UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea Magistrale in Ingegneria Elettronica

Analisi della tail sequence in trasmissioni codificate, con applicazioni all'ambito satellitare

Tail sequence analysis in coded transmissions, with applications to satellite communications

Relatore:

Dr. Massimo Battaglioni

Tesi di Laurea di:

Rebecca Giuliani

Correlatore: Chiar.mo

Prof. Franco Chiaraluce

A.A. 2022 / 2023

Abstract

Questo elaborato si incentra sull'analisi delle prestazioni dello schema di comunicazione descritto nello standard CCSDS riguardo il Telecommand (TC) Space Data Link Protocol, nel caso in cui venga scelto il trigger di un errore nel processo di decodifica come approccio per il riconoscimento della *Tail Sequence*. Tale schema viene utilizzato nei collegamenti di comunicazione terra-spazio o spazio-spazio dalle missioni spaziali.
Il principale contributo del presente lavoro di tesi consiste nel dimostrare ed evidenziare che, seguendo le specifiche dello standard proposto dal CCSDS appena menzionato e usando il decoder del codice per la correzione d'errore per il riconoscimento della *Tail Sequence*, non vengono sfruttate le proprietà della sequenza proposta. In particolare, si evidenzia come randomizzare quest'ultima, in accordo con lo schema di trasmissione, porti tre diversi ma comunemente utilizzati decoder basati su algoritmi iterativi a confondere frequentemente la tail sequence con altre parole di codice (se ne sono individuate tre, nello specifico). In conclusione del lavoro, si è studiato un algoritmo per la ricerca della *Tail Sequence*, implementato come punto di partenza per la ricerca di un'eventuale nuova sequenza.

Indice

Elenco delle figure

Capitolo 1

Introduzione

Dal lancio del primo satellite artificiale, lo Sputnik 1 nel 1957, l'orbita terrestre è stata sempre più popolata da una miriade di artefatti spaziali. Attualmente, numerosi Paesi e società private hanno messo in orbita un gran numero di satelliti per una varietà di scopi, come le comunicazioni, l'osservazione della Terra, la navigazione e la scienza ([2]).
Visti gli svariati utilizzi dei satelliti, dal 1957 ad oggi ne sono stati mandati in orbita più di 11 000, di cui attualmente operativi ne rimangono poco più di 4 500 (si veda Figura 1.1).

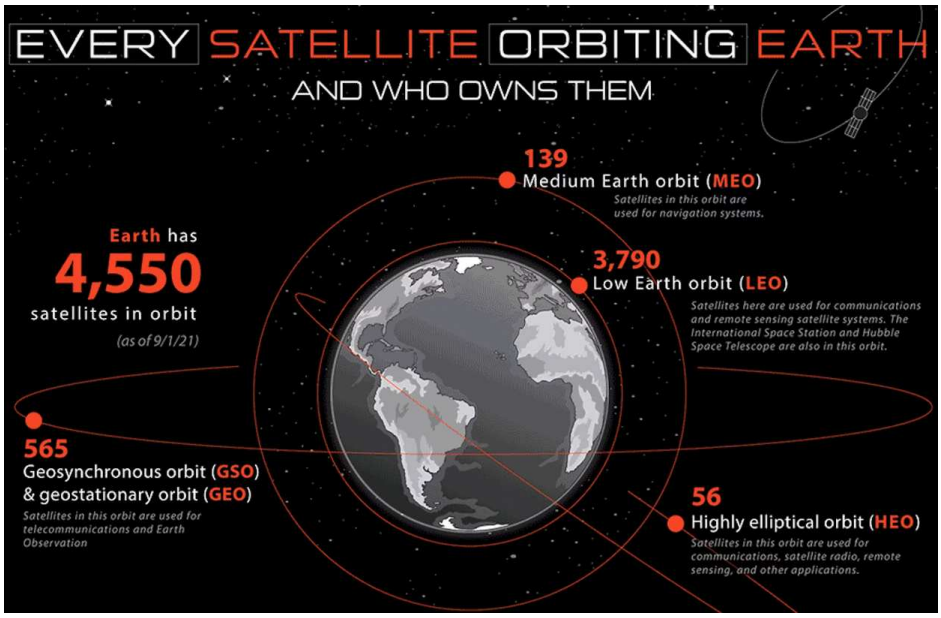

Figura 1.1: Satelliti in Orbita

Oltre a quelli già in orbita, è stato già pianificato il lancio di altrettanti satelliti negli anni a seguire (si veda Figura 1.2). Inoltre si può predire che molti altri ne verranno pianificati e lanciati, considerando i numerosissimi servizi che le comunicazioni satellitari possono fornire (vedi [3, 4]).

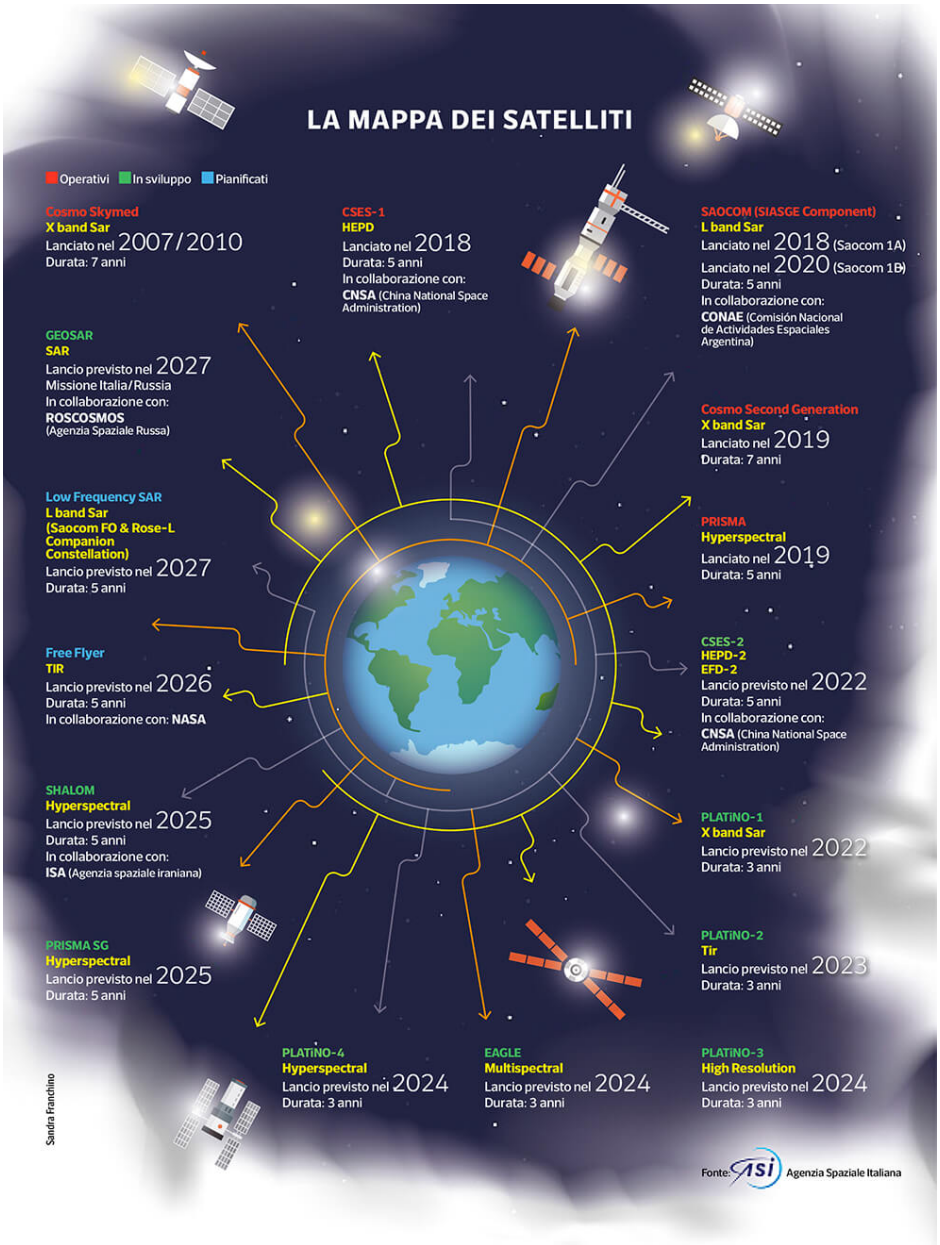

Figura 1.2: Mappa dei Satelliti

1.1 CCSDS

Il *Consultative Commitee for Space Data System* (CCSDS) è un forum multinazionale per lo sviluppo di standard di comunicazione e sistemi di dati per il volo spaziale. Vari esperti in comunicazioni spaziali collaborano per definire standard che permettano una gestione dei dati e delle comunicazioni sempre più efficienti (si veda [5]). Gli standard definiti dal CCSDS sono stati scelti per più di 1 000 missioni spaziali.

1.1.1 CLTU

Lo standard proposto dal CCSDS in [6, 7] propone come struttura della *Communication Link Transmission Unit* (CLTU) l'unione in sequenza della *Start Sequence*, dei dati codificati e della *Tail Sequence*, come mostrato in Figura 1.3.

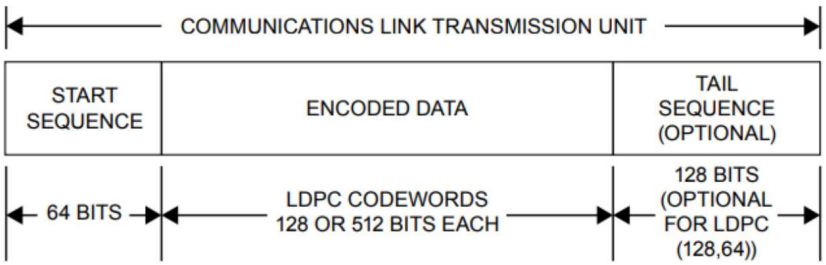

Figura 1.3: Struttura della Communication Link Transmission Unit

La *Start Sequence*, riportata in Figura 1.4, è una sequenza lunga 64 bit.

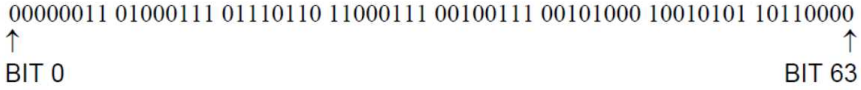

Figura 1.4: Start Sequence per Codifica LDPC (128,64)

La *Tail Sequence*, quando è prevista codifica a correzione d'errore tramite un codice LDPC $(128, 64)$, invece, è una sequenza di 128 bit, proposta in [8] e riportata in Figura 1.5.

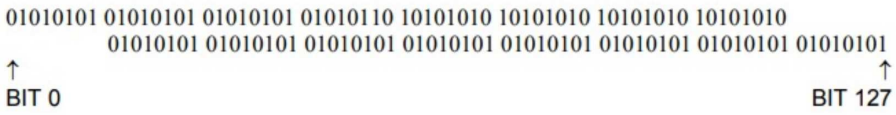

Figura 1.5: Tail Sequence per Codifica LDPC $(128, 64)$

1.1.2 Descrizione del Sistema di Telecomunicazioni

Il sistema di telecomunicazioni consigliato dal CCSDS è riportato in maniera semplificata (sono stati omessi modulatore e demodulatore) in Figura 1.6.

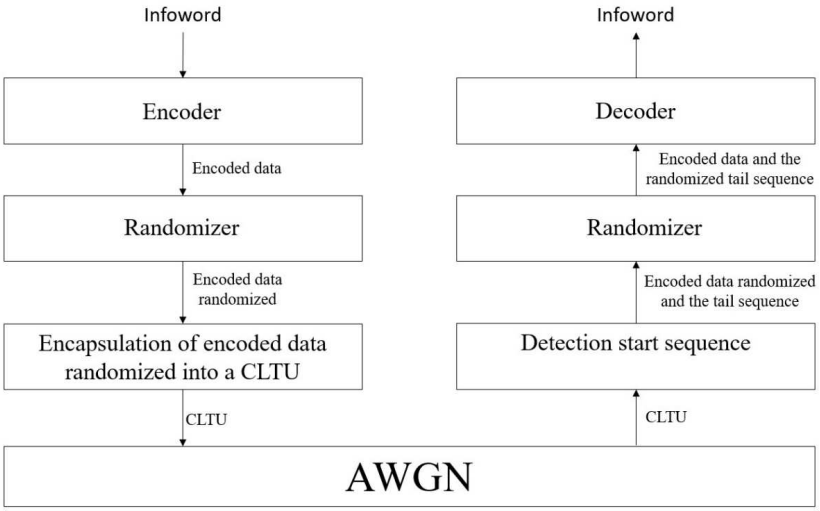

Figura 1.6: Schema Sistema di Comunicazione

Seguendo lo standard proposto, in trasmissione l'informazione viene codificata, poi randomizzata (il randomizer consigliato in [6] è raffigurato in Figura 1.7) ed infine incapsulata nella CLTU prima di essere inviata tramite il canale. Si noti che, in trasmissione, la Tail Sequence non viene randomizzata.

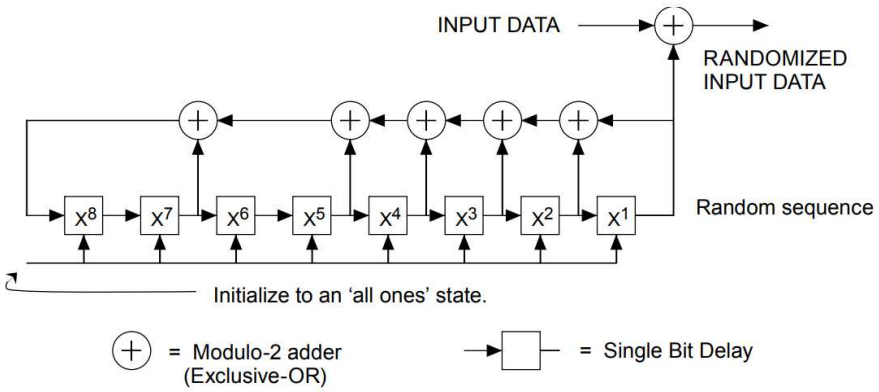

Figura 1.7: Randomizer

Come si può osservare in Figura 1.6, in ricezione, data la CLTU, innanzitutto viene ricercata la *Start Sequence*. Una volta identificata tale sequenza, i dati codificati vengono derandomizzati e l'output del randomizer, che si compone appunto di dati derandomizzati e della *Tail Sequence* randomizzata, viene dato in input al decoder.

Il decoder si occuperà quindi della decodifica dei dati codificati e, a meno dell'uso di approcci alternativi, del tentato riconoscimento della *Tail Sequence*.

1.1.3 Approcci per il riconoscimento della Tail Sequence

Per il riconoscimento della *Tail Sequence* esistono principalmente due approcci:

- l'impiego di un correlatore;
- il trigger di un errore nel processo di decodifica.

Il primo approccio permette di avere le prestazioni migliori, ma la sua implementazione è complessa e dispendiosa.

D'altro canto, il secondo approccio permette di risparmiare in termini di complessità, ma non permette di avere le stesse prestazioni del primo. In questo caso, è fondamentale progettare al meglio la *Tail Sequence*, come una sequenza che, quando arriva in input al decoder, inneschi l'interruzione della lettura. Questa sequenza dovrebbe essere il più possibile distante dalle parole di codice, cosicché il decoder non riesca a colmare il gap (in termini di distanza

di Hamming) tra la tail sequence ed esse.

È proprio su questo secondo scenario che si è incentrato il presente lavoro di tesi.

1.2 Obiettivi

Inizialmente è stato analizzato lo stato dell'arte.

In seguito, sono state studiate le prestazioni del decoder, quando esso riceve in input la *Tail Sequence* proposta in [8], randomizzata.

Dopodiché, sono state valutate le prestazioni dell'intero sistema. In particolare, è stato dimostrato che randomizzare la sequenza proposta in [8] non è una scelta consigliabile, poiché tale sequenza non era stata pensata per essere randomizzata e, randomizzandola, si perdono alcune caratteristiche peculiari ottenute in fase di progetto.

Lo standard proposto dal CCSDS non permette quindi di avere le ottime prestazioni previste con l'utilizzo della sequenza proposta in [8] come *Tail Sequence*, poiché essa non era stata pensata per essere randomizzata. Per questo motivo, sono state gettate le basi per il progetto di una nuova *Tail Sequence*, che fosse più performante di quella attualmente in utilizzo.

1.3 Sommario

Il lavoro svolto si può suddividere in tre diverse fasi, che sono state esposte in tre diversi capitoli:

- **Analisi Preliminare:** in questo capitolo sono state analizzate le prestazioni del sistema in termini di probabilità di rifiuto del TeleComando, e parte dei risultati ottenuti sono stati confrontati con quelli riportati in [1].
- **Analisi delle Occorrenze delle Parole di codice:** in questo capitolo sono stati analizzati i risultati delle simulazioni Monte Carlo per la stima delle prestazioni del sistema, ottenuti considerando svariati algoritmi di decodifica.
- **Ricerca della Tail Sequence:** in questo capitolo è stato studiato il procedimento seguito in [8] per definire la *Tail Sequence* attualmente

in utilizzo ed è stato sviluppato del codice, come base per l'eventuale progetto di una nuova Tail Sequence.

