```
1  void mappa_pattern(int Erank[], int Ecan[], int pi[]) {
2
3      int k;
```

```
4
5       // Il nuovo pattern fisico parte dal vettore nullo
6       azzera_vettore(Ecan, N);
7
8       // Scansione dei ranghi di affidabilita'
9       for (k = 0; k < N; k++) {
10
11          if (Erank[k] == 1) {
12
13              // Il rango k+1 corrisponde alla posizione fisica pi[k]
14              Ecan[pi[k]] = 1;
15          }
16      }
17  }
```

A partire da E_{can} viene costruita la parola candidata

$$Z = Y_{\text{hard}} \oplus E_{\text{can}}, \tag{2.30}$$

a cui verrà applicato il test di appartenenza della codeword al codebook, ossia:

$$ZH^T = 0. \tag{2.31}$$

La candidata verrà, dunque, accettata quando la sindrome sarà nulla, esattamente come in GRAND hard.

Pertanto, l'informazione soft non modifica il test di appartenenza al codice, ma ne modifica solo l'ordine con cui si formulano le ipotesi di rumore.

2.9 Esempio numerico su codice toy

Per concretizzare la differenza rispetto al decoder hard, si considera lo stesso tipo di codice lineare utilizzato nelle verifiche iniziali del lavoro. Fissando, per semplicità, un messaggio informativo

$$M = [0\ 1], \tag{2.32}$$

si ottiene

$$X = 01111 \tag{2.33}$$

e, con BPSK,

$$S = (+1, -1, -1, -1, -1). \tag{2.34}$$

Si consideri come esempio il vettore ricevuto

$$Y_{\text{soft}} = (-0.1, -1.7, -0.8, -2.4, +0.2). \tag{2.35}$$

La decisione hard è

$$Y_{\text{hard}} = 11110. \tag{2.36}$$

Poiché, ai soli fini dell'ordinamento, ordinare i bit secondo $|L_i|$ equivale a ordinarli secondo $|y_i|$, dai campioni ricevuti si ottiene la permutazione

$$\pi = (1, 5, 3, 2, 4). \tag{2.37}$$

Per $w_L = 0$, il pattern nullo non produce una parola valida. Per $w_L = 1$ viene testato il solo rango 1, analogamente anche per $w_L = 2$.

Con $w_L = 3$, sono possibili le combinazioni

$$1 + 2 \qquad \text{e} \qquad 3. \tag{2.38}$$

La combinazione $1 + 2$ seleziona i ranghi 1 e 2, che mediante π corrispondono alle posizioni fisiche 1 e 5, dunque il pattern

$$E_{\text{can}} = 10001 \tag{2.39}$$

condurrà alla candidata corretta

$$Z = 01111 \tag{2.40}$$

e alla sindrome nulla.

L'esempio esplicita il vantaggio teorico dell'ordinamento soft. Il pattern corretto ha peso di Hamming 2, ma coinvolge i due bit meno affidabili. Pertanto, il pattern corretto può essere testato prima nell'ordinamento ORBGRAND rispetto all'ordinamento hard basato sul peso di Hamming.

2.10 Struttura dell'implementazione in C

La struttura complessiva del decoder ORBGRAND implementato è sintetizzata nel flowchart riportato in Figura 2.1. A partire dall'informazione soft ricevuta, il decoder ordina i bit secondo affidabilità, genera i pattern nel dominio dei ranghi in ordine crescente di Logistic Weight e li riconduce alle corrispondenti posizioni fisiche prima del controllo di sindrome.

L'implementazione è stata costruita mantenendo distinti i vari blocchi logici. Partendo dalla costruzione del canale e la relativa generazione del rumore, seguono modulazione BPSK, decisione hard, calcolo degli LLR, costruzione della permutazione π, generazione dei pattern secondo il Logistic Weight, mappatura tramite π del pattern dal dominio dei ranghi al corrispondente pattern fisico e controllo finale di sindrome.

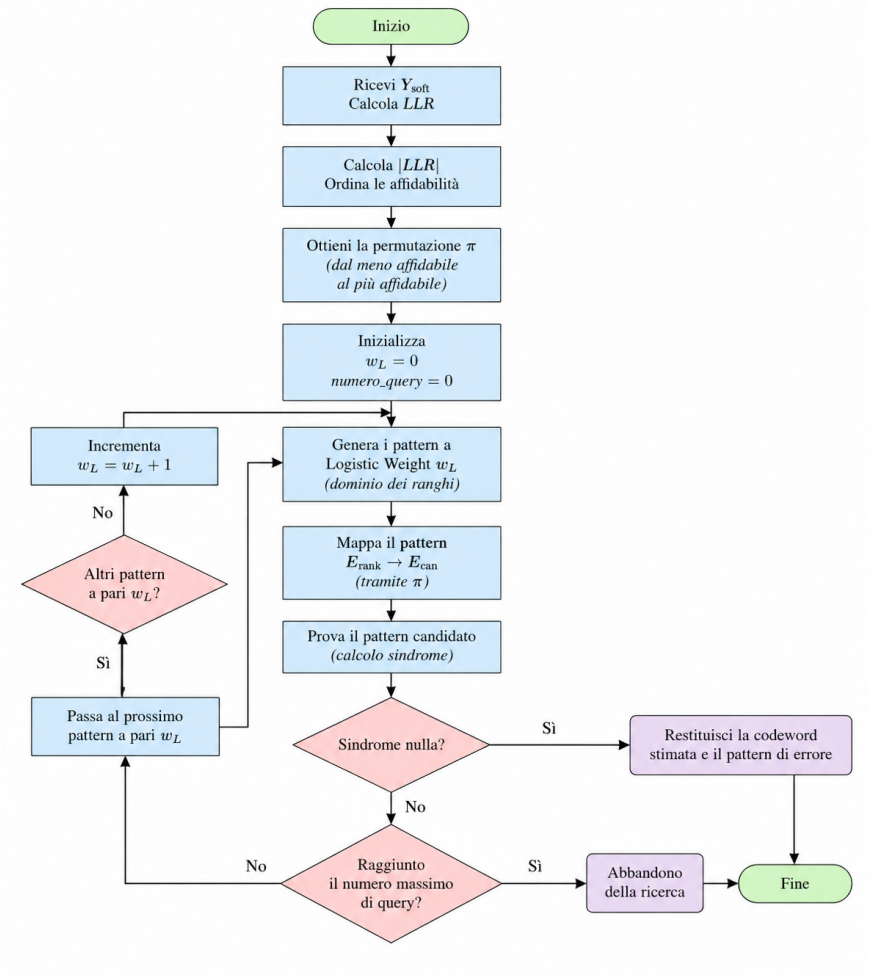


Figura 2.1: Flowchart del procedimento di decodifica ORBGRAND.

A tal proposito, per la permutazione è stato adottato un ordinamento degli indici in funzione delle affidabilità, evitando di modificare il vettore originale.

Invece, il generatore opera direttamente nel dominio dei ranghi, costruendo la sequenza E_{rank} in maniera ricorsiva e producendo tutte le combinazioni compatibili e coerenti al valore attuale di w_L. Una variabile di controllo arresta la ricorsione e il ciclo sui Logistic Weight non appena viene trovata una parola valida appartenente al codebook.

2.11 Applicazione al codice LDPC

Per le simulazioni di ORBGRAND è stato utilizzato lo stesso codice LDPC descritto nella Sezione 1.8. La struttura di codice verrà mantenuta anche per il confronto successivo con CDF-ORBGRAND, in modo da isolare perfettamente il carattere distintivo proprio di ogni algoritmo, ovvero l'ordine di guessing.