- **Conclusioni:** in questo capitolo sono state esposte le conclusioni del lavoro di tesi.

Capitolo 2

Analisi delle Prestazioni del Sistema

La prima parte di questo elaborato contiene un'analisi delle prestazioni del sistema descritto in Figura 1.6, e la riproduzione e rielaborazione di alcuni risultati d'interesse contenuti in [1]. Il focus dell'analisi è stato posto sulla *Tail Sequence Randomizzata*, in accordo con lo schema considerato.

2.1 Probabilità di Rifiuto del TeleComando

Una quantità fondamentale per l'analisi delle prestazioni del sistema è la *Probabilità di Rifiuto del TC* (TeleCommand). Questa definisce la probabiltà che un'unità ricevuta venga scartata e questo può avvenire per tre motivi: se la *Start Sequence* non viene riconosciuta, se la terminazione della CLTU non viene identificata o se il blocco di codice LDPC viene rifiutato.

La P_{TCrej} si calcola come

$$P_{TCrej} = P_{term} + (1 - Pterm) \cdot [P_{md} + (1 - P_{md}) \cdot P_{LDPC}] \tag{2.1}$$

dalla quale risaltano le tre probabilità che la compongono, quali:

- La P_{md}, *Probabilità di Missed Detection*, ossia la probabilità che la *Start Sequence* non venga riconosciuta, viene calcolata utilizzando (2.2) ([1, Eq. (7.1)]), dove P_{b} rappresenta la probabilità di errore sul bit.

$$P_{md} = 1 - (1 - P_b)^{64} - \sum_{n=1}^{7} \binom{64}{n} \cdot P_b^n \cdot (1 - P_b)^{64-n} \tag{2.2}$$

- La P_{term}, *Probabilità di Terminazione Non Riconosciuta*, è il rapporto tra il numero dei successi nella decodifica quando era stata trasmessa la tail sequence, contro il numero totale di decodifiche (si veda (2.3)).

$$P_{term} = \frac{Decoding_{Success}}{Decoding_{Total}} \tag{2.3}$$

- La P_{LDPC} è la *Probabilità di Errore sulla Codeword* (*CER*), relativa al codice LDPC (128,64).

2.2 Prestazioni del sistema

Considerando quanto spiegato nella sezione precedente, volendo definire le curve della *Probabilità di Rifiuto del TeleComando* al variare dei decoder, sono stati visionati anche gli andamenti delle probabilità che la compongono.

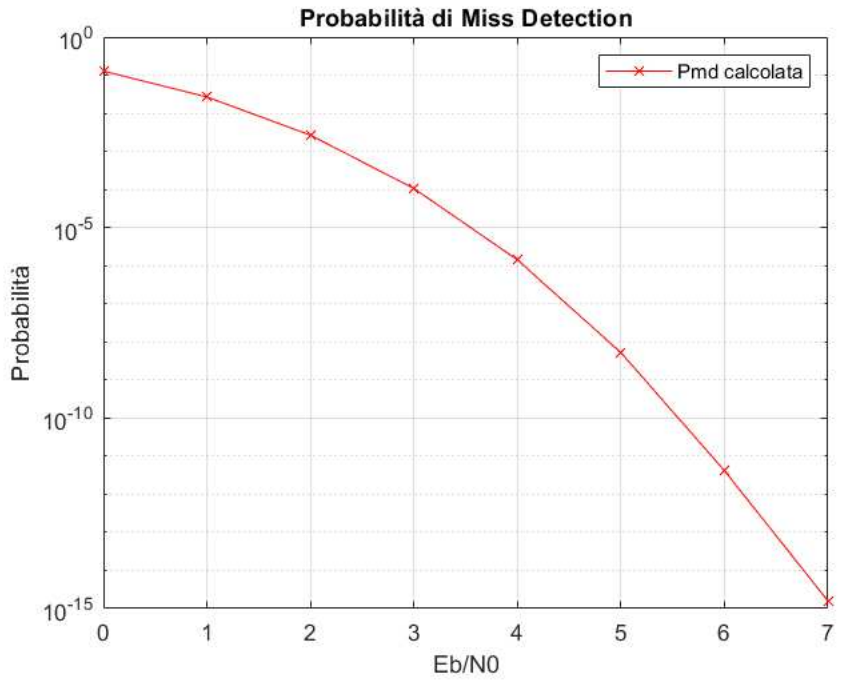

Figura 2.1: P_{md} - Probabilità di Miss Detection

L'andamento di P_{md}, calcolato con l'equazione (2.2), è riportato in Figura 2.1.

L'andamento di P_{term}, invece, cambia, seppur di poco, al variare del decoder, ed è osservabile nei grafici che riportano le curve che ricompongono la P_{TCrej} (si vedano le Figure 2.4, 2.6 e 2.8). Questa probabilità è definita dal rapporto del numero di successi del decoder e il numero totale di decodifiche, e si calcola con l'equazione (2.3), i cui valori in ingresso sono stati estratti dai risultati delle simulazioni.

L'andamento di P_{LDPC} è legato allo studio delle prestazioni, in termini di probabilità d'errore, del codice LDPC $(128, 64)$. Per valutarla, sono state effettuate simulazioni Monte Carlo, considerando tre decoder diversi: il Sum Product Algorithm Logarithm Likelihood Ratio (SPA-LLR), l'algoritmo MinSum (MS) e l'algoritmo Normalized MinSum (NMS). Come ulteriori parametri per l'interruzione della decodifica, sono stati fissati a 100 sia il *numero massimo di iterazioni* svolte dal decoder sia il *numero massimo di errori sul frame (FEmax)* che si attendono, per ogni E_b/N_0, prima di interrompere la simulazione. Come output delle simulazioni, al variare di E_b/N_0, sono state ottenute tre curve di Codeword Error Rate (CER), mostrate in Figura 2.2.

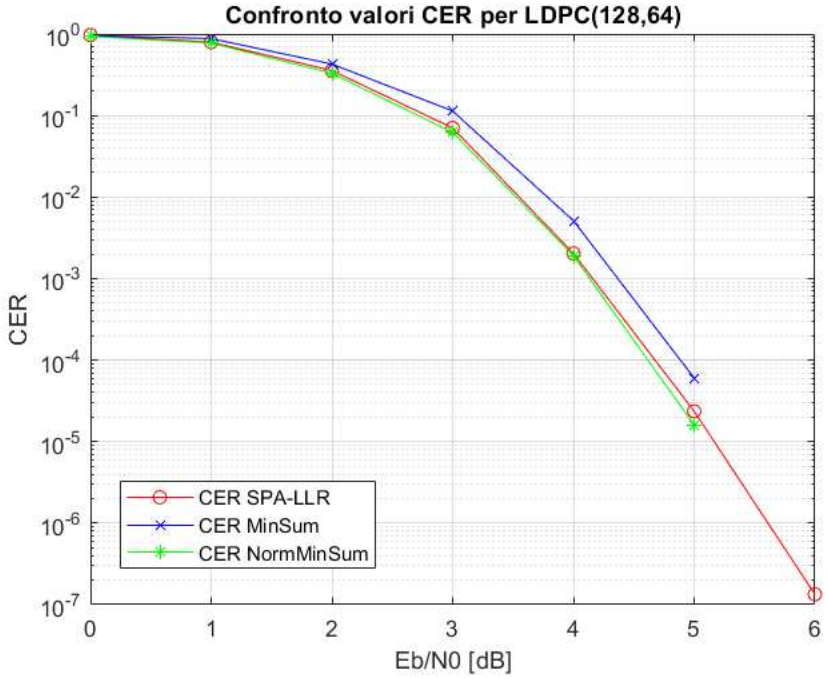

Figura 2.2: Confronto CER del codice LDPC (128,64), ottenute utilizzando tre diversi algoritmi di decodifica

Essendo l'andamento delle curve riportato in Fugura 2.2 quello atteso, è stato possibile affermare che le prestazioni del codice LDPC $(128, 64)$ fossero accettabili per la prosecuzione dell'analisi.

Come è stato fatto per la verifica delle prestazioni del codice LDPC, anche per la valutazione di P_{term} le simulazioni Monte Carlo sono state lanciate utilizzando rispettivamente l'SPA-LLR, l'algoritmo MS e il NMS.
In questo caso, per avere una maggiore affidabilità rispetto ai casi convenzionali, come anche in [1], il *numero massimo di iterazion*i e il *numero massimo di errori sul frame (FEmax)* prima di interrompere la simulazione relativa a un certo $\frac{E_b}{N_0}$ sono stati impostati rispettivamente a 150 e a 3000000.

Gli output di queste simulazioni, insieme ai valori mostrati nelle Figure 2.1 e 2.2, sono stati dati in input all'equazione che calcola la *Probabilità di Rifiuto del TC* (eq. (2.1)).
I risultati ottenuti sono anche stati utilizzati per generare i grafici che permettono di visionare l'andamento delle probabilità che compongono la P_{TCrej} al variare del tipo decoder (si vedano le Figure 2.4, 2.6, 2.8).

Queste figure sono state poi confrontate con quelle riportate in [1] (si vedano le Figure 2.3, 2.5, 2.7).
Si rimarca che l'analisi nel presente lavoro è stata svolta ragionando esclusivamente sul caso della *Tail Sequence Randomizzata*, in accordo con la Figura 1.6.
Per garantire la chiarezza delle figure estratte da [1], si spiega il significato delle quantità in legenda:

- $P_{md}start$ corrisponde alla P_{md}
- *SPA-LLR / MS / NMS LDPC*$(128, 64)$ corrispondono alla P_{LDPC} per i tre diversi decoder
- P_r *tail randomized SPA-LLR / MS / NMS* corrispondono alla P_{TCrej} per i tre diversi decoder
- P_r *tail not randomized SPA-LLR / MS / NMS* non hanno corrispondenti in questa tesi, poiché tali quantità non sono state considerate
- P_r *idle not randomized SPA-LLR / MS / NMS* non hanno corrispondenti in questa tesi, poiché tali quantità non sono state considerate
- P_r *idle randomized SPA-LLR / MS / NMS* non hanno corrispondenti in questa tesi, poiché tali quantità non sono state considerate

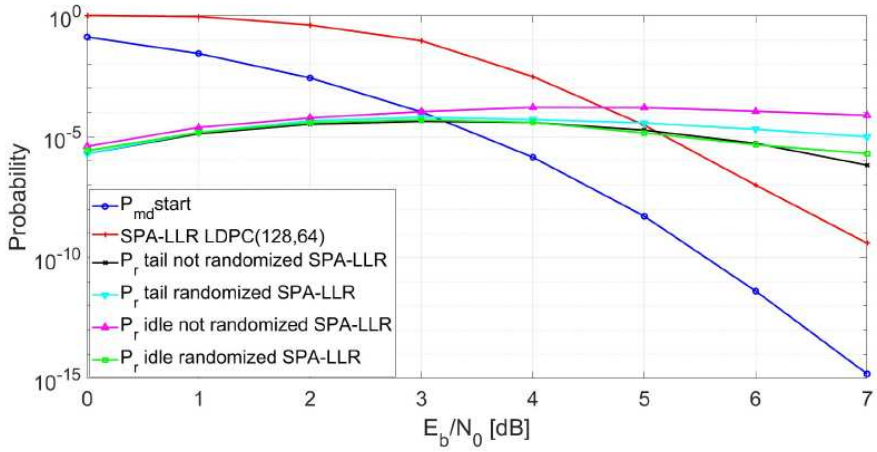

Figura 2.3: Scomposizione della Probabilità di Rifiuto del TC con decoder SPA-LLR, da [1]

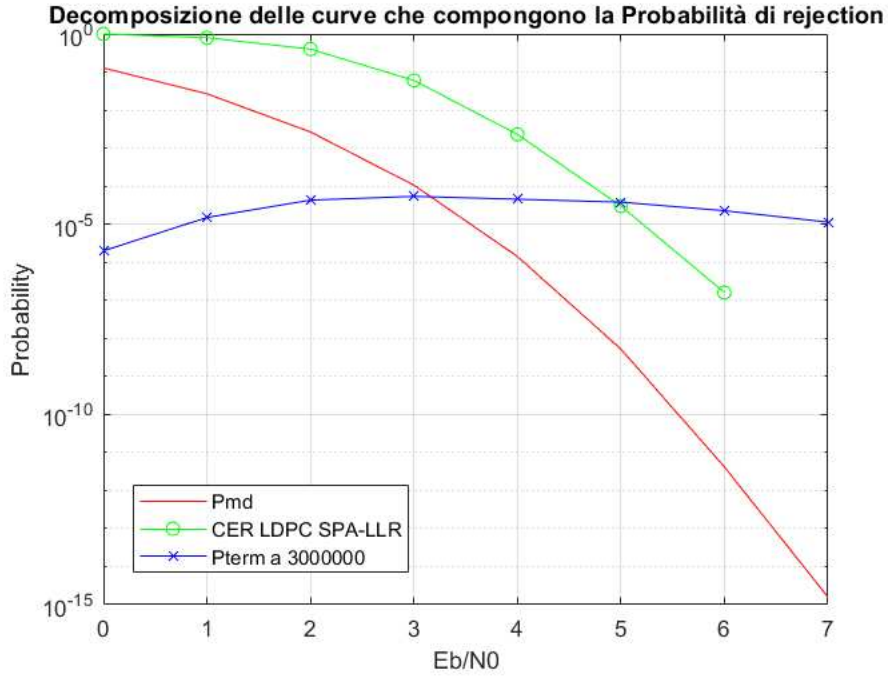

Figura 2.4: Scomposizione della Probabilità di Rifiuto del TC con decoder SPA-LLR

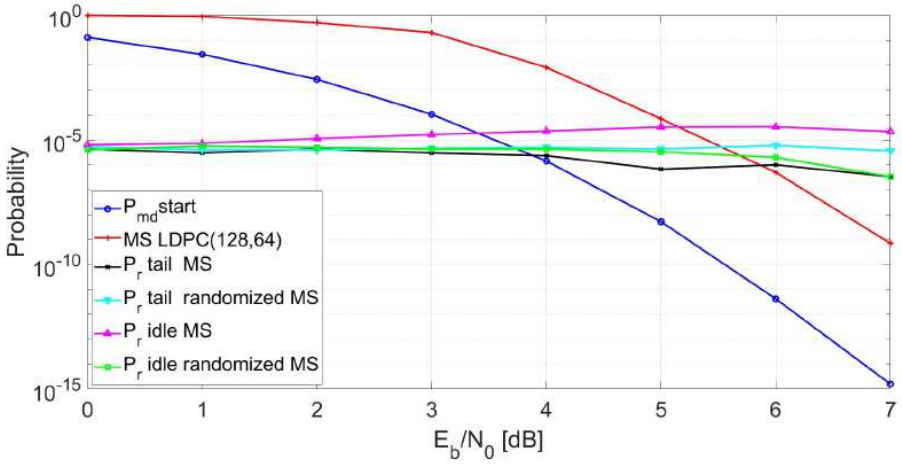

Figura 2.5: Scomposizione della Probabilità di Rifiuto del TC con decoder MinSum, da [1]

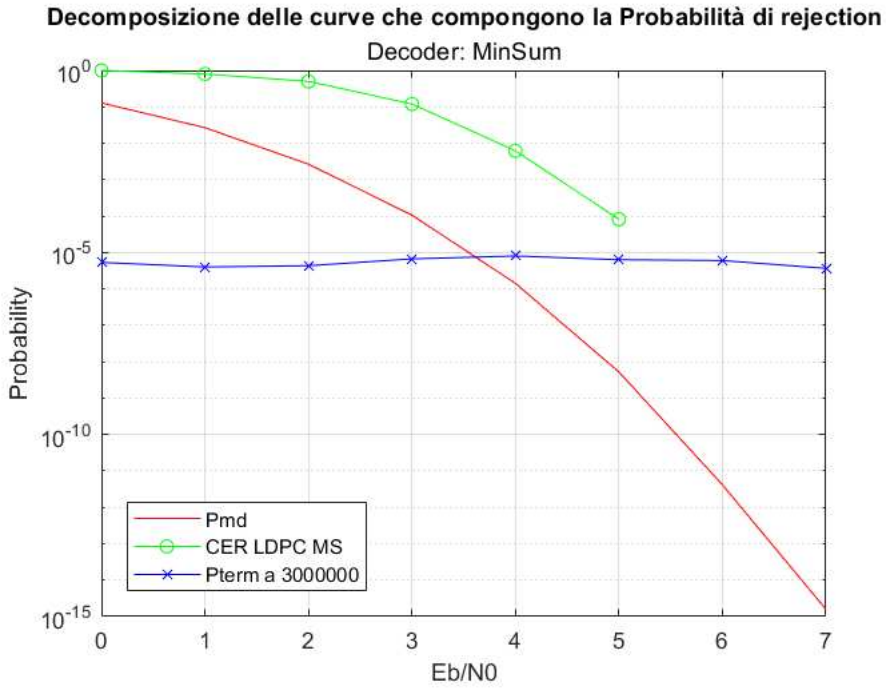

Figura 2.6: Scomposizione della Probabilità di Rifiuto del TC con decoder MinSum

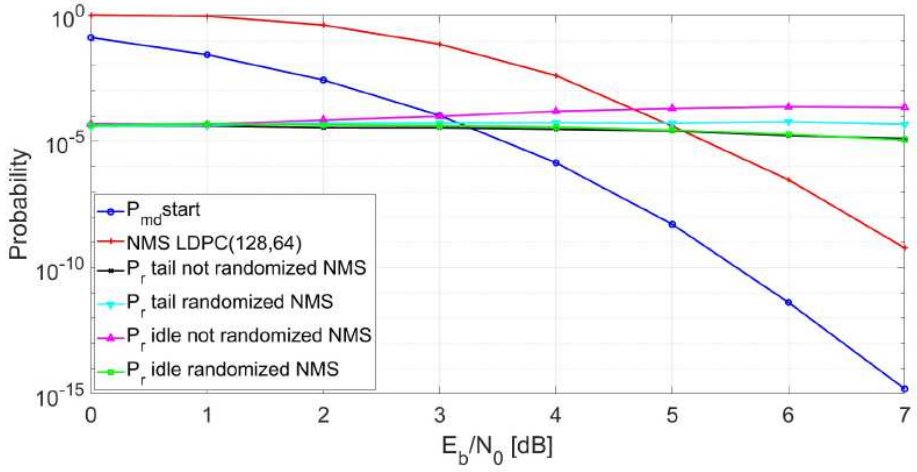

Figura 2.7: Scomposizione della Probabilità di Rifiuto del TC con decoder Normalised MinSum, da [1]

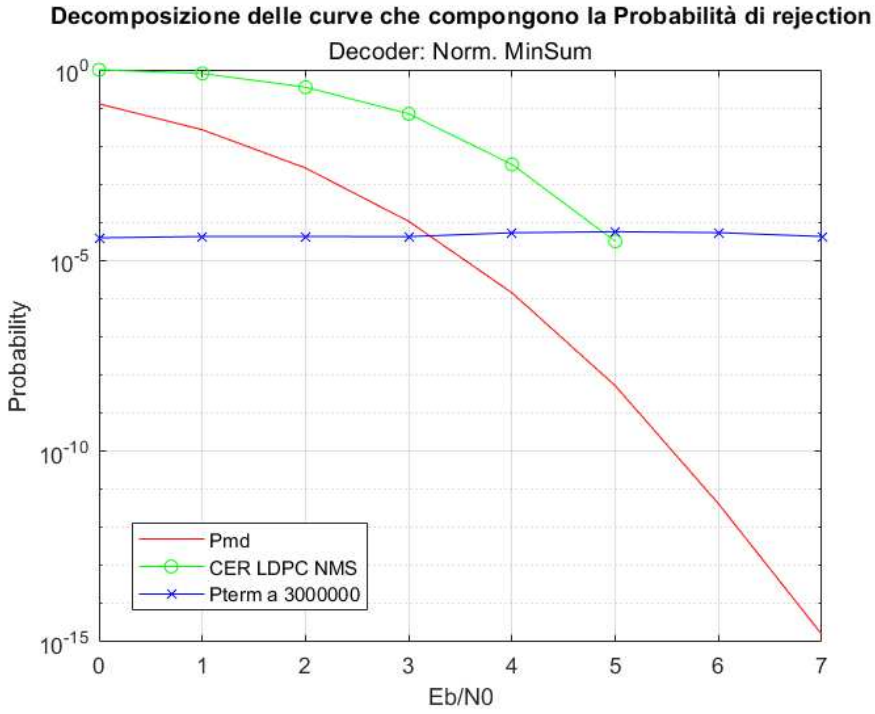

Figura 2.8: Scomposizione della Probabilità di Rifiuto del TC con decoder Normalised MinSum

Esclusa la differenza in numero delle curve la cui motivazione è stata spiegata

in precedenza, i grafici generati sono paragonabili a quelli ottenuti in [1].

Per maggiore chiarezza, con i risultati ottenuti dal calcolo della *Probabilità di Rifiuto del TC*, effettuato utilizzando l'equazione (2.1), è stato generato un grafico che raffigurasse le curve corrispondenti ai tre diversi decoder (si veda Figura 2.9), da cui si evince che il decoder MinSum è quello che dimostra di avere le prestazioni migliori, seguito dal SPA-LLR e dal Normalised MinSum. Convenzionalmente, in corrispondenza del punto di lavoro, ossia per un $\frac{E_b}{N_0} = 6$ dB, la *Probabilità di Rifiuto del TC* dovrebbe essere inferiore a 10^{-5}. Tuttavia, come possiamo osservare in Figura 2.9, fatta eccezione per il decoder MinSum, con entrambi gli altri due decoder ci troviamo al di sopra di tale soglia.

Per questo motivo si è deciso di proseguire l'analisi e studiare le occorrenze delle singole parole di codice nei file di decodifica, supponendo che la tail sequence utilizzata non sia ottimale.

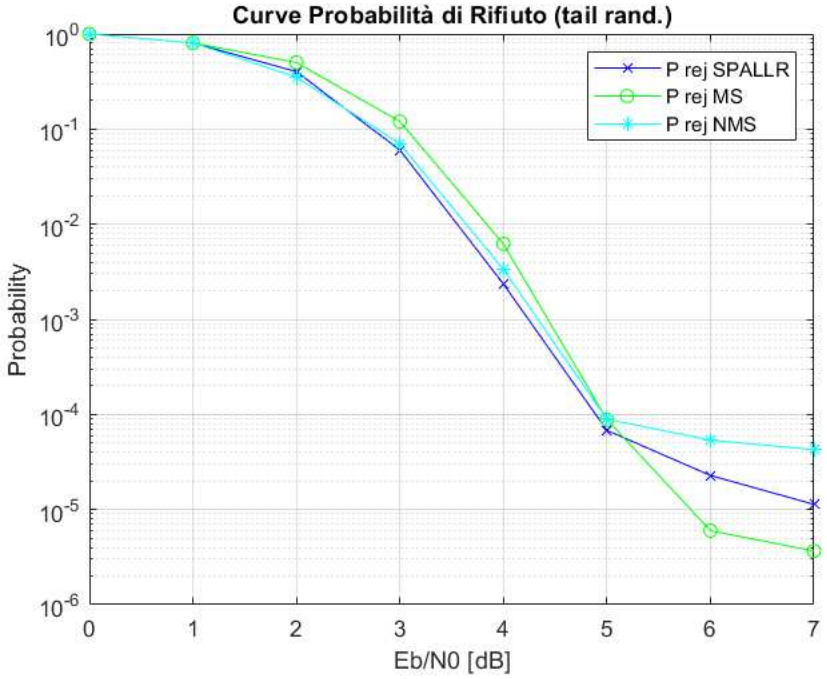

Figura 2.9: Curve Probabilità di Rifiuto del TC

Capitolo 3

Analisi delle Occorrenze delle Parole di Codice

Nel corso delle simulazioni svolte, le parole ricevute sono state salvate in un file di testo, dividendole per valore di $\frac{E_b}{N_0}$ (vedi Figura 3.1).
Con *parole ricevute* si intendono tutte le parole di codice che il decoder ha restituito in output, associandole alla *Tail Sequence Randomizzata* (e quindi non riconoscendola), che era stata in realtà trasmessa.

Figura 3.1: Estratto del File di Testo dove sono state salvate le parole ricevute

In primis si è verificato che le parole ricevute fossero effettivamente parole di codice. È stata calcolata la sindrome di ogni parola, ovvero, ogni parola è stata moltiplicata con la trasposta della matrice di parità caratteristica

del codice LDPC$(128, 64)$, verificando poi che il vettore risultante fosse un vettore nullo.
A questo punto, come mostrato in seguito, si è osservato quante volte una stessa parola di codice venisse ottenuta dal decodificatore, tramite un conteggio delle occorrenze delle singole parole.

3.1 Occorrenze delle Parole di Codice per Tail Sequence Randomizzata

In questa sezione sono stati analizzati e commentati i risultati delle simulazioni effettutate per il calcolo della P_{TCrej}, (si veda la Sezione 2.3 del Capitolo 2). Questi hanno permesso di studiare il caso della *Tail Sequence Randomizzata* al variare del decoder utilizzato.

3.1.1 Occorrenze delle parole di codice - Decoder SPA-LLR

In questo primo caso in esame sono state riconosciute 385 parole di codice differenti, su un totale di 628 parole ricevute. Le loro occorrenze, considerando tutti i valori di $\frac{E_b}{N_0}$, si possono osservare in Figura 3.2.
Considerando, invece, i singoli valori di $\frac{E_b}{N_0}$, si nota che queste 385 parole sono risultate uniformemente distribuite per $\frac{E_b}{N_0}$ bassi, ad esempio pari a 0 dB e 1 dB (si vedano , le Figure 3.3 e 3.4).
Risultano invece distribuite in modo più disomogeneo per valori di $\frac{E_b}{N_0}$ più alti (si vedano le Figure 3.5, 3.6, 3.7, 3.8, 3.9 e 3.10).

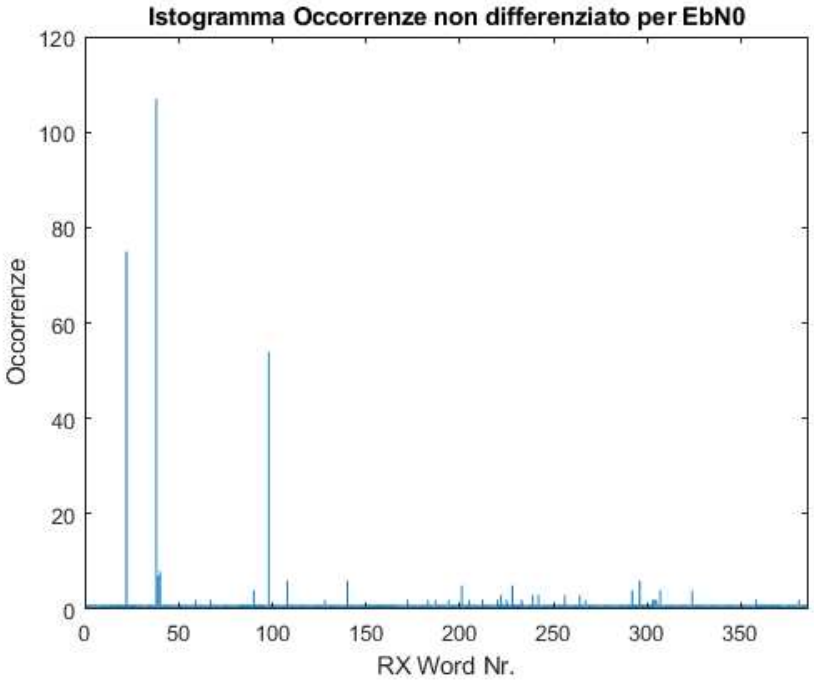

Figura 3.2: Occorenze non differenziate per $\frac{E_b}{N_0}$ - Decoder SPA-LLR, Tail Sequence Randomizzata

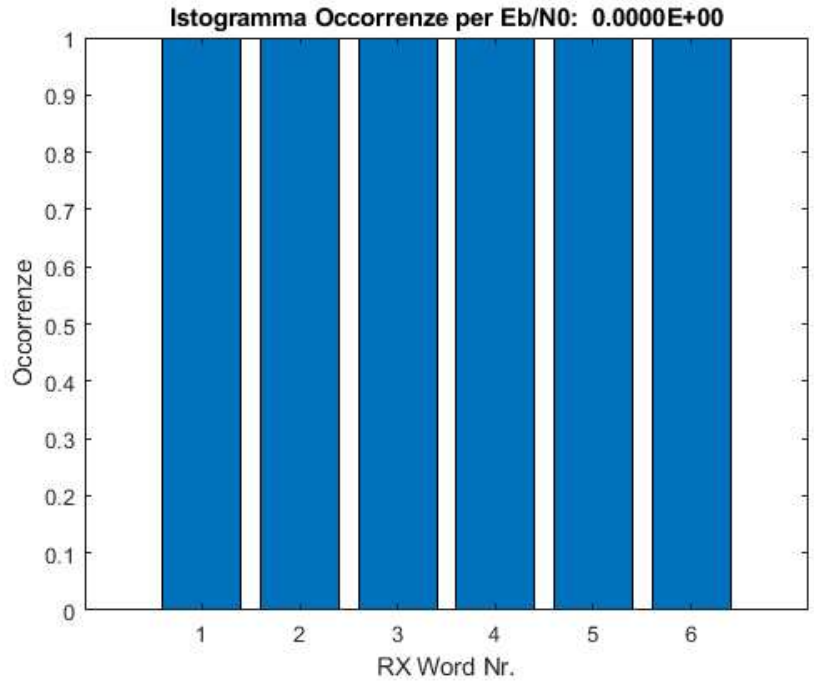

Figura 3.3: Occorrenze per $\frac{E_b}{N_0} = 0$ dB - Decoder SPA-LLR, Tail Sequence Randomizzata

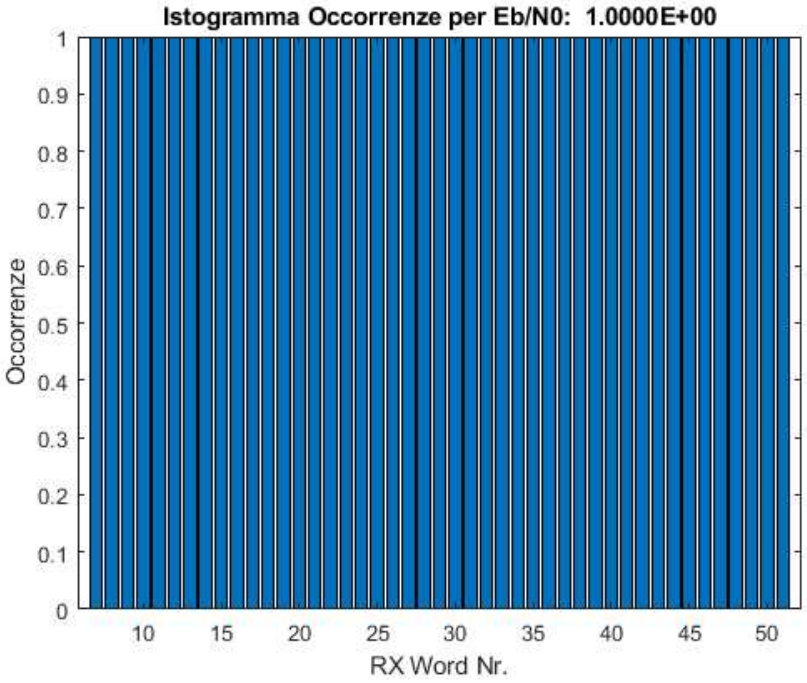

Figura 3.4: Occorrenze per $\frac{E_b}{N_0} = 1$ dB - Decoder SPA-LLR, Tail Sequence Randomizzata

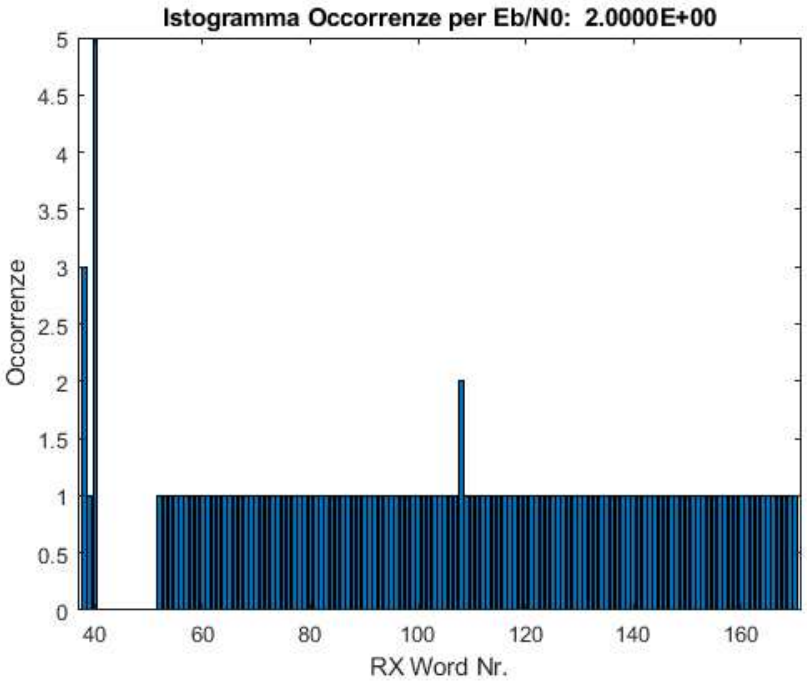

Figura 3.5: Occorrenze per $\frac{E_b}{N_0} = 2$ dB - Decoder SPA-LLR, Tail Sequence Randomizzata