Anche per ORBGRAND, la ricerca è soggetta a un limite massimo di query,

$$Q_{\max} = 2.5 \cdot 10^6, \tag{2.41}$$

con ogni candidata sottoposta al controllo di sindrome che costituisce, per definizione, una query.

Di nuovo, per rendere più efficiente il calcolo della sindrome, operazione che viene ripetuta milioni di volte, viene sfruttata la struttura sparsa di H.

La versione finale di tale algoritmo di decodifica viene inserita in una simulazione multi-blocco su AWGN. Per ciascun valore di E_b/N_0, vengono generati nuovi messaggi informativi e nuove realizzazioni del rumore, da cui il decoder registra l'esito della decodifica e il numero di query. Le curve di *Block Error Rate* e di complessità verranno riportate nel capitolo dedicato ai risultati numerici.

Capitolo 3

CDF-ORBGRAND

Nel precedente capitolo, ORBGRAND è stato introdotto come algoritmo di decodifica soft, che sfrutta l'ordinamento delle affidabilità dei bit. Il Logistic Weight assegna al rango r un costo pari proprio a r e ordina i pattern in base alla somma dei ranghi coinvolti, seguendo la definizione analitica. Questa scelta mantiene una struttura semplice e soprattutto indipendente dai valori numerici esatti degli LLR, ma conserva il solo ordine relativo delle affidabilità. Per essere più concreti, si assume, ad esempio, un vettore ricevuto del tipo

$$Y_{\text{soft}} = (0.05, -4.50, +0.06, -8.00, +4.60). \tag{3.1}$$

Poiché, ai soli fini dell'ordinamento, i moduli dei campioni ricevuti e degli LLR sono proporzionali, si ha

$$|Y_{\text{soft}}| = (0.05, 4.50, 0.06, 8.00, 4.60). \tag{3.2}$$

Ordinando dal meno affidabile al più affidabile si ottiene

$$0.05 < 0.06 \ll 4.50 < 4.60 < 8.00. \tag{3.3}$$

Quindi, operando come ORBGRAND richiede, la permutazione risulta essere

$$\pi = (1, 3, 2, 5, 4). \tag{3.4}$$

Si osservano differenze numeriche molto diverse tra affidabilità associate a ranghi consecutivi, in particolare tra il rango 2 e il rango 3 e tra il rango 4 e il rango 5. ORBGRAND conserva l'ordine relativo delle affidabilità, ma non l'entità di tali differenze. Infatti, confrontando due pattern nel dominio dei ranghi, definendoli

$$E_{\text{rank},1} = 11000 \qquad \text{e} \qquad E_{\text{rank},2} = 00100, \tag{3.5}$$

dove, rispettivamente, uno indica gli errori sui due bit meno affidabili e l'altro corrisponde all'errore sul solo terzo bit meno affidabile, ambedue sono caratterizzati dallo stesso Logistic Weight

$$w_L = 3. \tag{3.6}$$

Per ORBGRAND i due pattern hanno esattamente lo stesso costo, ossia

$$w_L(11000) = w_L(00100) = 3, \tag{3.7}$$

eppure, guardando i valori soft effettivi, la situazione è molto diversa. Infatti,

$$0.05 + 0.06 = 0.11 \qquad \text{contro} \qquad 4.50. \tag{3.8}$$

Cioè, il primo pattern coinvolge due bit estremamente inaffidabili, mentre il secondo richiede di ipotizzare un errore su un bit che è già molto lontano dalla soglia. Pertanto, ORBGRAND sa che il terzo bit è più affidabile dei primi due, ma non sa di quanto. Il limite di ORBGRAND non riguarda, quindi, la capacità di distinguere quale bit sia meno affidabile, ma la perdita di informazione relativa a quanto un'affidabilità sia distante dalle altre. Questa osservazione suggerisce di modificare il costo attribuito ai ranghi, senza abbandonare la struttura generale basata sulle permutazioni.

A partire da ciò, CDF-ORBGRAND nasce con l'obiettivo di mantenere il vantaggio strutturale dell'ordinamento per ranghi, sostituendo, però, il costo lineare r con un valore ricavato dalla distribuzione statistica del modulo degli LLR. In questo modo, il rango non viene più trattato come una semplice posizione nella gerarchia delle affidabilità, ma viene associato a un valore rappresentativo dell'affidabilità corrispondente [5].

3.1 Rank Companding

Sia $\Psi(t)$ la funzione di ripartizione del modulo degli LLR,

$$\Psi(t) = P(|L| \leq t). \tag{3.9}$$

Per un blocco sufficientemente lungo, il rango normalizzato di un'osservazione approssima il valore della funzione di ripartizione dell'affidabilità. Indicando con $|L|_{(r)}$ l'r-esimo valore più piccolo tra i moduli degli LLR, per una distribuzione continua si ha

$$\mathbb{E}\left[\Psi(|L|_{(r)})\right] = \frac{r}{N+1}. \tag{3.10}$$

Si assume pertanto, per blocchi sufficientemente lunghi,

$$\frac{r}{N+1} \approx \Psi(|L|_{(r)}). \tag{3.11}$$

L'utilizzo di $N+1$ consente di mantenere il rango normalizzato strettamente compreso tra 0 e 1, evitando gli estremi della funzione di ripartizione.

Applicando l'inversa della funzione di ripartizione, viene definito il costo associato al rango, ossia:

$$\gamma_r = \Psi^{-1}\left(\frac{r}{N+1}\right). \tag{3.12}$$

Tale trasformazione prende il nome di *rank companding*. Rispetto ad ORBGRAND, si può osservare come CDF-ORBGRAND utilizzi un quantile γ_r della distribuzione delle affidabilità, invece di attribuire linearmente il costo al rango.

L'ordine dei bit rimane quello determinato dalla permutazione π, cambia, invece, la misura con cui vengono confrontati i pattern.

3.2 Peso CDF dei pattern

Per un pattern espresso nel dominio dei ranghi, il costo complessivo verrà ottenuto sommando i pesi dei ranghi attivi:

$$W_{\mathrm{CDF}}(E) = \sum_{r:e_r=1} \gamma_r. \tag{3.13}$$

CDF-ORBGRAND, quindi, testa tutti i pattern in ordine crescente di W_{CDF}, analogamente a ORBGRAND. Tuttavia, la differenza principale da quest'ultimo consiste nel fatto che da un costo lineare al rango, del tipo

$$w_L(E) = \sum_{r:e_r=1} r, \tag{3.14}$$

si passa a un costo determinato dalla distribuzione statistica dell'affidabilità associata al canale, descritta attraverso il modulo degli LLR. Quest'ultimi sono definiti come

$$L_i = \frac{2\sqrt{E_s}}{\sigma^2} y_i. \tag{3.15}$$

D'altra parte, il principio GRAND rimane invariato, ossia il decoder continua a formulare ipotesi sulle possibili realizzazioni del rumore e accetta la prima candidata che soddisfa i vincoli di parità del codice.

La modifica riguarda solo l'ordine con cui le ipotesi vengono sottoposte a tale controllo di sindrome.