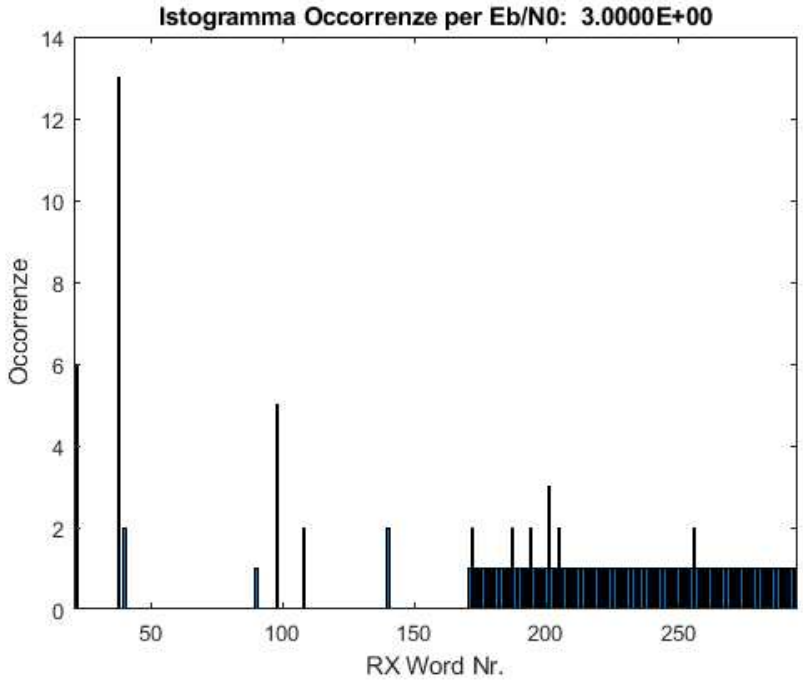

Figura 3.6: Occorrenze per $\frac{E_b}{N_0} = 3$ dB - Decoder SPA-LLR, Tail Sequence Randomizzata

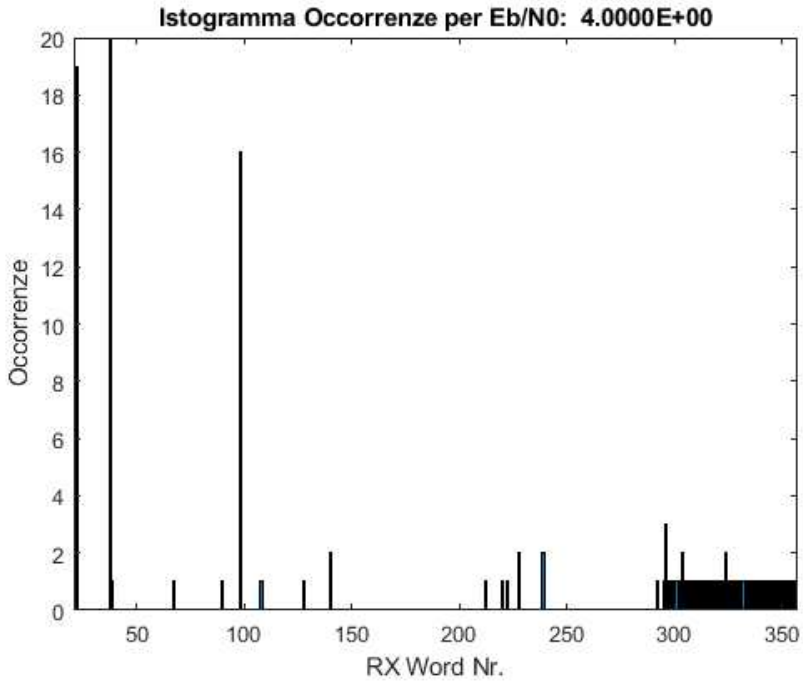

Figura 3.7: Occorrenze per $\frac{E_b}{N_0} = 4$ dB - Decoder SPA-LLR, Tail Sequence Randomizzata

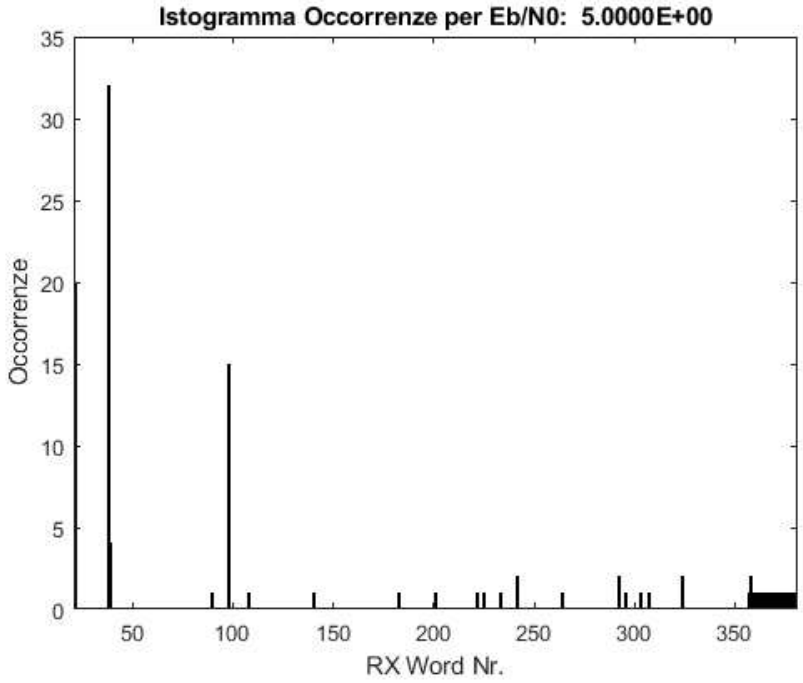

Figura 3.8: Occorrenze per $\frac{E_b}{N_0} = 5$ dB - Decoder SPA-LLR, Tail Sequence Randomizzata

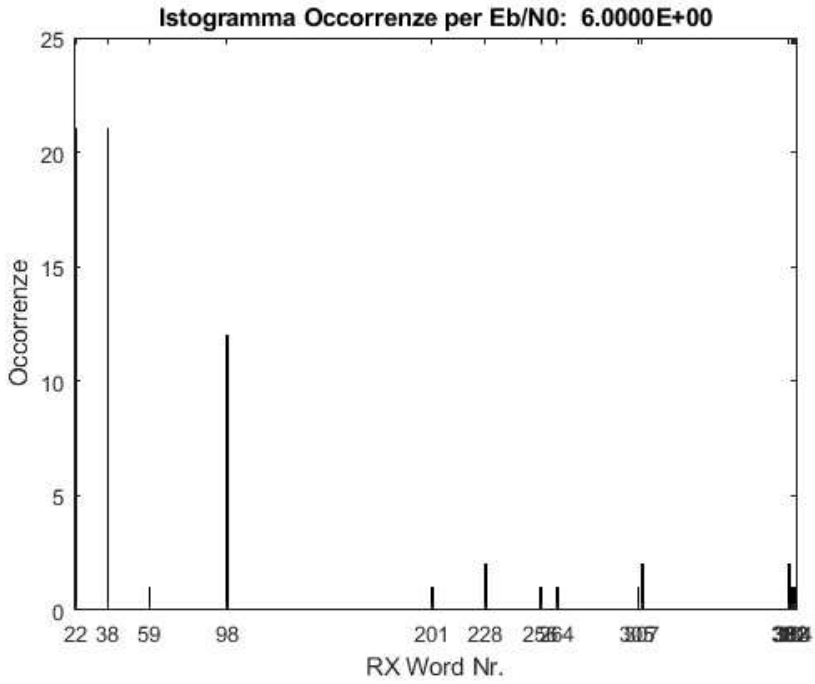

Figura 3.9: Occorrenze per $\frac{E_b}{N_0} = 6$ dB - Decoder SPA-LLR, Tail Sequence Randomizzata

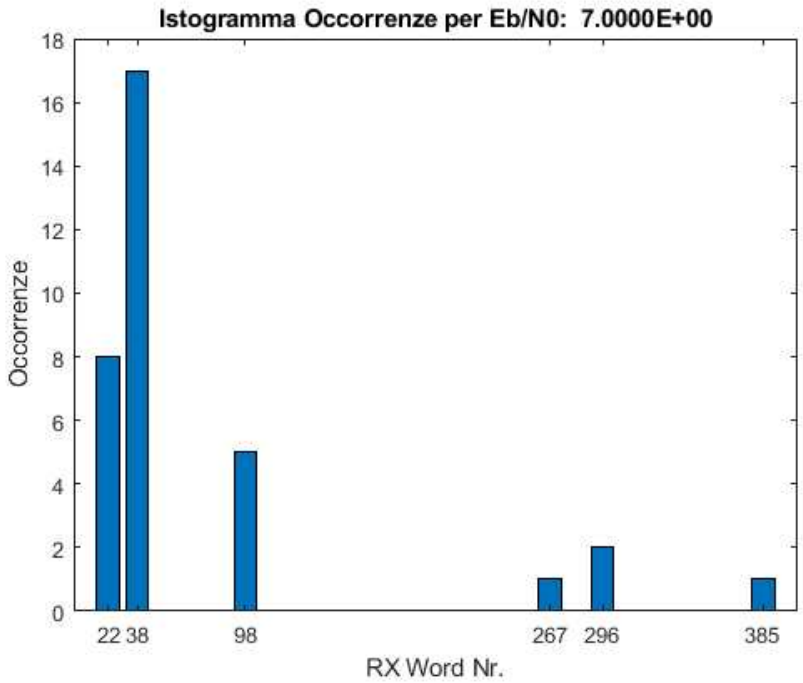

Figura 3.10: Occorrenze per $\frac{E_b}{N_0} = 7$ dB - Decoder SPA-LLR, Tail Sequence Randomizzata

3.1.2 Occorrenze delle Parole di Codice - Decoder MinSum

In questo secondo caso in esame sono state riconosciute 74 parole di codice diverse, sulle 133 parole ricevute. Le loro occorrenze senza considerare il variare di $\frac{E_b}{N_0}$ si possono osservare in Figura 3.11.
Dall'analisi puntuale, invece, queste 74 parole sono risultate uniformemente distribuite per $\frac{E_b}{N_0}$ bassi, ad esempio pari a 0 dB, 1 dB e 2 dB (si vedano le Figure 3.12, 3.13 e 3.14).
Risultano invece distribuite in modo più disomogeneo per valori di $\frac{E_b}{N_0}$ più alti (vedi Figure 3.15, 3.16, 3.17, 3.18 e 3.19).

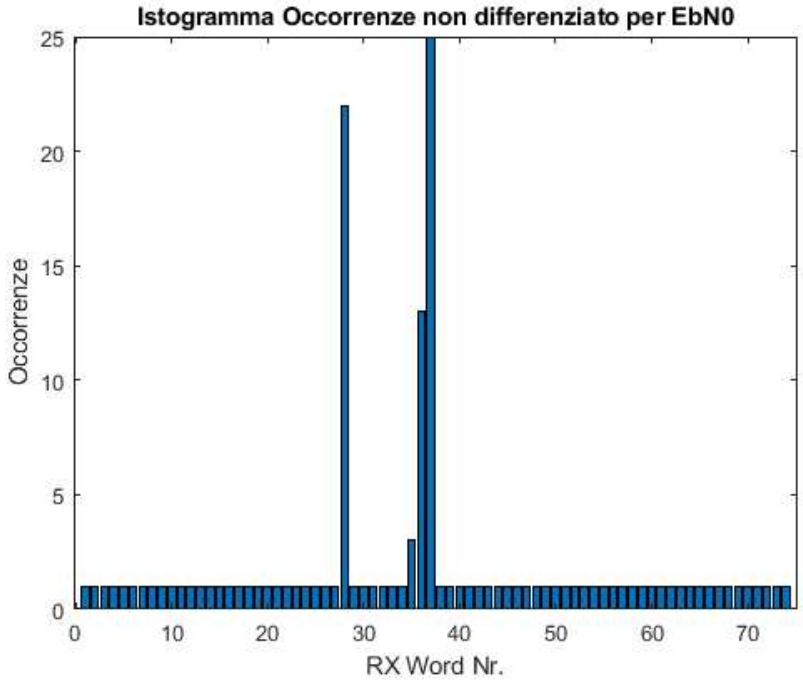

Figura 3.11: Occorenze non differenziate per $\frac{E_b}{N_0}$ - Decoder MinSum, Tail Sequence Randomizzata

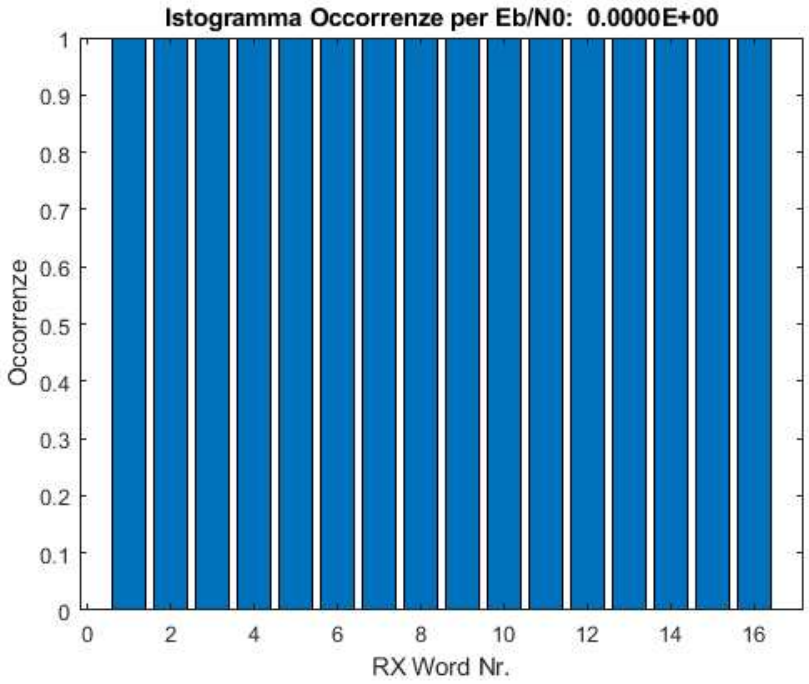

Figura 3.12: Occorrenze per $\frac{E_b}{N_0} = 0$ dB - Decoder MinSum, Tail Sequence Randomizzata

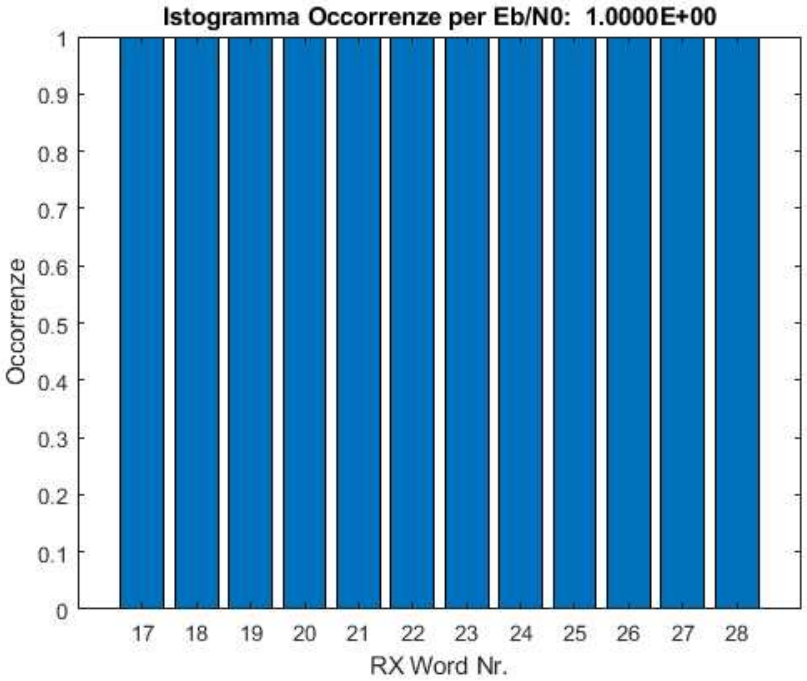

Figura 3.13: Occorrenze per $\frac{E_b}{N_0} = 1$ dB - Decoder MinSum, Tail Sequence Randomizzata

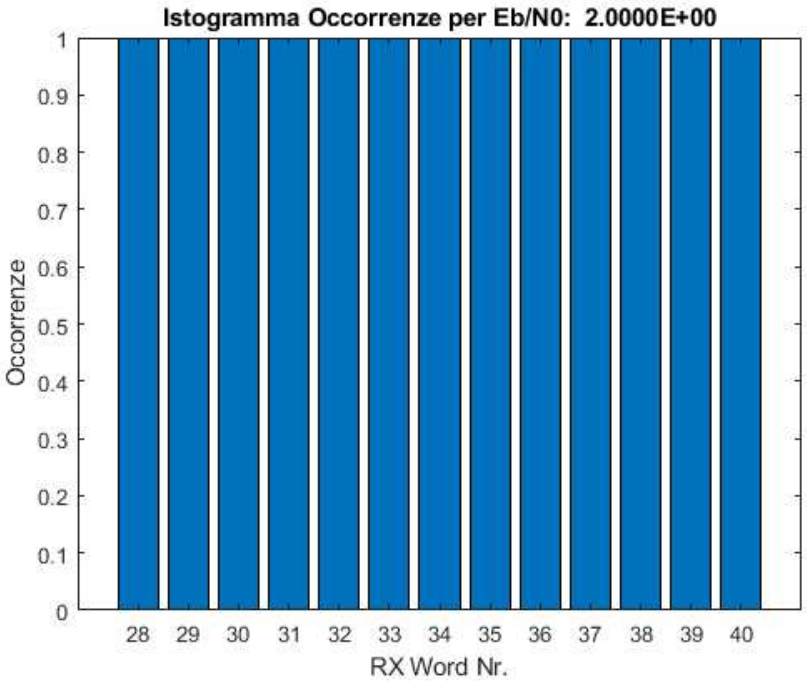

Figura 3.14: Occorrenze per $\frac{E_b}{N_0} = 2$ dB - Decoder MinSum, Tail Sequence Randomizzata