3.3 CDF del modulo dell'LLR nel canale AWGN

Nel sistema considerato, la trasmissione avviene sul canale AWGN con modulazione BPSK. Per la convenzione già introdotta, con energia unitaria, l'LLR è una variabile normale, condizionata al simbolo trasmesso. Ponendo, dunque,

$$s_i = +\sqrt{E_s}, \tag{3.16}$$

si ha una media e una deviazione standard dell'LLR date da

$$\mu_L = \frac{2E_s}{\sigma^2}, \qquad \sigma_L = \frac{2\sqrt{E_s}}{\sigma}. \tag{3.17}$$

Per simmetria e per equiprobabilità, la distribuzione di $|L|$ è la stessa per i due simboli BPSK. La CDF del modulo può essere espressa mediante la CDF normale standard Φ come

$$\Psi(t) = \Phi\left(\frac{t-\mu_L}{\sigma_L}\right) + \Phi\left(\frac{t+\mu_L}{\sigma_L}\right) - 1, \qquad t \geq 0. \tag{3.18}$$

Nel programma, l'inversa Ψ^{-1} non viene inserita tramite una formula chiusa, ma viene calcolata numericamente mediante bisezione. Infatti, per ogni valore di E_b/N_0, si ricava σ e, per ciascun rango r, viene calcolato γ_r corrispondente al rango normalizzato

$$\frac{r}{N+1}. \tag{3.19}$$

Di conseguenza, i pesi CDF vengono scelti automaticamente dalle condizioni del canale.

3.4 Significato del risultato teorico

Il lavoro di Li e Zhang associa al *rank companding* un risultato teorico di *achievable rate*, mostrando che, per canali senza memoria con ingresso binario e distribuzione d'ingresso simmetrica, CDF-ORBGRAND raggiunge asintoticamente la capacità simmetrica [5].

Ovviamente, questo risultato deve essere interpretato con cautela, nel contesto del lavoro svolto. Tale proprietà vale solo asintoticamente, dunque per lunghezze di codice che tendono all'infinito. Peraltro, la stessa analisi teorica non considera il troncamento dovuto al raggiungimento di un numero massimo di query.

Di conseguenza, per un codice LDPC caratterizzato da

$$N = 128 \tag{3.20}$$

e

$$Q_{\max} = 2.5 \cdot 10^6, \tag{3.21}$$

il risultato di [5] non implica che, a lunghezza finita, CDF-ORBGRAND coincida con la decodifica ML o richieda necessariamente meno query di ORBGRAND in ogni condizione.

3.5 Esempio di verifica dell'ordinamento CDF

Per illustrare e verificare il corretto funzionamento dell'ordinamento dei pattern secondo la metrica CDF, è stata effettuata una validazione numerica dell'implementazione sviluppata in linguaggio C.

Come riferimento iniziale è stato considerato l'esempio di lunghezza $N = 5$ presentato da Li e Zhang [5], relativo a una trasmissione BPSK su canale AWGN con SNR pari a 6 dB.

Tale esempio è stato utilizzato come punto di partenza per verificare la correttezza del calcolo dei pesi CDF e del conseguente ordinamento dei pattern prodotto dall'implementazione realizzata.

Il vettore delle affidabilità, espresso tramite i moduli degli LLR, è

$$(|L_1|,\ |L_2|,\ |L_3|,\ |L_4|,\ |L_5|) = (5.17,\ 6.08,\ 7.93,\ 9.56,\ 12.01). \tag{3.22}$$

Poiché i valori sono già ordinati in senso crescente, i ranghi corrispondenti sono

$$(1,\ 2,\ 3,\ 4,\ 5). \tag{3.23}$$

Applicando la trasformazione CDF ai cinque ranghi si ottengono, quindi, tali valori sottoposti al companding

$$(4.12,\ 6.25,\ 7.96,\ 9.68,\ 11.82). \tag{3.24}$$

Per la lunghezza considerata, il numero complessivo di pattern possibili è pari a $2^5 = 32$. Pertanto, è possibile calcolare il peso CDF di ciascun pattern e ottenere un ordinamento di riferimento mediante una procedura esaustiva.

A partire dai valori numerici dell'esempio di riferimento, è stato costruito un caso controllato per verificare il comportamento dell'implementazione sviluppata in C e analizzare le differenze tra l'ordinamento di ORBGRAND e quello di CDF-ORBGRAND.

In particolare, è stata considerata una realizzazione nella quale il pattern di errore effettivo interessa esclusivamente il terzo bit meno affidabile, corrispondente al pattern $\{3\}$ nel dominio dei ranghi.

Il confronto viene effettuato rispetto al pattern $\{1, 2\}$, che rappresenta, invece, due errori sui primi due bit meno affidabili.

Applicando la metrica CDF ai valori precedentemente ottenuti, si ha

$$W_{\mathrm{CDF}}(\{3\}) = \gamma_3 = 7.96, \tag{3.25}$$

mentre

$$W_{\mathrm{CDF}}(\{1, 2\}) = \gamma_1 + \gamma_2 = 10.37. \tag{3.26}$$

Di conseguenza, CDF-ORBGRAND assegna una priorità maggiore al pattern $\{3\}$ rispetto al pattern $\{1, 2\}$.

Al contrario, utilizzando il Logistic Weight di ORBGRAND, entrambi i pattern hanno peso pari a 3:

$$w_L(\{3\}) = w_L(\{1,2\}) = 3. \tag{3.27}$$

ORBGRAND attribuisce quindi ai due pattern la stessa priorità, mentre CDF-ORBGRAND li distingue sulla base della distribuzione statistica delle affidabilità.

Questo esempio evidenzia come la trasformazione CDF possa modificare le priorità di ricerca rispetto al Logistic Weight, senza tuttavia costituire una dimostrazione di miglioramento delle prestazioni di decodifica.

3.6 Il ruolo della min-heap

Per $N = 128$, risulta computazionalmente infattibile generare tutti i

$$2^{128} \tag{3.28}$$

pattern, per poi ordinarli in memoria. Il decoder, dunque, deve costruire gradualmente le ipotesi, producendo solo i candidati che reputa necessari.

Per questo motivo, è stata adottata una struttura *min-heap*, la cui proprietà fondamentale è che il nodo posto in radice è sempre quello con peso minimo tra quelli attualmente presenti. Ogni nodo contiene il pattern nel dominio dei ranghi, insieme al relativo peso CDF e il rango massimo presente.

A partire, dunque, da un pattern il cui rango massimo è j, vengono generati al massimo due nuovi candidati, uno aggiungendo $j+1$ e uno sostituendo j con $j+1$. Di conseguenza, i relativi pesi vengono aggiornati utilizzando i rispettivi valori di γ_r. Poiché i valori γ_r crescono con il rango, l'aggiunta del rango $j+1$ e la sostituzione di j con $j+1$ non producono un costo inferiore a quello del pattern da cui hanno origine. Tale proprietà è coerente con l'esplorazione progressiva dei candidati gestita attraverso la min-heap. Inoltre, l'heap stabilisce quale pattern estrarre successivamente, sulla base del peso totale e indipendentemente dalla profondità raggiunta nell'albero di generazione.

La generazione progressiva dei candidati è realizzata mediante la funzione `genera_nuovi_pattern_cdf`, riportata nel Listato 3.1. A partire da un pattern con rango massimo j, vengono costruiti i due possibili successori ottenuti aggiungendo $j+1$ oppure sostituendo j con $j+1$.