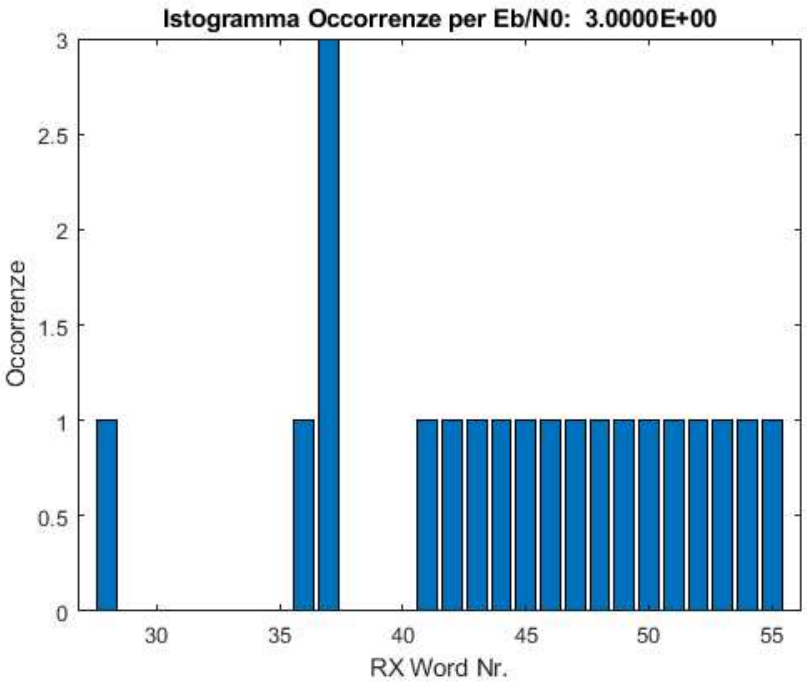

Figura 3.15: Occorrenze per $\frac{E_b}{N_0} = 3$ dB - Decoder MinSum, Tail Sequence Randomizzata

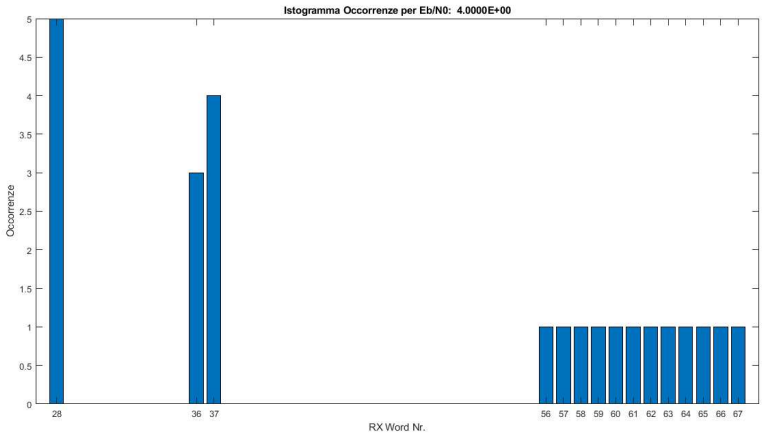

Figura 3.16: Occorrenze per $\frac{E_b}{N_0} = 4$ dB - Decoder MinSum, Tail Sequence Randomizzata

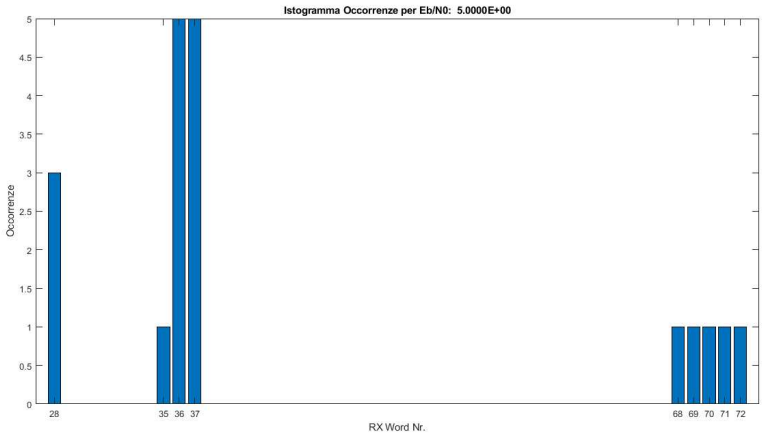

Figura 3.17: Occorrenze per $\frac{E_b}{N_0} = 5$ dB - Decoder MinSum, Tail Sequence Randomizzata

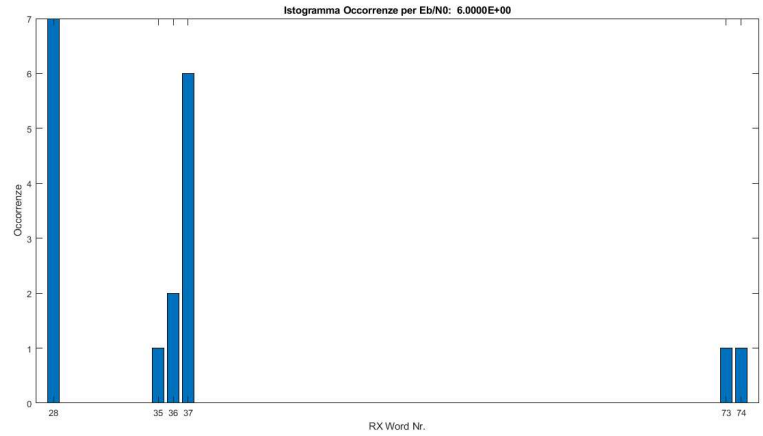

Figura 3.18: Occorrenze per $\frac{E_b}{N_0} = 6$ dB - Decoder MinSum, Tail Sequence Randomizzata

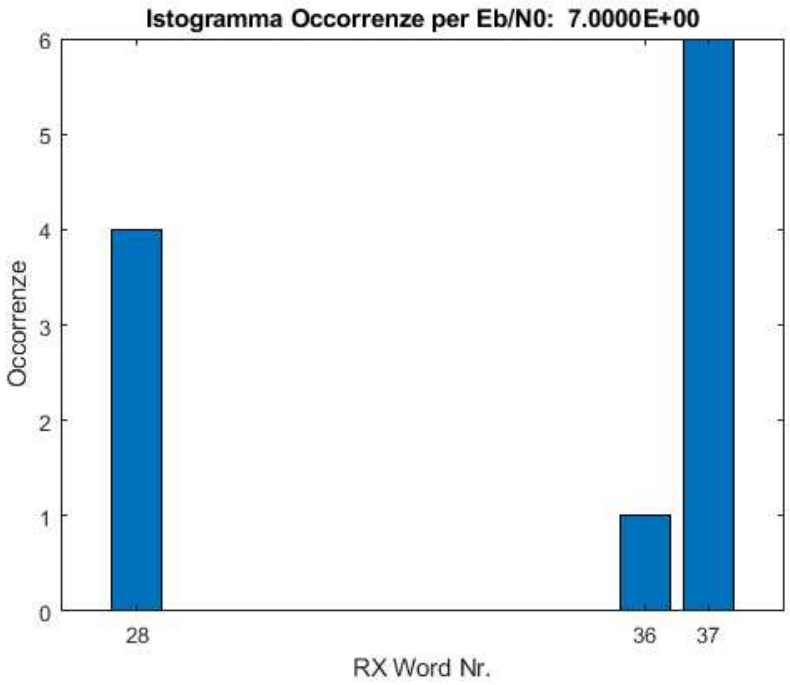

Figura 3.19: Occorrenze per $\frac{E_b}{N_0} = 7$ dB - Decoder MinSum, Tail Sequence Randomizzata

3.1.3 Occorrenze delle Parole di Codice - Decoder Normalised MinSum

In questo terzo e ultimo caso in esame, per quanto riguarda l'analisi dei riconoscimento della *Tail Sequence Randomizzata*, sono state trovate 585 parole di codice diverse su un totale di 1121 parole ricevute. Le loro occorrenze senza considerare il variare di $\frac{E_b}{N_0}$ si possono osservare in Figura 3.20.

Queste 585 parole sono risultate uniformemente distribuite per $\frac{E_b}{N_0}$ bassi, ad esempio pari a 0 dB, 1 dB e 2 dB (si vedano le Figure 3.21, 3.22 e 3.23). Risultano invece distribuite in modo più disomogeneo per valori di $\frac{E_b}{N_0}$ più alti (si vedano le Figure 3.24, 3.25, 3.26, 3.27 e 3.28).

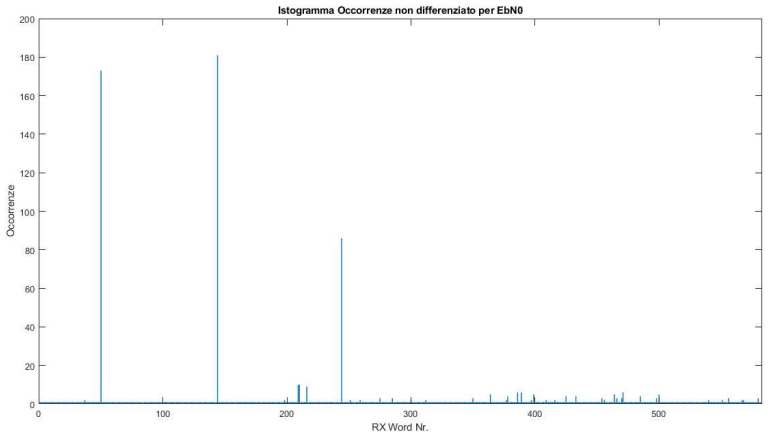

Figura 3.20: Occorenze non differenziate per $\frac{E_b}{N_0}$ - Decoder Normalised Min-Sum, Tail Sequence Randomizzata

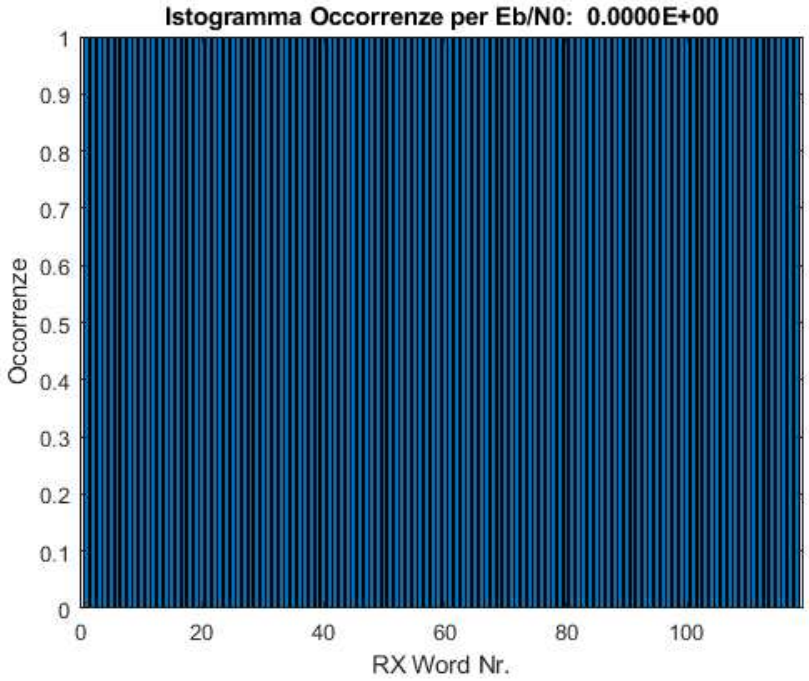

Figura 3.21: Occorrenze per $\frac{E_b}{N_0} = 0$ dB - Decoder Normalised MinSum, Tail Sequence Randomizzata

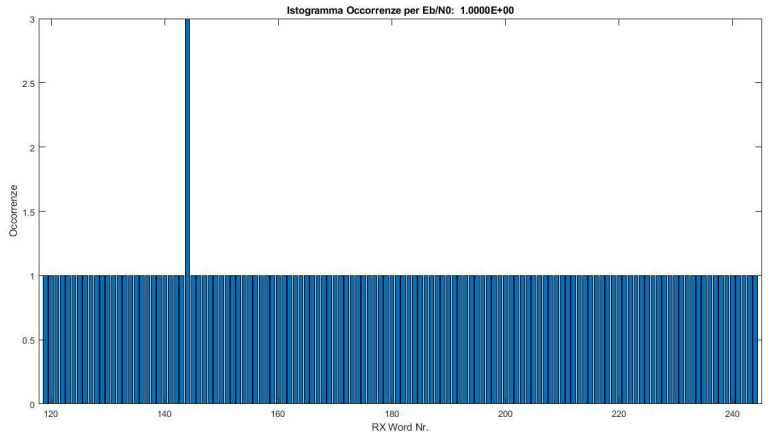

Figura 3.22: Occorrenze per $\frac{E_b}{N_0} = 1$ dB - Decoder Normalised MinSum, Tail Sequence Randomizzata

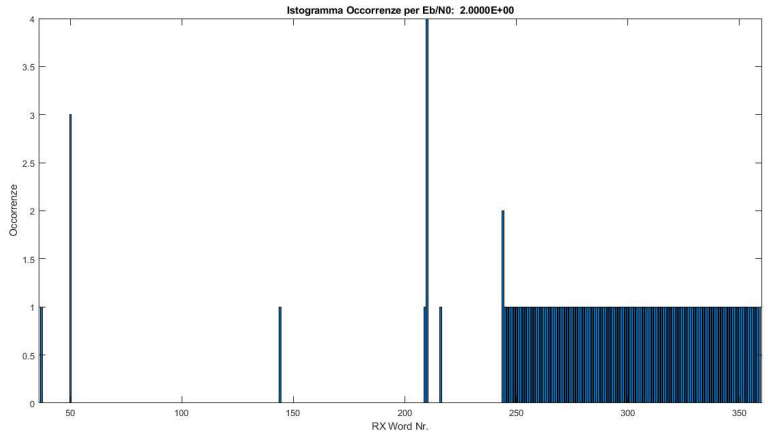

Figura 3.23: Occorrenze per $\frac{E_b}{N_0} = 2$ dB - Decoder Normalised MinSum, Tail Sequence Randomizzata

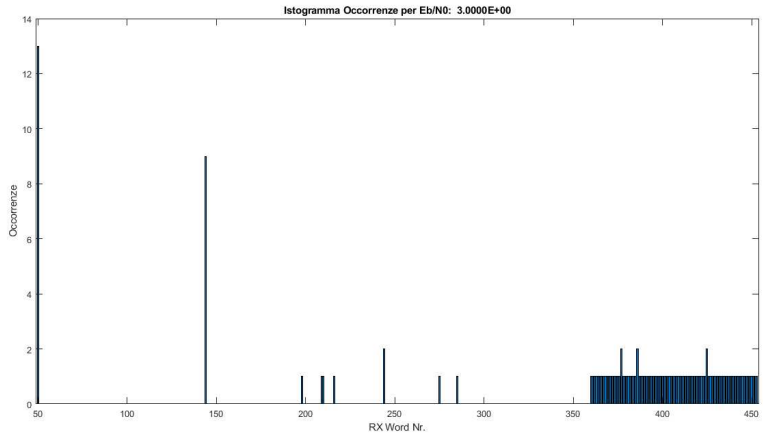

Figura 3.24: Occorrenze per $\frac{E_b}{N_0} = 3$ dB - Decoder Normalised MinSum, Tail Sequence Randomizzata

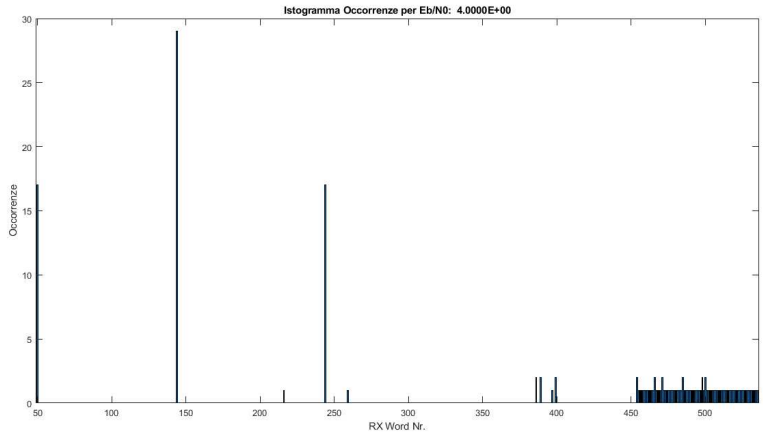

Figura 3.25: Occorrenze per $\frac{E_b}{N_0} = 4$ dB - Decoder Normalised MinSum, Tail Sequence Randomizzata