Listato 3.1: Generazione dei successori di un pattern CDF.

```
1 void genera_nuovi_pattern_cdf(NodoHeapCDF nodo_corrente,
2     double pesi_cdf[], HeapCDF* heap) {
3
4     int rango_massimo;
5     NodoHeapCDF pattern_aggiunta;
6     NodoHeapCDF pattern_sostituzione;
7
8     rango_massimo = nodo_corrente.rango_massimo;
```

```
9
10      // Nessun rango successivo disponibile
11      if (rango_massimo >= N) {
12          return;
13      }
14
15      // I due successori partono dal pattern corrente
16      pattern_aggiunta = nodo_corrente;
17      pattern_sostituzione = nodo_corrente;
18
19      // Primo successore: aggiunta del rango j+1
20      inserisci_rango_cdf(&pattern_aggiunta, rango_massimo + 1);
21
22      pattern_aggiunta.peso_cdf =
23          nodo_corrente.peso_cdf + pesi_cdf[rango_massimo];
24
25      pattern_aggiunta.rango_massimo = rango_massimo + 1;
26
27      inserisci_heap(heap, pattern_aggiunta);
28
29      // Secondo successore: sostituzione di j con j+1
30      elimina_rango_cdf(&pattern_sostituzione, rango_massimo);
31      inserisci_rango_cdf(&pattern_sostituzione, rango_massimo + 1);
32
33      pattern_sostituzione.peso_cdf =
34          nodo_corrente.peso_cdf
35          - pesi_cdf[rango_massimo - 1]
36          + pesi_cdf[rango_massimo];
37
38      pattern_sostituzione.rango_massimo = rango_massimo + 1;
39
40      inserisci_heap(heap, pattern_sostituzione);
41  }
```

3.7 Rappresentazione compatta dei pattern per N=128

Per $N = 128$, rappresentare ogni pattern dei ranghi mediante un vettore di N elementi interi comporterebbe un'occupazione di memoria elevata, soprattutto considerando che la min-heap può contenere contemporaneamente un numero significativo di nodi.

Nel progetto, dunque, il pattern E_{rank} viene rappresentato attraverso due interi da 64 bit. Il primo contiene i ranghi

$$1, \ldots, 64, \tag{3.29}$$

il secondo i ranghi

$$65, \ldots, 128. \tag{3.30}$$

In questo modo, i 128 possibili ranghi occupano proprio 128 bit, ossia 16 byte, invece dei 512 byte richiesti da un vettore di 128 elementi interi a 4 byte.

L'inserimento, la rimozione e la verifica di un rango vengono realizzati mediante operazioni bit a bit. Tale scelta non modifica la metrica CDF, ma riduce significativamente l'occupazione di memoria associata ai nodi della min-heap.

La struttura dati utilizzata per rappresentare ciascun nodo della min-heap è riportata nel Listato 3.2. I 128 ranghi vengono suddivisi tra due interi senza segno da 64 bit, mantenendo inoltre nel nodo il peso CDF complessivo e il rango massimo attivo.

Listato 3.2: Rappresentazione compatta di un pattern CDF per N=128.

```
typedef struct {

    uint64_t Erank_basso; // Ranghi da 1 a 64
    uint64_t Erank_alto; // Ranghi da 65 a 128

    double peso_cdf; // Peso CDF complessivo
    int rango_massimo; // Massimo rango presente

} NodoHeapCDF;
```

L'inserimento di un rango nel pattern compatto viene quindi effettuato tramite operazioni bit a bit, distinguendo i primi 64 ranghi dai successivi.

Listato 3.3: Inserimento di un rango nella rappresentazione compatta.

```
void inserisci_rango_cdf(NodoHeapCDF* nodo, int rango) {

    if (rango <= 64) {
        nodo->Erank_basso |= (1ULL << (rango - 1));
    }
    else {
        nodo->Erank_alto |= (1ULL << (rango - 65));
    }
}
```

3.8 Struttura del decoder CDF-ORBGRAND

La struttura complessiva del decoder CDF-ORBGRAND è sintetizzata nel flowchart riportato in Figura 3.1.

Il ciclo di decodifica mantiene lo stesso principio di base dei decoder precedenti. Il pattern nullo viene testato sempre come prima query e se la sindrome non è nulla

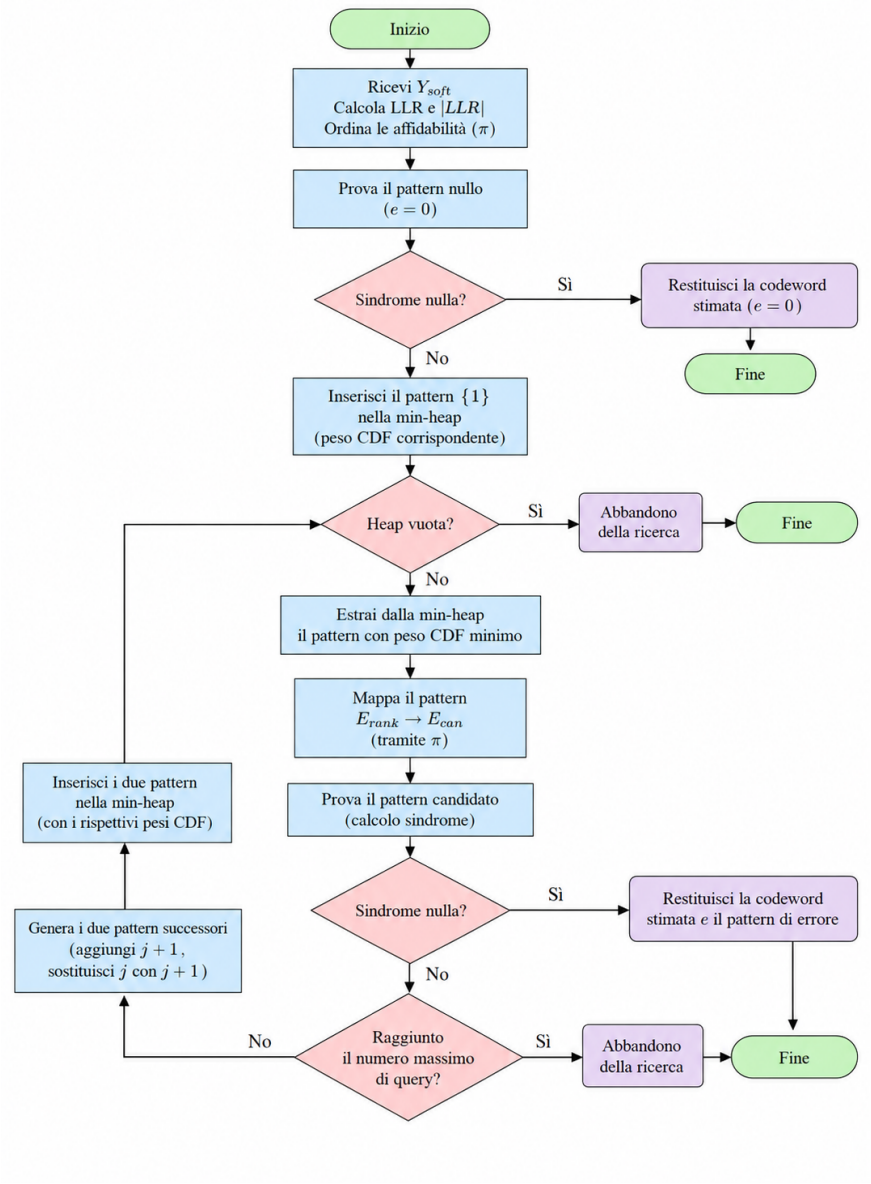


Figura 3.1: Flowchart del procedimento di decodifica CDF-ORBGRAND.

si inserisce nell'heap il primo pattern non nullo, da cui, successivamente, il decoder estrae il nodo con peso minimo, mappa il pattern dal dominio dei ranghi alle posizioni fisiche tramite la permutazione e costruisce la candidata.

Pertanto,

$$Z = Y_{\text{hard}} \oplus E_{\text{can}} \tag{3.31}$$

rimane invariata rispetto al procedimento utilizzato in ORBGRAND.

Ovviamente, se il test della candidata tramite sindrome fallisce e il numero massimo di query non è stato raggiunto, dal nodo corrente vengono generati i nuovi candidati CDF, che vengono inseriti poi nell'heap. Tale procedura continua estraendo il pattern di peso minimo disponibile.