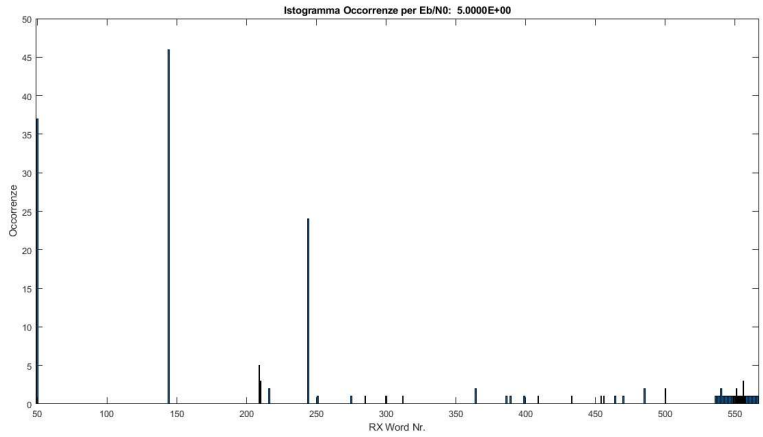

Figura 3.26: Occorrenze per $\frac{E_b}{N_0} = 5$ dB - Decoder Normalised MinSum, Tail Sequence Randomizzata

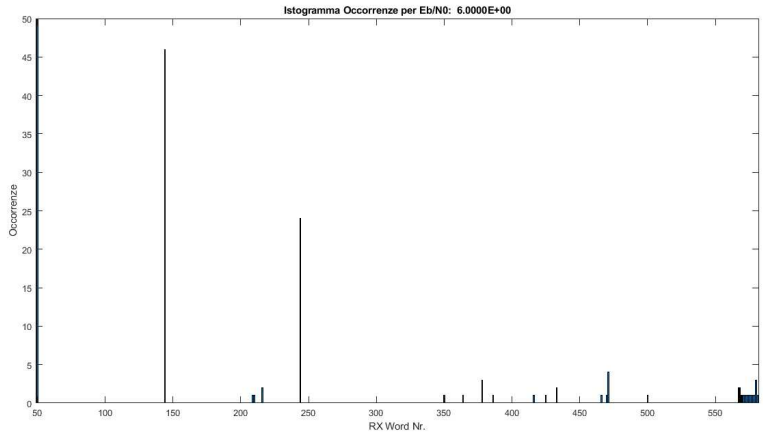

Figura 3.27: Occorrenze per $\frac{E_b}{N_0} = 6$ dB - Decoder Normalised MinSum, Tail Sequence Randomizzata

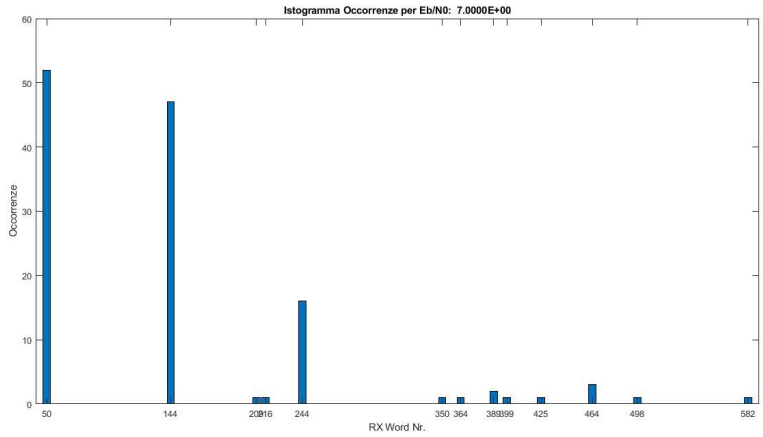

Figura 3.28: Occorrenze per $\frac{E_b}{N_0} = 7$ dB - Decoder Normalised MinSum, Tail Sequence Randomizzata

3.2 Analisi dei Risultati - Tail Sequence Randomizzata

Dall'analisi dei risultati è quindi emerso che, a prescindere dal decoder utilizzato, le parole che vengono più spesso associate alla *Tail Sequence* sono sempre le stesse.
Tutte e tre le parole ricevute più frequentemente si trovano ad una distanza di Hamming dalla *Tail Sequence Randomizzata* pari a 15. Tale valore rappresenta la distanza di Hamming minima tra la *Tail Sequence* e una parola di codice, ossia il minimo numero di bit per cui una parola si può differeziare dalla *Tail Sequence*.

Si mostrano di seguito le tre parole che ricorrono più spesso, come evidente nelle Figure 3.2, 3.11 e 3.20.

- **Parola 1:** 10101010111011001000111000011001100101001000011001
  01100010111110011111101011000011110001111010000000100100010111
  0001110110110101
- **Parola 2:** 10101110011011001110111101001100110000000101011110
  11110001111111000111011101110011111011111010001100100000110
  110101110110000101

- **Parola 3:** 0000101001001100100010110000110011000011010010111010110011011101001010011101110111111110111101000010010101000010110101110110010111

I bit evidenziati in rosso sono i bit che differenziano le tre parole di codice dalla *Tail Sequence Randomizzata*.

L'estrema vicinanza di queste parole di codice alla *Tail Sequence Randomizzata* porta i decoder ad essere "polarizzati" verso di esse. Infatti, spesso la *Tail Sequence*, in decodifica, viene associata ad esse e ciò porta ad un mancato riconoscimento della terminazione dei dati codificati.

3.3 Occorrenze delle Parole di Codice per Tail Sequence Non Randomizzata

In questa sezione sono stati analizzati e commentati i risultati delle simulazioni Monte Carlo lanciate per studiare il caso della *Tail Sequence non Randomizzata*. Considerando che la tail sequence proposta in [8] era stata pensata per lavorare in questo regime, ci si aspetta che i risultati di queste simulazioni siano migliori rispetto a quelli descritti precedentemente.

Anche questa volta sono stati impiegati i tre diversi decoder e i parametri *numero di iterazioni* e *numero massimo di errori sul frame (FEmax)* sono stati impostati rispettivamente a 150 e 3000000, come spiegato in Sezione 2.2.

3.3.1 Occorrenze delle Parole di Codice - Decoder SPA-LLR

In questo primo caso in esame per la *Tail Sequence Non Randomizzata*, tra le 465 parole ricevute sono state identificate 432 parole di codice diverse.

Facendo il paragone con i risultati del caso randomizzato, ora le parole ricevute risultano distribuite in modo molto più uniforme.
Questa conclusione è facilmente raggiungibile anche osservando gli istogrammi che raffigurano le occorrenze delle diverse parole di codice sia al variare del valore del $\frac{E_b}{N_0}$ (si vedano le Figure 3.30, 3.31, 3.32, 3.33, 3.34, 3.35, 3.36 e 3.37) sia indifferentemente dal suo valore (si veda la Figura 3.29).

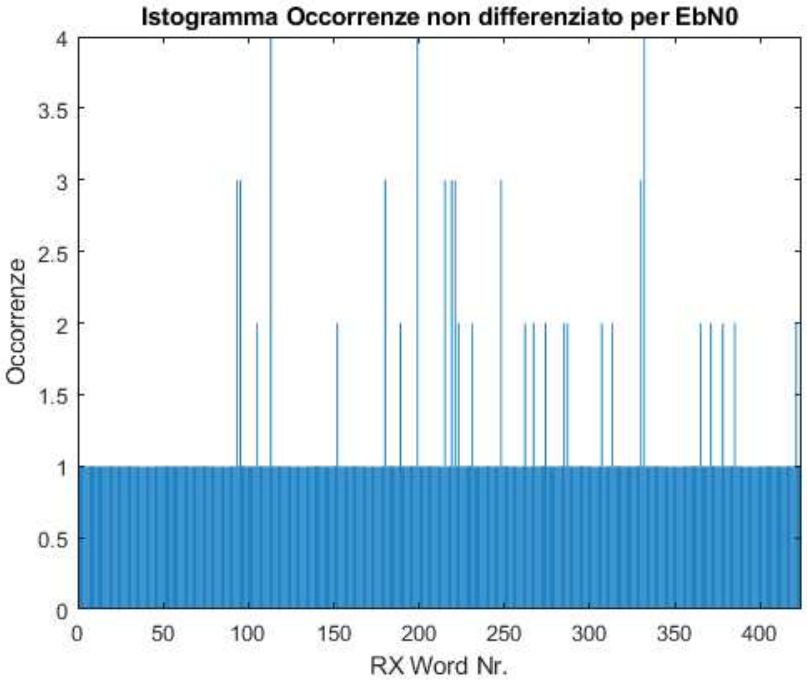

Figura 3.29: Occorenze non differenziate per $\frac{E_b}{N_0}$ - Decoder SPA-LLR, Tail Sequence Non Randomizzata

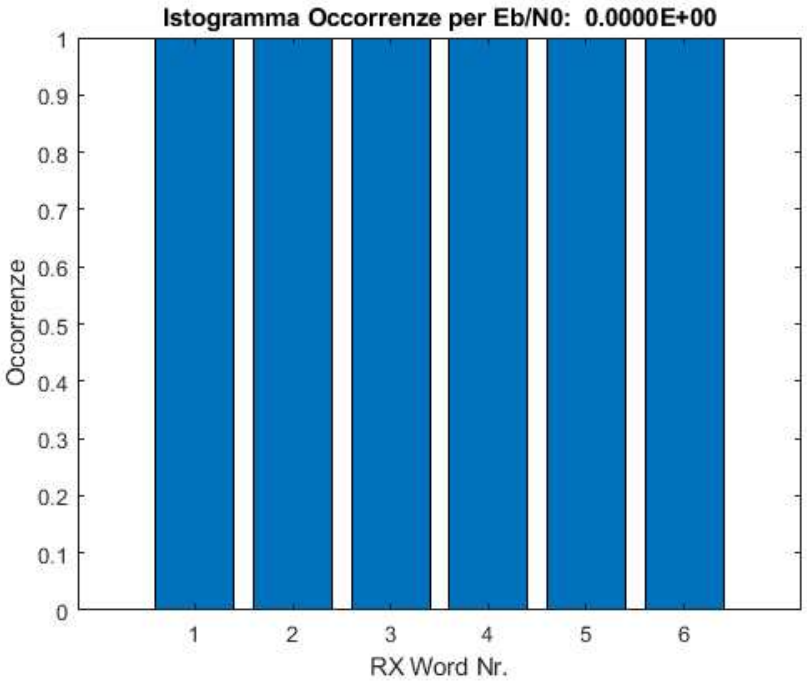

Figura 3.30: Occorrenze per $\frac{E_b}{N_0} = 0$ dB - Decoder SPA-LLR, Tail Sequence Non Randomizzata

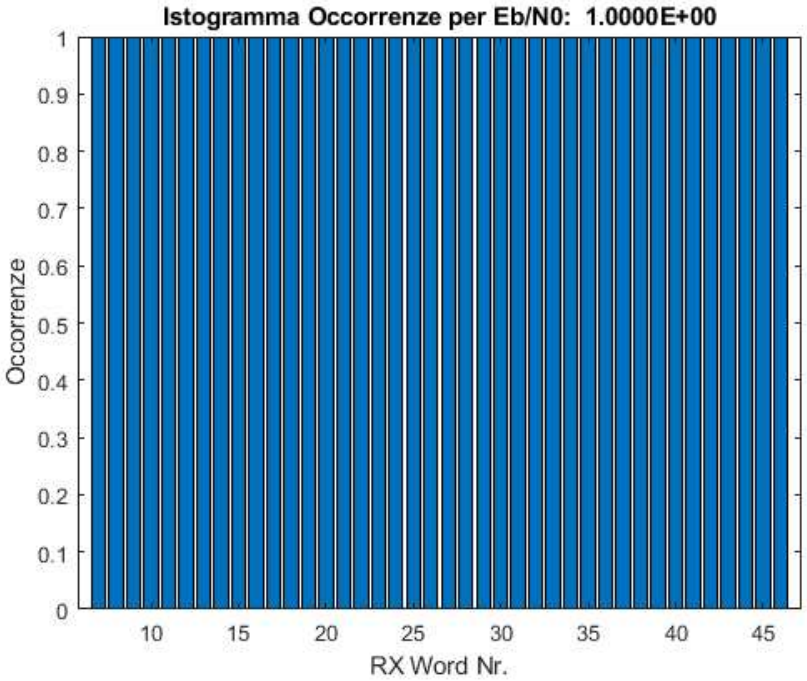

Figura 3.31: Occorrenze per $\frac{E_b}{N_0} = 1$ dB - Decoder SPA-LLR, Tail Sequence Non Randomizzata

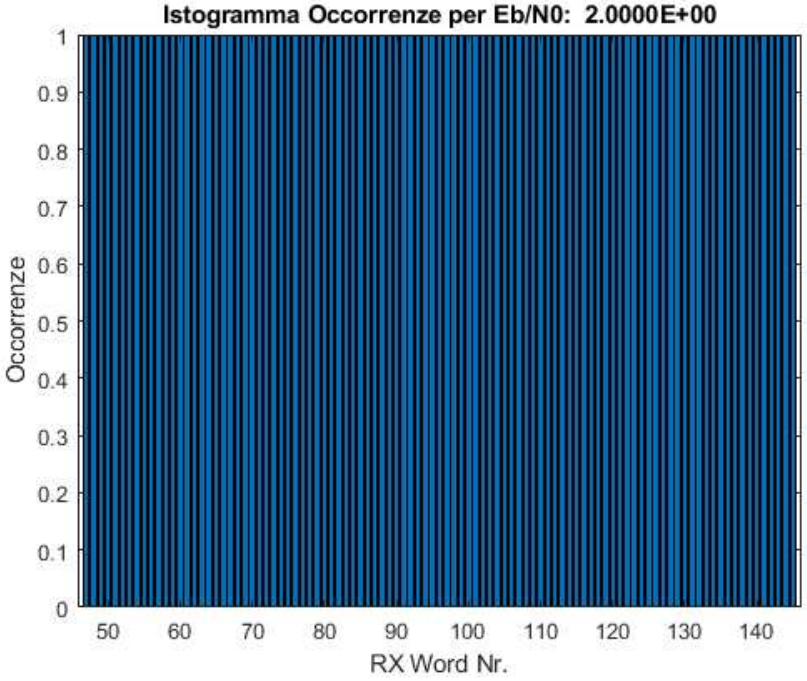

Figura 3.32: Occorrenze per $\frac{E_b}{N_0} = 2$ dB - Decoder SPA-LLR, Tail Sequence Non Randomizzata

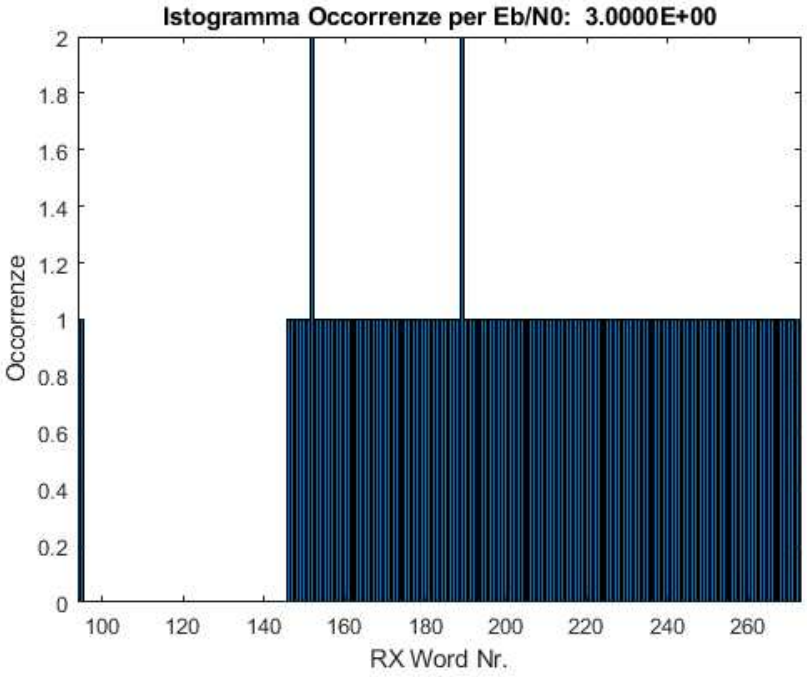

Figura 3.33: Occorrenze per $\frac{E_b}{N_0} = 3$ dB - Decoder SPA-LLR, Tail Sequence Non Randomizzata

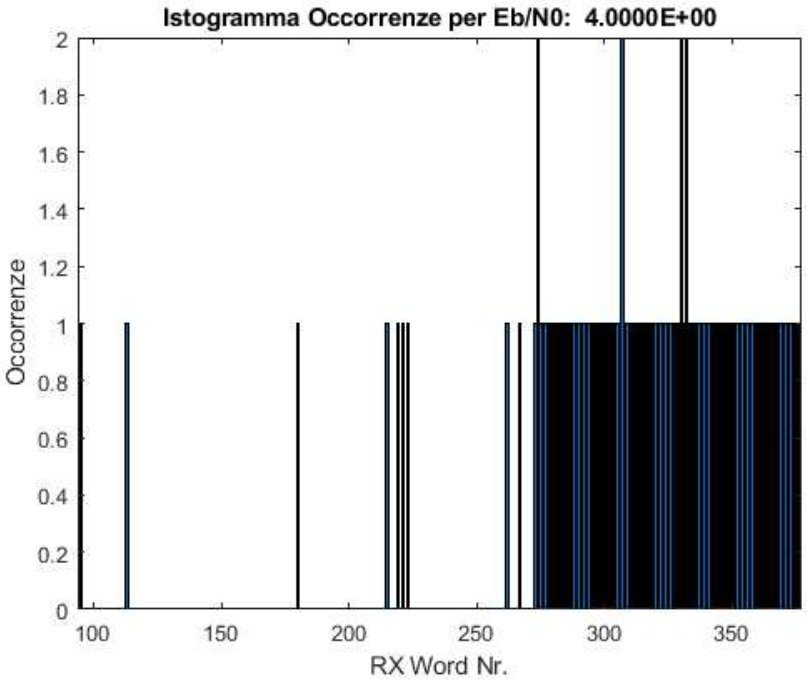

Figura 3.34: Occorrenze per $\frac{E_b}{N_0} = 4$ dB - Decoder SPA-LLR, Tail Sequence Non Randomizzata

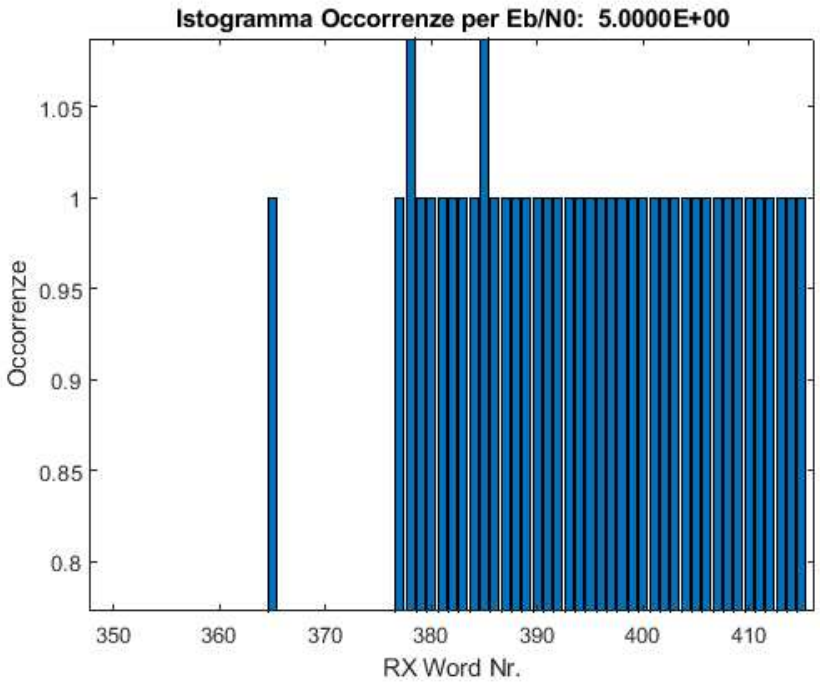

Figura 3.35: Occorrenze per $\frac{E_b}{N_0} = 5$ dB - Decoder SPA-LLR, Tail Sequence Non Randomizzata

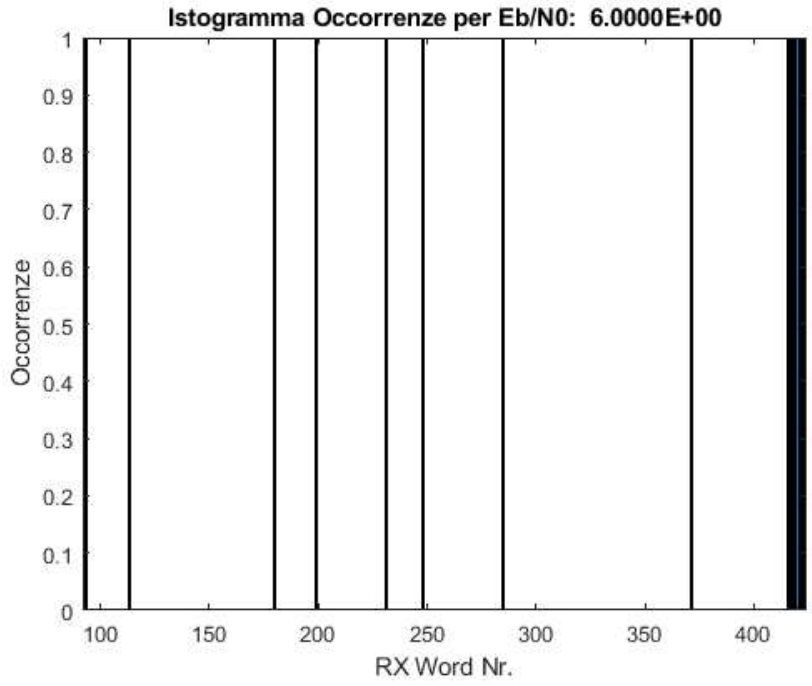

Figura 3.36: Occorrenze per $\frac{E_b}{N_0} = 6$ dB - Decoder SPA-LLR, Tail Sequence Non Randomizzata

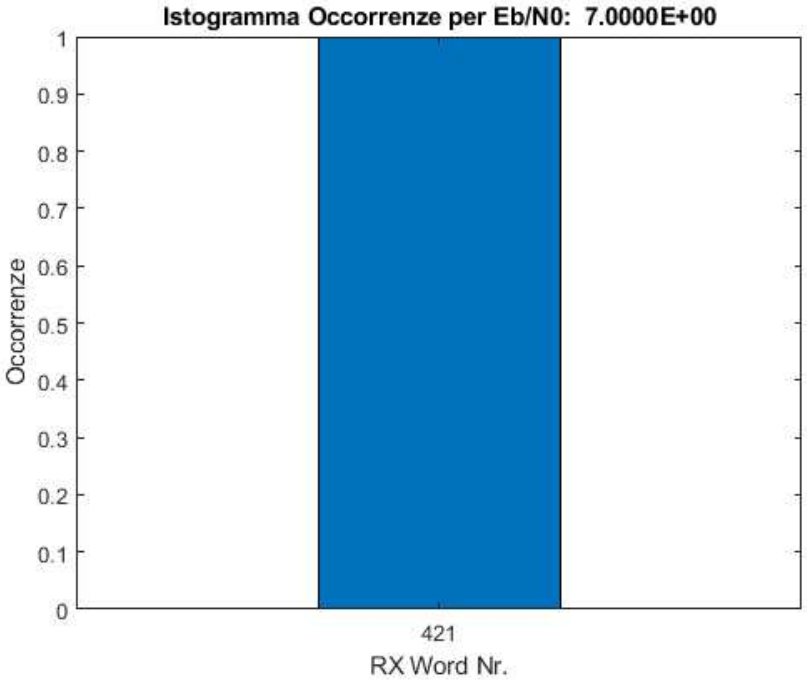

Figura 3.37: Occorrenze per $\frac{E_b}{N_0} = 7$ dB - Decoder SPA-LLR, Tail Sequence Non Randomizzata

Di tutte le parole ricevute, in questo caso, le più ricorrenti risultano essere la *113-esima*, la *199-esima* e la *332-esima* che sono state rilevate in totale solamente 4 volte ognuna, al variare del valore di $\frac{E_b}{N_0}$.
La minima distanza di Hamming dalla *Tail Sequence Non Randomizzata* è pari a 18. Tutte e tre queste parole si trovano alla minima distanza individuata tra Tail Sequence e parole di codice ma, a differenza del caso con *Tail Sequence Randomizzata*, queste non sono le uniche a trovarsi a tale distanza. È stato identificato un totale di 46 parole a distanza 18.

3.3.2 Occorrenze delle Parole di Codice - Decoder MinSum

In questo secondo caso in esame, su un totale di 34 parole ricevute sono state identificate esattamente 34 parole di codice diverse; quindi, le parole sono distribuite in modo equo sia se non si considera il variare del valore del $\frac{E_b}{N_0}$ (si veda la Figura 3.38) sia se lo si considera (si vedano le Figure 3.39, 3.40, 3.41, 3.42, 3.43 e 3.44).

Inoltre, si osserva che per $\frac{E_b}{N_0}$ uguale a 3 e 4 dB non è stata ottenuta nessuna parola, e per questo motivo non sono stati riportati i rispettivi grafici.

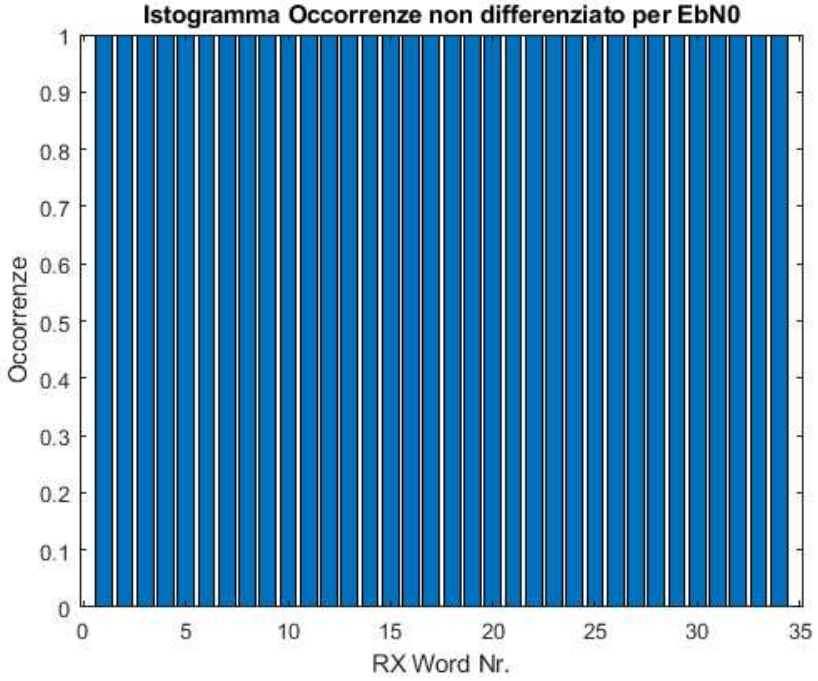

Figura 3.38: Occorenze non differenziate per $\frac{E_b}{N_0}$ - Decoder MinSum, Tail Sequence Non Randomizzata

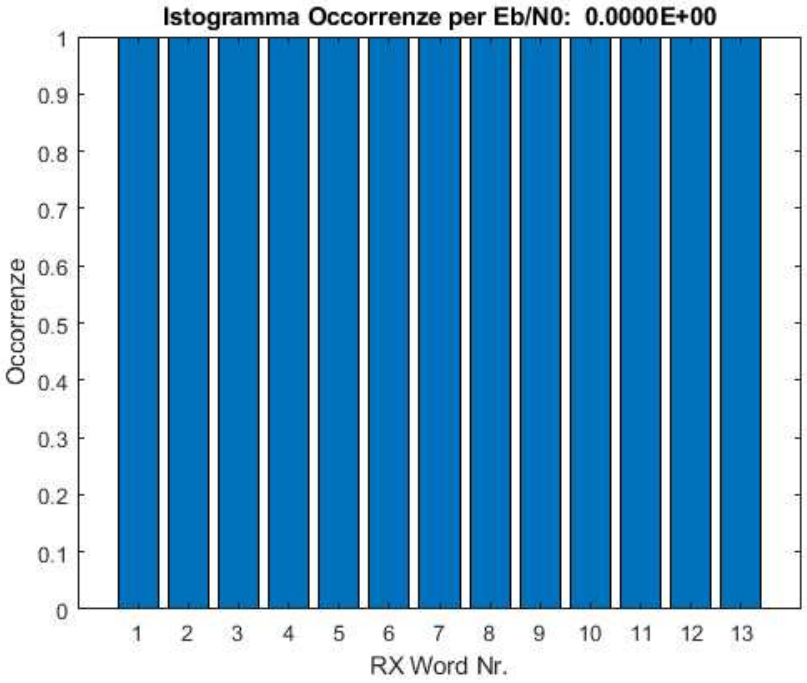

Figura 3.39: Occorrenze per $\frac{E_b}{N_0} = 0$ dB - Decoder MinSum, Tail Sequence Non Randomizzata

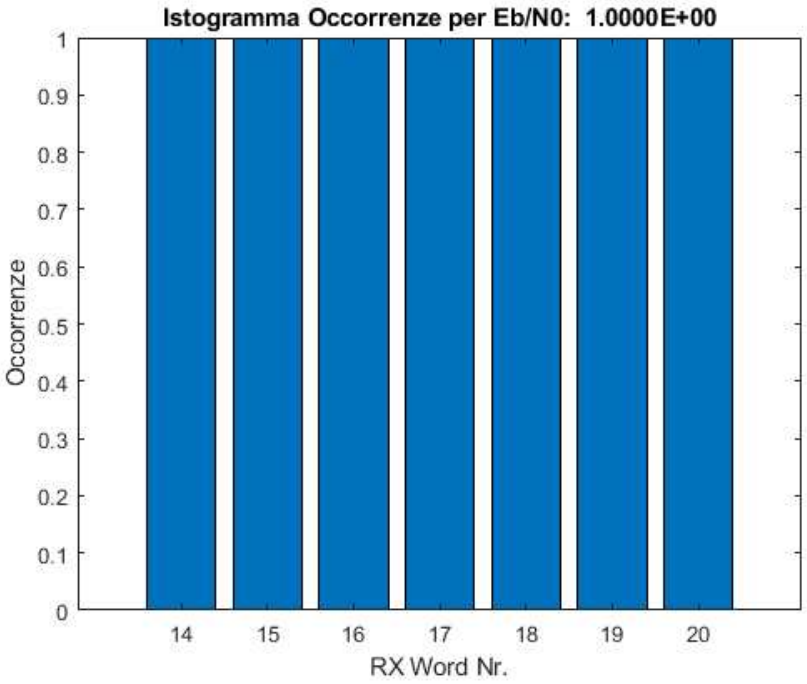

Figura 3.40: Occorrenze per $\frac{E_b}{N_0} = 1$ dB - Decoder MinSum, Tail Sequence Non Randomizzata

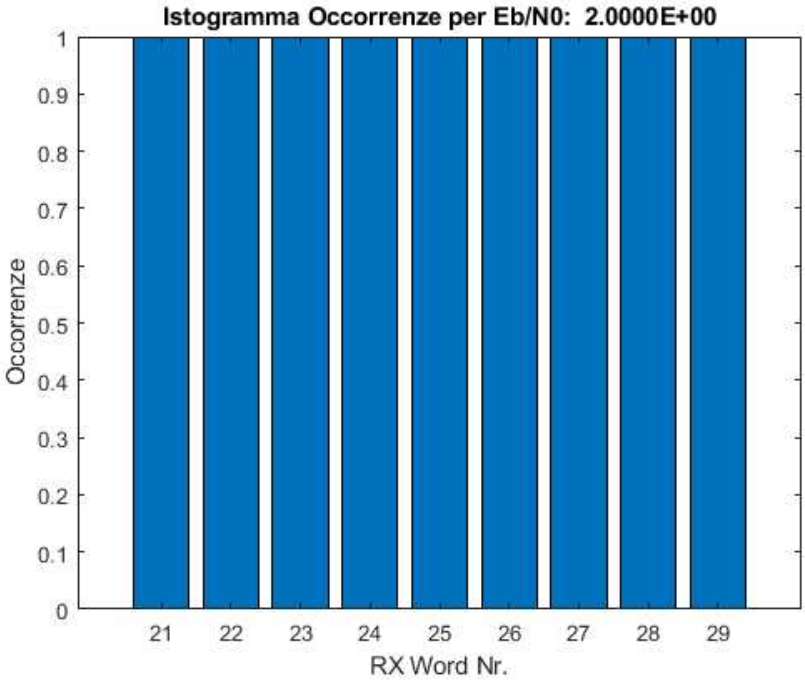

Figura 3.41: Occorrenze per $\frac{E_b}{N_0} = 2$ dB - Decoder MinSum, Tail Sequence Non Randomizzata

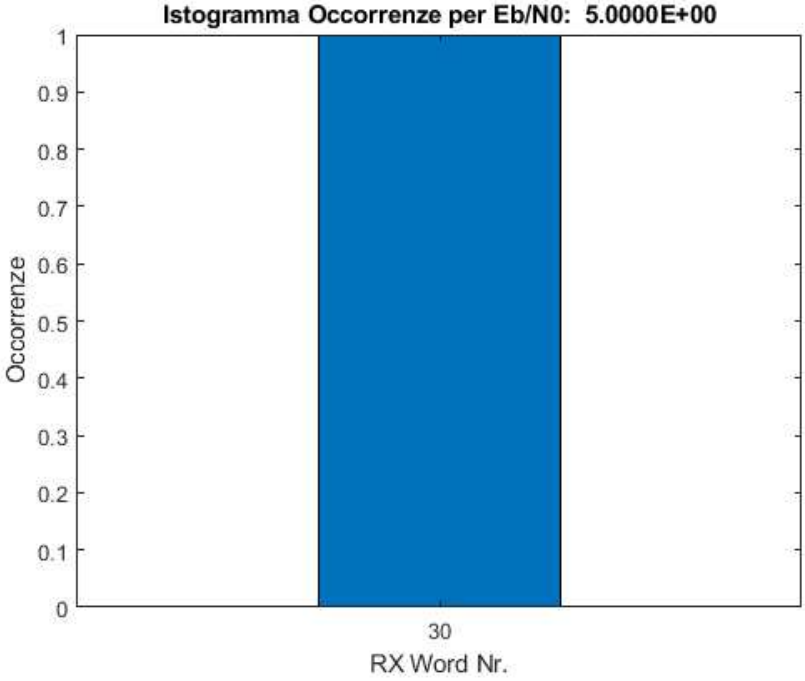

Figura 3.42: Occorrenze per $\frac{E_b}{N_0} = 5$ dB - Decoder MinSum, Tail Sequence Non Randomizzata