Il ciclo principale della ricerca CDF è riportato nel Listato 3.4. Il pattern nullo viene verificato separatamente; successivamente, ad ogni iterazione, viene estratto dalla min-heap il candidato di peso minimo, mappato nelle posizioni fisiche e sottoposto al controllo di sindrome.

Listato 3.4: Struttura essenziale del ciclo di decodifica CDF-ORBGRAND.

```
// Prima query: pattern nullo
azzera_vettore(Ecan, N);

prova_pattern(Yhard, H, Ecan, Z, SZ,
    Xcan, Efin, &trovato);

// Inizializzazione della min-heap con il rango 1
if (trovato == 0 && abbandono == 0) {

    inserisci_rango_cdf(&nodo_iniziale_decoder_cdf, 1);

    nodo_iniziale_decoder_cdf.peso_cdf = pesi_cdf[0];
    nodo_iniziale_decoder_cdf.rango_massimo = 1;

    inserisci_heap(&heap_decoder_cdf,
        nodo_iniziale_decoder_cdf);
}

// Ricerca in ordine crescente di peso CDF
while (trovato == 0 &&
       abbandono == 0 &&
       estrai_minimo_heap(&heap_decoder_cdf,
           &nodo_corrente_decoder_cdf) == 1) {

    mappa_pattern_cdf(nodo_corrente_decoder_cdf,
        Ecan, pi);

    prova_pattern(Yhard, H, Ecan, Z, SZ,
```

```
29          Xcan, Efin, &trovato);
30
31      if (trovato == 0 && numero_query >= MAX_QUERY) {
32          abbandono = 1;
33          break;
34      }
35
36      if (trovato == 0) {
37          genera_nuovi_pattern_cdf(
38              nodo_corrente_decoder_cdf,
39              pesi_cdf,
40              &heap_decoder_cdf);
41      }
42  }
```

3.9 Validazione dell'ordine a $N = 128$

Per verificare il corretto funzionamento della procedura di ordinamento basata sulla min-heap, è stato considerato un caso con lunghezza $N = 128$ e rapporto $E_b/N_0 = 4\,\text{dB}$. L'obiettivo della verifica è controllare che i pattern vengano estratti dalla min-heap in ordine crescente di peso CDF, indipendentemente dall'ordine con cui sono stati generati.

Applicando la trasformazione descritta nella Sezione 3.1, i pesi associati ai primi quattro ranghi risultano

$$\begin{aligned} \gamma_1 &= 0.117303, \\ \gamma_2 &= 0.234358, \\ \gamma_3 &= 0.350922, \\ \gamma_4 &= 0.466764. \end{aligned} \tag{3.32}$$

I pattern costituiti da un solo rango hanno un peso CDF pari al peso del rango stesso. Per il pattern $\{1, 2\}$, invece, il peso complessivo è

$$W_{\text{CDF}}(\{1, 2\}) = \gamma_1 + \gamma_2 = 0.351661. \tag{3.33}$$

Si osserva, quindi, che

$$\gamma_3 < \gamma_1 + \gamma_2 < \gamma_4. \tag{3.34}$$

Di conseguenza, i primi cinque pattern non nulli, ordinati per peso CDF crescente, sono

$$\{1\}, \quad \{2\}, \quad \{3\}, \quad \{1, 2\}, \quad \{4\}. \tag{3.35}$$

In particolare, il pattern $\{3\}$ deve essere estratto prima del pattern $\{1, 2\}$, nonostante quest'ultimo possa essere stato generato precedentemente. Questo comportamento dipende dalla proprietà della min-heap di selezionare il candidato con peso minimo tra quelli disponibili, anziché seguire l'ordine di generazione.

La sequenza attesa coincide con quella osservata durante l'esecuzione del generatore CDF-ORBGRAND. La verifica conferma quindi, per i pattern considerati, la coerenza tra i pesi calcolati e l'ordine di estrazione dalla min-heap.

3.10 Collegamento con il confronto numerico

Per il confronto numerico, CDF-ORBGRAND utilizza lo stesso codice LDPC e le stesse condizioni di simulazione adottate per ORBGRAND, comprese la trasmissione BPSK su canale AWGN, il criterio di validità tramite sindrome e il limite massimo di query, già definiti nelle sezioni precedenti.

Il confronto viene così effettuato a parità di condizioni, distinguendo i due decoder per la metrica di ordinamento e per la conseguente strategia di generazione dei pattern.

Le curve di BLER, il numero medio di query e il confronto tra i due decoder vengono, quindi, rimandati al capitolo dedicato ai risultati numerici.

Capitolo 4

Risultati numerici

Dopo aver descritto separatamente GRAND, ORBGRAND e CDF-ORBGRAND, il presente capitolo raccoglie i risultati delle simulazioni numeriche. Tale analisi considera sia la probabilità di errore sia l'onere computazionale, misurato mediante il numero di query necessarie per blocco, calcolato per ciascun punto di simulazione.

4.1 Impostazione delle simulazioni

Per ciascun punto vengono generati nuovi messaggi informativi e nuove realizzazioni del canale. Il singolo punto di simulazione è caratterizzato dalla probabilità di crossover p per GRAND hard e dal rapporto E_b/N_0 per ORBGRAND e CDF-ORBGRAND.

La parola trasmessa non viene utilizzata da alcun decoder per scegliere la candidata, ma viene confrontata con l'uscita soltanto a posteriori, ai fini dell'analisi e della conseguente classificazione del risultato.

Gli esiti possibili sono decodifica corretta, undetection e abbandono, con undetection che indica il ritrovamento di una codeword valida, ma diversa da quella trasmessa, mentre l'abbandono si verifica quando viene raggiunto il limite massimo di query, senza individuare una parola valida.

Dunque,

$$N_{\text{errori}} = N_{\text{undetection}} + N_{\text{abbandoni}}. \tag{4.1}$$

Per GRAND hard ciascun punto viene arrestato al raggiungimento di 100 errori sul blocco. Nel confronto a parità di parametri tra ORBGRAND e CDF-ORBGRAND, invece, la simulazione prosegue sugli stessi blocchi finché entrambi i decoder hanno raggiunto almeno 100 errori; di conseguenza, uno dei due può superare tale soglia. Pertanto, il Block Error Rate viene stimato come

$$\text{BLER} = \frac{N_{\text{errori}}}{M} = \frac{N_{\text{undetection}} + N_{\text{abbandoni}}}{M}. \tag{4.2}$$

Una query, ovviamente, coincide con una candidata realmente soggetta al controllo di sindrome, con il pattern nullo che conta come prima query. Per tali simulazioni è stato fissato

$$Q_{\max} = 2.5 \cdot 10^6. \tag{4.3}$$

Il raggiungimento di tale limite senza aver individuato una codeword valida determina un abbandono.

4.2 Risultati di GRAND hard sul canale BSC

GRAND con decisione hard è stato valutato sul Binary Symmetric Channel al variare della probabilità di crossover p. Riducendo p, naturalmente, diminuisce il numero medio di flip per blocco. Di conseguenza, il vero pattern di errore tende ad avere un peso di Hamming più basso e GRAND tende quindi a raggiungerlo prima nella ricerca. I risultati delle simulazioni sono rappresentati nelle Figure 4.1 e 4.2, che mostrano, rispettivamente, l'andamento del BLER e del numero medio di query per blocco al variare della probabilità di crossover p.

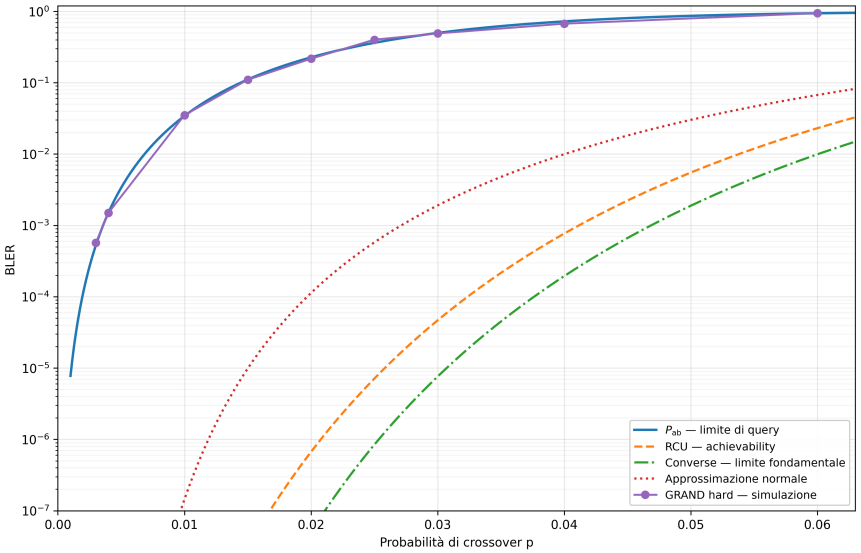


Figura 4.1: Confronto tra il BLER simulato di GRAND hard e i riferimenti teorici sul canale BSC: probabilità di abbandono P_{ab}, bound RCU, converse e approssimazione normale, al variare della probabilità di crossover p.

Oltre ai punti simulati, la Figura 4.1 riporta quattro riferimenti teorici, calcolati per il codice $(N, K) = (128, 67)$ e per $Q_{\max} = 2.5 \cdot 10^6$.

Il primo è la probabilità di abbandono P_{ab}, ossia la probabilità che il vero pattern di errore non rientri tra le prime $Q_{\max}$ ipotesi esplorate da GRAND. Indicando con

$$A_w = \sum_{j=0}^{w-1} \binom{N}{j} \tag{4.4}$$

il numero di query effettuate prima di raggiungere i pattern di peso w, il numero di pattern di peso w testati entro il limite di query è

$$r_w = \min\left\{\max\left(Q_{\max} - A_w,\, 0\right),\, \binom{N}{w}\right\}. \tag{4.5}$$

Poiché nel BSC i pattern di uguale peso sono equiprobabili, la posizione del vero pattern all'interno del proprio livello di peso è distribuita uniformemente e, di conseguenza,

$$P_{\text{ab}} = \sum_{w=0}^{N} \binom{N}{w} p^w (1-p)^{N-w} \left(1 - \frac{r_w}{\binom{N}{w}}\right). \tag{4.6}$$

La grandezza P_{ab} dipende soltanto da N, $Q_{\max}$ e p, e non dal codice, e costituisce un limite inferiore per il BLER: ogni blocco il cui vero pattern non viene raggiunto entro $Q_{\max}$ si traduce necessariamente in un errore. Inoltre, quando il vero pattern rientra nel budget, e ha quindi peso al più 4, una undetection richiederebbe una codeword non nulla di peso al più $4 + 4 = 8$. Se la distanza minima del codice è almeno 9, il BLER coincide perciò esattamente con P_{ab}. La curva supporta questa ipotesi.

Il secondo riferimento è il bound RCU (*Random Coding Union*) per il BSC [8, Teorema 33],

$$\varepsilon \le \sum_{t=0}^{N} \binom{N}{t} p^t (1-p)^{N-t} \min\left\{1,\, (2^K - 1)\, 2^{-N} \sum_{s=0}^{t} \binom{N}{s}\right\}, \tag{4.7}$$

che garantisce l'esistenza di un codice (N, K) il cui BLER, con decodifica a massima verosimiglianza (ML), non supera tale valore. Poiché sul BSC GRAND senza limite di query esplora i pattern in ordine di probabilità non crescente, e realizza quindi una decodifica ML [3], il bound RCU rappresenta le prestazioni raggiungibili da GRAND in assenza di $Q_{\max}$.

Il terzo riferimento è il converse di Polyanskiy, Poor e Verdú [8, Teorema 35], che nel BSC assume la forma di un bound di *sphere packing*. Un codice lineare (N, K) presenta 2^{N-K} sindromi distinte e può quindi correggere al più 2^{N-K} pattern di errore; nel caso più favorevole, questi coincidono con i 2^{N-K} pattern più probabili. Il converse estende tale argomento a qualsiasi codice con 2^K parole e a qualsiasi decoder: detto L il peso tale che $A_L \le 2^{N-K} < A_{L+1}$, e posto $P_w = \binom{N}{w} p^w (1-p)^{N-w}$, risulta

$$\varepsilon \ge \sum_{w=L+1}^{N} P_w + \left(1 - \frac{2^{N-K} - A_L}{\binom{N}{L}}\right) P_L. \tag{4.8}$$

Infine, l'approssimazione normale [8, Teorema 52],

$$\varepsilon \approx \mathcal{Q}\left(\frac{NC - K + \frac{1}{2}\log_2 N}{\sqrt{NV}}\right), \tag{4.9}$$

dove $\mathcal{Q}(\cdot)$ è la funzione Q gaussiana, $C = 1 - h_2(p)$ è la capacità del BSC, con

$h_2(p) = -p\log_2 p - (1-p)\log_2(1-p)$, e $V = p(1-p)\log_2^2 \frac{1-p}{p}$ è la dispersione del canale, è riportata soltanto come riferimento, poiché per valori piccoli di p non risulta affidabile e può scendere al di sotto del converse.

Dal confronto emerge che i punti simulati si sovrappongono alla curva di P_{ab} su tutto l'intervallo di p considerato, entro l'incertezza statistica dovuta ai 100 errori raccolti per punto. Tale accordo è coerente con l'assenza di undetection, discussa nel seguito: tutti gli errori osservati sono abbandoni e il BLER di GRAND hard risulta, quindi, determinato dal limite $Q_{\max}$ piuttosto che dalla struttura del codice.

Il divario rispetto ai bound teorici è, invece, molto ampio. A $p = 10^{-2}$, ad esempio, si ha $P_{\text{ab}} \approx 3.4 \cdot 10^{-2}$, mentre il bound RCU vale circa $4.6 \cdot 10^{-10}$ e il converse circa $4.5 \cdot 10^{-12}$: circa otto ordini di grandezza separano GRAND hard con $Q_{\max} = 2.5 \cdot 10^6$ dalle prestazioni ottenibili con decodifica ML. Tale divario non è imputabile al codice LDPC adottato, ma al budget di query: le $2.5 \cdot 10^6$ query coprono una frazione di circa 10^{-12} dei 2^{61} pattern che un decoder ML potrebbe correggere, e garantiscono la correzione dei soli pattern fino al peso 3. Avvicinarsi al bound RCU richiederebbe, pertanto, un numero di query superiore di molti ordini di grandezza, non sostenibile in simulazione.

Si noti che, in tutti i punti simulati, non sono state osservate undetection; infatti, i 100 errori che determinano l'arresto sono costituiti dai soli abbandoni. Ciò indica che, nelle realizzazioni osservate, l'errore è dovuto al mancato raggiungimento di una codeword valida entro $Q_{\max}$, piuttosto che all'accettazione di una codeword errata.

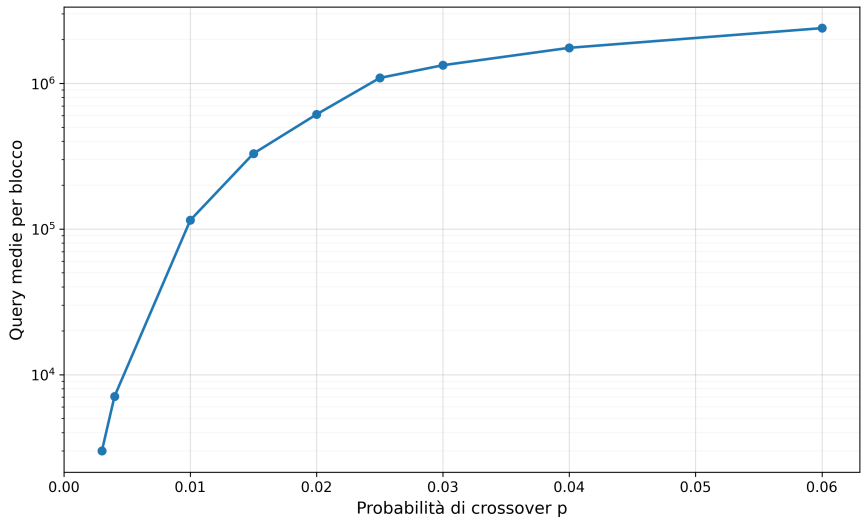


Figura 4.2: Numero medio di query per blocco di GRAND hard al variare della probabilità di crossover p.

Anche la complessità media diminuisce con rapidità al diminuire di p. Si passa, infatti, da circa $2.39 \cdot 10^6$ query per blocco, a $p = 0.06$, a circa $2.99 \cdot 10^3$ query per blocco a $p = 0.003$.

Il comportamento risulta coerente con la struttura di GRAND hard. Infatti,

quando il canale introduce pochi errori, i pattern a basso peso di Hamming possono essere raggiunti facilmente entro le prime fasi di ricerca.

4.3 Confronto ORBGRAND e CDF-ORBGRAND

Il confronto tra ORBGRAND e la sua variante CDF-ORBGRAND è stato effettuato, come già detto, sullo stesso codice LDPC, sul medesimo canale AWGN e con lo stesso limite massimo di query. I valori di E_b/N_0 considerati appartengono al range

$$3\,\text{dB} \leq \frac{E_b}{N_0} \leq 7\,\text{dB} \tag{4.10}$$

con passo di 0.5 dB.

Per rendere il confronto quanto più diretto, i due decoder sono stati sottoposti alle stesse realizzazioni pseudo-casuali di messaggi e rumore, mediante un seme fissato. La scelta, in questo caso, non modifica il funzionamento a monte dei due decoder, ma rende maggiormente interpretabili le differenze come conseguenza delle strategie di formulazione di ipotesi di rumore e conseguente generazione dei pattern, a parità di tutte le altre caratteristiche.

I risultati del confronto sono rappresentati nelle Figure 4.3 e 4.4, che mostrano, rispettivamente, l'andamento del BLER e del numero medio di query per blocco dei due decoder al variare di E_b/N_0.

4.3.1 Confronto del BLER

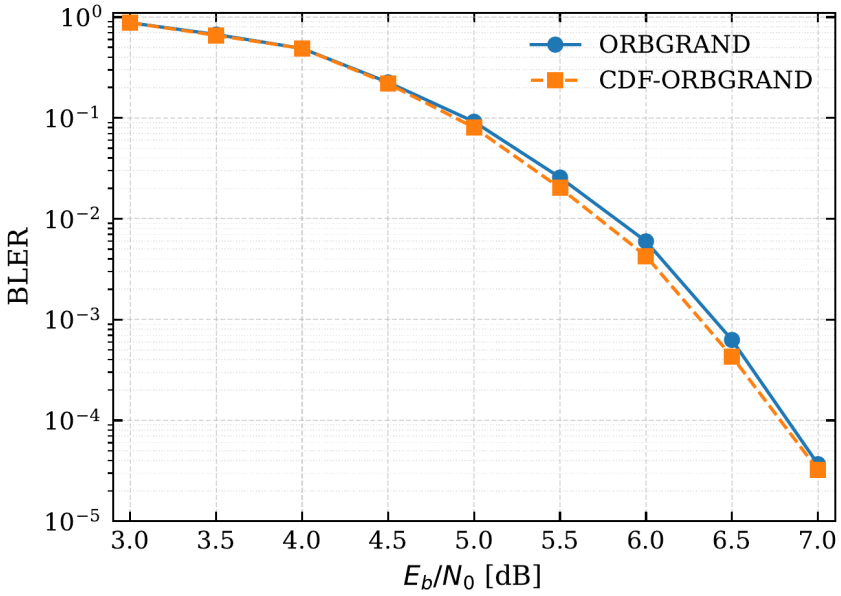


Figura 4.3: Confronto del BLER tra ORBGRAND e CDF-ORBGRAND al variare di E_b/N_0.

Il BLER diminuisce all'aumentare di E_b/N_0 per ambedue i decoder. A 3 dB e 4 dB i valori di BLER dei due decoder coincidono esattamente, mentre negli altri punti CDF-ORBGRAND mostra generalmente un BLER inferiore. Il vantaggio più visibile, tuttavia, si osserva nella regione intermedia. Infatti, tra 5 dB e 6.5 dB CDF-ORBGRAND presenta un numero maggiore di blocchi decodificati correttamente.

Ciò non implica, ovviamente, che CDF-ORBGRAND sia migliore per ogni blocco ma, dai risultati delle simulazioni, si evince che con una certa frequenza questa variante può anticipare alcuni pattern mentre ne posticipa altri, data la diversa metrica.

Il confronto aggregato mostra, infatti, un vantaggio medio in gran parte dei punti analizzati.

A 7 dB entrambi i BLER sono dell'ordine di 10^{-5}, quindi, affinché entrambi i decoder raggiungano almeno 100 errori, sono necessari più di tre milioni di blocchi.

4.3.2 Confronto del numero medio di query

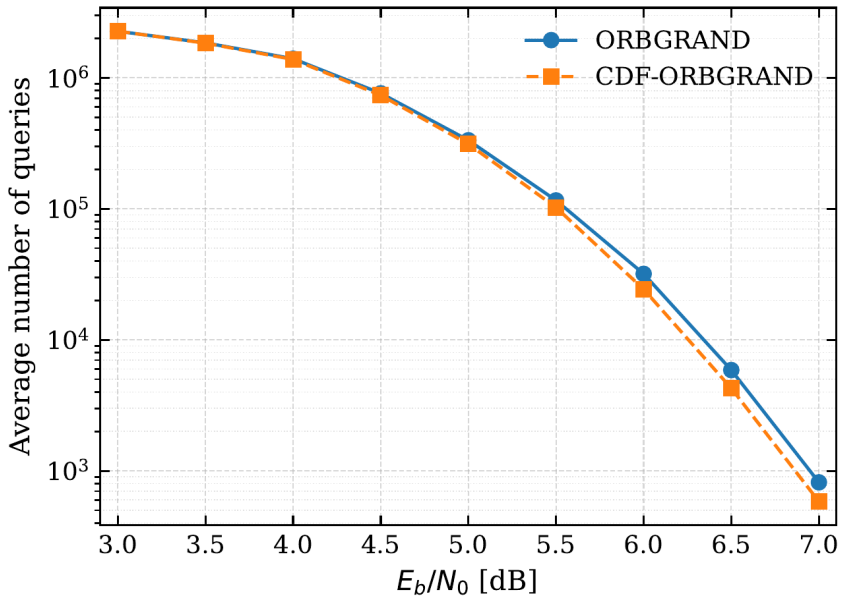


Figura 4.4: Confronto del numero medio di query per blocco tra ORBGRAND e CDF-ORBGRAND al variare di E_b/N_0.

Dal grafico si evince nettamente che il numero medio di query diminuisce fortemente all'aumentare di E_b/N_0. Infatti, a 3 dB, i due decoder risultano, in pratica, equivalenti e vicini al budget massimo di query, mentre, a partire dai punti successivi, CDF-ORBGRAND richiede un numero minore di query.

A 6 dB ORBGRAND utilizza circa $3.19 \cdot 10^4$ query per blocco, contro circa $2.44 \cdot 10^4$ di CDF-ORBGRAND. A 7 dB i valori scendono rispettivamente a circa 817.89 e

583.72. Essendo il controllo di sindrome identico, la differenza deriva dall'ordine con cui i pattern vengono proposti.

4.3.3 Analisi sui medesimi blocchi

Poiché i due decoder lavorano sulle stesse realizzazioni, è possibile contare i blocchi recuperati esclusivamente da CDF-ORBGRAND e quelli recuperati, analogamente, da ORBGRAND.

Tabella 4.1: Numero di blocchi recuperati esclusivamente da uno dei due decoder.

E_b/N_0 [dB]	RecCDF	RecORB
3.0	0	0
3.5	2	0
4.0	4	4
4.5	7	4
5.0	24	10
5.5	43	16
6.0	64	24
6.5	86	40
7.0	72	58

In Tabella 4.1, con RecCDF si indicano i blocchi decodificati correttamente da CDF-ORBGRAND per i quali ORBGRAND abbandona, mentre RecORB indica la situazione opposta.

Si può notare come nessuno dei due ordinamenti domini l'altro su ogni singola realizzazione; tuttavia, da 4.5 dB a 7 dB i blocchi recuperati da CDF-ORBGRAND sono più numerosi di quelli recuperati da ORBGRAND, in accordo con il vantaggio, in media, osservato sul BLER.

4.4 Sintesi dei risultati

Sono due i principali aspetti che risaltano da queste analisi. Il primo è che, nel caso hard, la riduzione della probabilità di crossover del BSC determina un progressivo miglioramento delle prestazioni di GRAND, sia in termini di BLER sia di numero medio di query necessarie alla decodifica. Il secondo è che, nel confronto soft, il rank companding di CDF-ORBGRAND è l'elemento che modifica l'ordine di guessing e dunque ha prodotto, nelle simulazioni effettuate, un vantaggio medio sia in termini di numero di query sia di BLER, per gran parte dell'intervallo di E_b/N_0.

Conclusioni

Il lavoro svolto ha avuto come obiettivo principale lo studio, l'analisi e l'implementazione degli algoritmi della famiglia *Guessing Random Additive Noise Decoding*, considerando GRAND nella formulazione hard-decision, ORBGRAND e la variante CDF-ORBGRAND.

L'attenzione è stata rivolta, anzitutto, agli aspetti teorici, alla base del lavoro, che caratterizzano i vari criteri di ordinamento delle ipotesi di rumore, per poi passare alla loro implementazione in linguaggio C.

Nel caso di GRAND con decisione hard, il decoder è stato analizzato sul Binary Symmetric Channel (BSC), in cui le ipotesi di rumore vengono ordinate in funzione del peso di Hamming. ORBGRAND, invece, introduce l'informazione soft proveniente dal canale AWGN e utilizza i ranghi di affidabilità dei bit per determinare l'ordine di guessing attraverso il Logistic Weight. Infine, CDF-ORBGRAND modifica ulteriormente tale ordinamento tramite il *rank companding*, associando ai ranghi un peso ottenuto a partire dalla funzione di ripartizione del modulo degli LLR.

Le simulazioni numeriche hanno permesso, successivamente, di verificare il comportamento dei decoder al variare dei principali parametri dei canali considerati, vale a dire la probabilità di crossover p per GRAND hard e il rapporto E_b/N_0 per ORBGRAND e CDF-ORBGRAND. Condizioni di canale progressivamente più favorevoli corrispondono, rispettivamente, alla diminuzione di p e all'aumento di E_b/N_0.

In particolare, sono stati analizzati il *Block Error Rate* e il numero medio di query. Per GRAND hard, la diminuzione della probabilità di crossover del BSC determina una graduale riduzione del BLER e della complessità media nelle condizioni di canale progressivamente più favorevoli considerate. Il confronto con i riferimenti teorici ha, inoltre, evidenziato una stretta corrispondenza tra il BLER simulato e la probabilità di abbandono P_{ab}, coerentemente con l'assenza di undetection nelle realizzazioni osservate. Il divario rispetto ai bound teorici RCU e converse mette, invece, in evidenza il ruolo determinante del limite massimo di query, che costituisce il principale vincolo alle prestazioni ottenute da GRAND hard nelle condizioni di simulazione considerate. Nel confronto tra ORBGRAND e CDF-ORBGRAND, eseguito sugli stessi blocchi e con il medesimo limite massimo di query, CDF-ORBGRAND ha mostrato un vantaggio medio per gran parte dell'intervallo di E_b/N_0 considerato, sia in termini di BLER sia in termini di numero medio di query per blocco.

L'analisi sui singoli blocchi ha, inoltre, evidenziato che nessuno dei due ordinamenti domina l'altro in ogni realizzazione, ma che il diverso criterio di generazione delle

ipotesi può portare CDF-ORBGRAND ad anticipare pattern che ORBGRAND raggiunge successivamente.

Pertanto, il lavoro mostra come la strategia con cui vengono ordinate le ipotesi di rumore rappresenti un elemento molto rilevante nella famiglia GRAND. Infatti, a parità di codice, canale e criterio di validità delle codeword, una diversa metrica di guessing può modificare, anche in maniera significativa, sia le prestazioni sia il numero di tentativi necessari alla decodifica.

L'implementazione completa degli algoritmi sviluppati nel corso del lavoro è disponibile nel repository GitHub GRAND-ORBGRAND-CDF-ORBGRAND.

Possibili sviluppi futuri possono riguardare l'estensione delle simulazioni ad altri codici e lunghezze di blocco, oltre a ulteriori ottimizzazioni dell'implementazione, volte a migliorarne l'efficienza in termini di costo computazionale e occupazione di memoria.

Bibliografia

[1] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[2] John G. Proakis and Masoud Salehi. *Communication Systems Engineering*. Prentice Hall, 2 edition, 2002. International Edition.

[3] Ken R. Duffy, Jiange Li, and Muriel Medard. Capacity-achieving guessing random additive noise decoding. *IEEE Transactions on Information Theory*, 65(7):4023–4040, 2019.

[4] Ken R. Duffy, Wenhao An, and Muriel Medard. Ordered reliability bits guessing random additive noise decoding. *IEEE Transactions on Signal Processing*, 70:4528–4542, 2022.

[5] Zhaoyang Li and Wenyi Zhang. Orbgrand is exactly capacity-achieving via rank companding. *IEEE Transactions on Communications*, 74:11550–11565, 2026.

[6] Robert G. Gallager. *Low Density Parity Check Codes*. PhD thesis, Massachusetts Institute of Technology, 1960.

[7] George E. P. Box and Mervin E. Muller. A note on the generation of random normal deviates. *The Annals of Mathematical Statistics*, 29(2):610–611, 1958.

[8] Yury Polyanskiy, H. Vincent Poor, and Sergio Verdú. Channel coding rate in the finite blocklength regime. *IEEE Transactions on Information Theory*, 56(5):2307–2359, 2010.