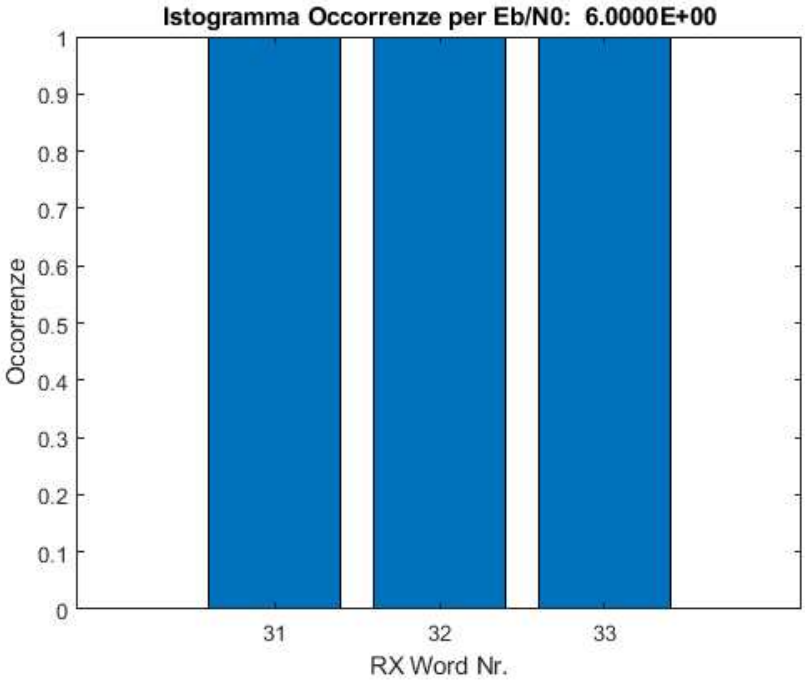

Figura 3.43: Occorrenze per $\frac{E_b}{N_0} = 6$ dB - Decoder MinSum, Tail Sequence Non Randomizzata

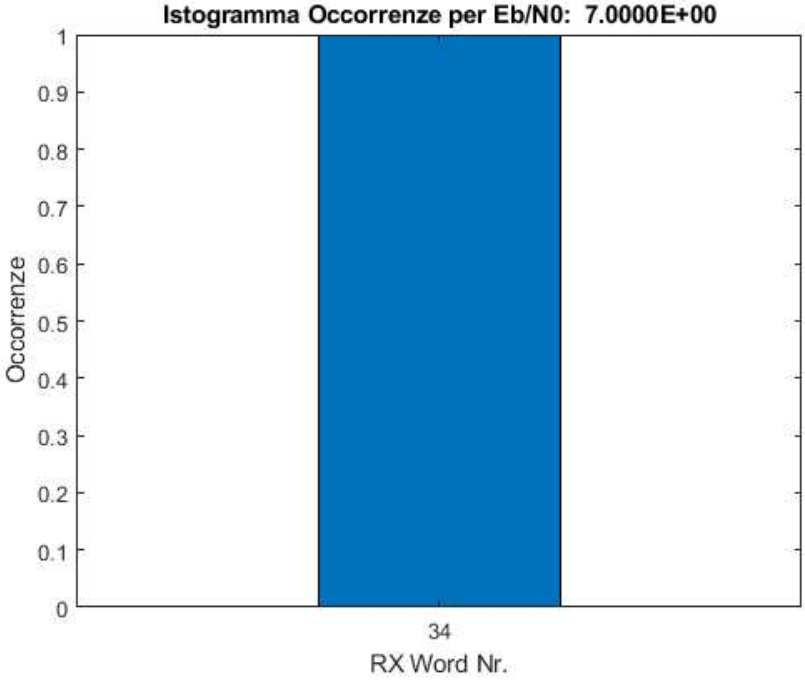

Figura 3.44: Occorrenze per $\frac{E_b}{N_0} = 7$ dB - Decoder MinSum, Tail Sequence Non Randomizzata

La minima distanza di Hamming delle parole di codice dalla *Tail Sequence Non Randomizzata* calcolata è pari a 18 e solo 6 parole di tutte quelle ricevute si trovano a tale distanza minima.

3.3.3 Occorrenze delle Parole di Codice - Decoder Normalised MinSum

In quest'ultimo caso preso in esame per la *Tail Sequence Non Randomizzata*, su un totale di 320 parole ricevute sono state identificate 318 parole di codice diverse.
Le due parole ricevute che risultano essere più ricorrenti sono la *296-esima* e la *297-esima*, ma solo con due occorrenze ciascuna.

Si può dire che le parole siano distribuite in modo uniforme sia considerando il variare del valore di $\frac{E_b}{N_0}$ (si veda la Figura 3.45) sia osservando le occorrenze in corrispondenza dei diversi valori (si vedano le Figure 3.46, 3.47, 3.48, 3.49, 3.50 e 3.51). Si osserva che, in questo caso, la decodifica non è mai andata a buon fine per $\frac{E_b}{N_0}$ uguale a 6 e 7 dB, e per le stesse motivazioni spiegate in precedenza non ne vengono riportati i rispettivi grafici.

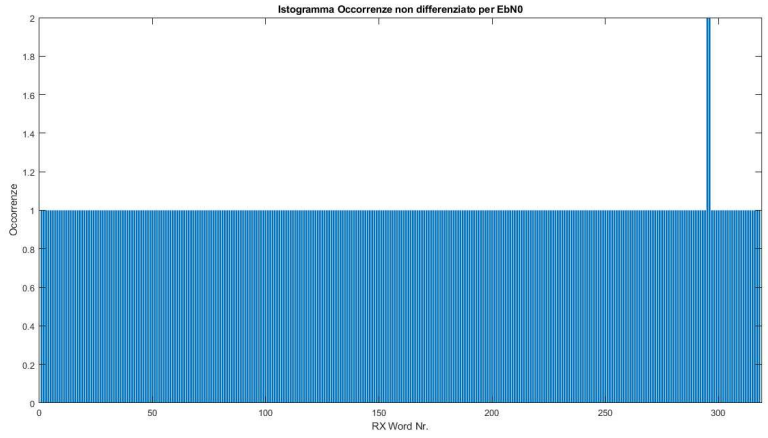

Figura 3.45: Occorenze non differenziate per $\frac{E_b}{N_0}$ - Decoder Normalised Min-Sum, Tail Sequence Non Randomizzata

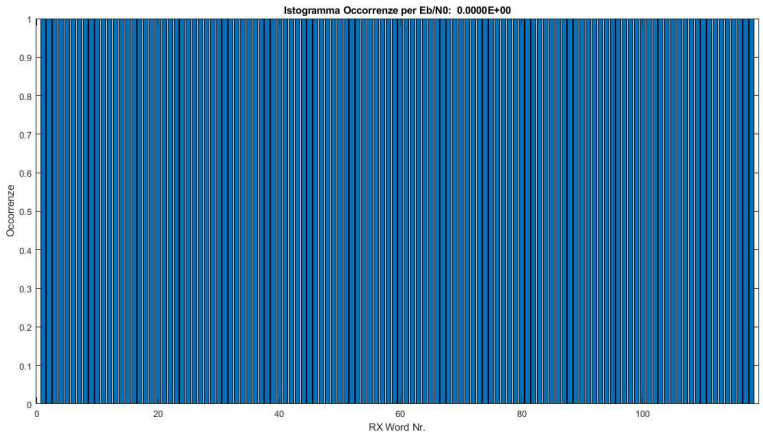

Figura 3.46: Occorrenze per $\frac{E_b}{N_0} = 0$ dB - Decoder Normalised MinSum, Tail Sequence Non Randomizzata

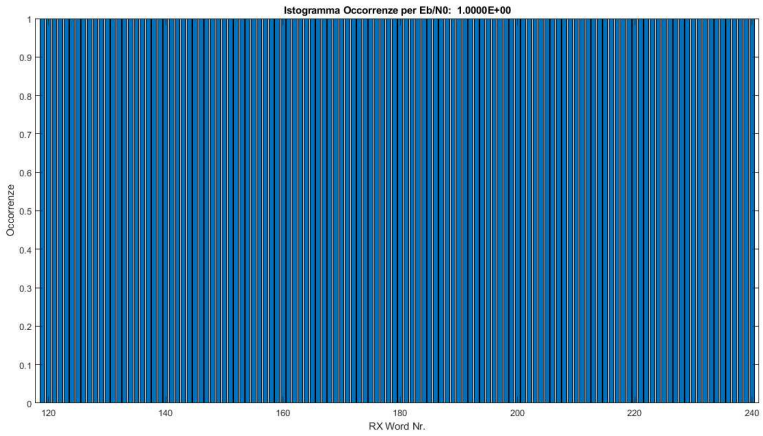

Figura 3.47: Occorrenze per $\frac{E_b}{N_0} = 1$ dB - Decoder Normalised MinSum, Tail Sequence Non Randomizzata

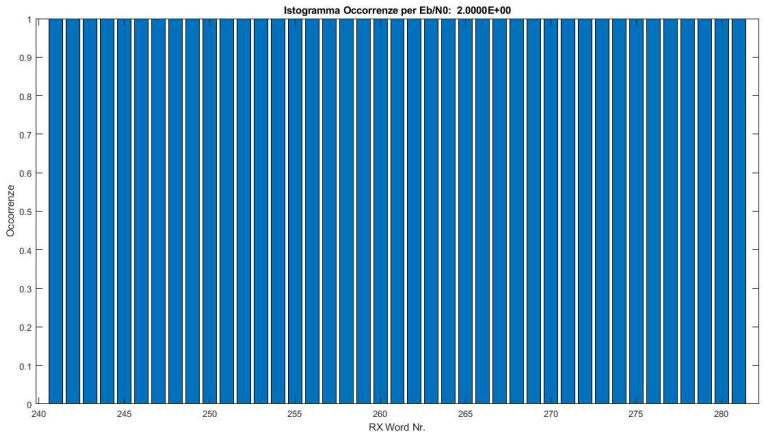

Figura 3.48: Occorrenze per $\frac{E_b}{N_0} = 2$ dB - Decoder Normalised MinSum, Tail Sequence Non Randomizzata

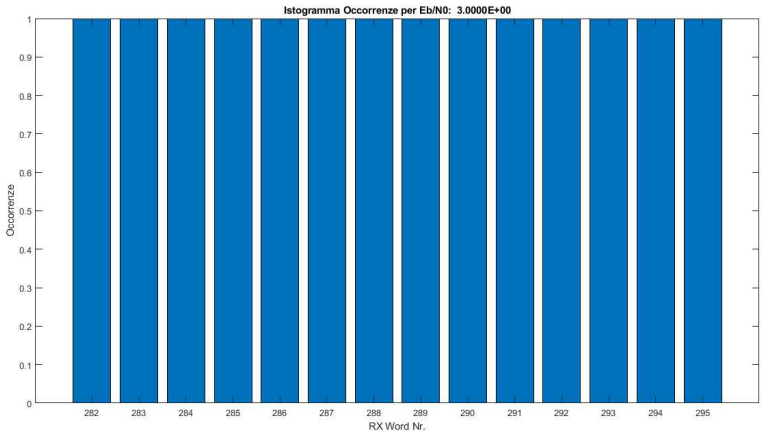

Figura 3.49: Occorrenze per $\frac{E_b}{N_0} = 3$ dB - Decoder Normalised MinSum, Tail Sequence Non Randomizzata

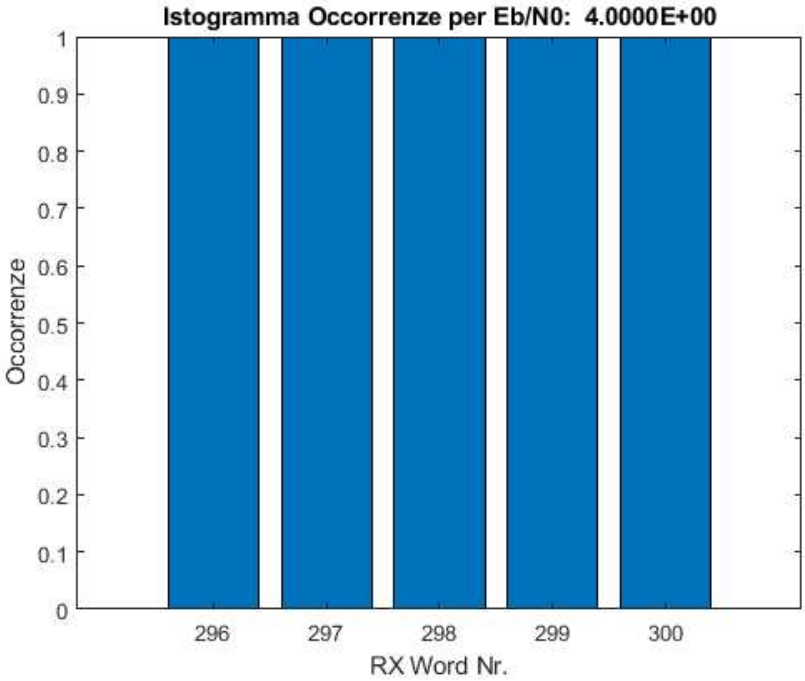

Figura 3.50: Occorrenze per $\frac{E_b}{N_0} = 4$ dB - Decoder Normalised MinSum, Tail Sequence Non Randomizzata

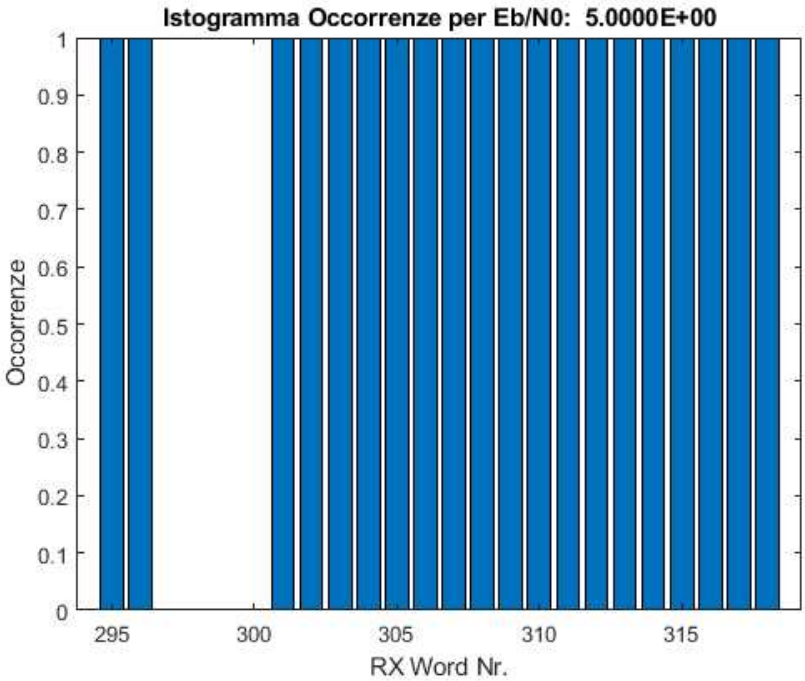

Figura 3.51: Occorrenze per $\frac{E_b}{N_0} = 5$ dB - Decoder Normalised MinSum, Tail Sequence Non Randomizzata

Anche nel caso di questo decoder, per il caso della *Tail Sequence Non Randomizzata*, la minima distanza di Hamming delle parole da essa è pari a 18, e le parole di codice ricevute che si trovano a tale distanza sono 12.

Dall'analisi delle occorrenze nel caso della *Tail Sequence Non Randomizzata* è emerso che la distanza minima di Hamming a cui si trovano le parole di codice dalla tail sequence è sempre pari a 18, come dovrebbe essere da definizione della stessa in [8]. A differenza del caso con la *Tail Sequence Randomizzata* non si nota, infatti, una polarizzazione verso un gruppo ristretto di parole di codice, e si può dire che le parole sono distribuite in modo uniforme a prescindere dal decoder impiegato.

3.4 Commento dei Risultati ottenuti dall'Analisi delle Occorrenze delle Parole di Codice

Ciò che è emerso dall'analisi riportata nel presente capitolo è di estrema importanza: si evidenzia come l'utilizzo della *Tail Sequence Randomizzata* porti a polarizzare il decoder intorno alle tre parole di codice riportate nella Sezione 3.1.
Nel caso in cui venga utilizzato il fallimento della decodifica come approccio di riconoscimento della *tail sequence*, questo risultato sottolinea come lo standard attualmente proposto dal CCSDS (si vedano [6, 7]), che consiglia l'utilizzo di un randomizer prima dell'incapsulamento dei dati codificati in una CLTU, non permetta di sfruttare al meglio la *Tail Sequence* proposta in [8], che è stata progettata per non essere randomizzata.

Al contrario, dai risultati dell'analisi nel caso della *Tail Sequence Non Randomizzata* si può facilmente osservare come i decoder non siano polarizzati verso nessuna parola di codice in particolare, nonostante più di una si trovi alla distanza di Hamming minima calcolata (pari a 18) dalla tail sequence, distanza che coincide esattamente con quella prevista da Kenneth Andrews in [8].

Capitolo 4

Ricerca della Tail Sequence

Innanzitutto, è stato analizzato il metodo tramite il quale è stata ricavata la sequenza che attualmente viene utilizzata come *Tail Sequence* [8].

Si è studiata la possibile struttura geometrica delle regioni di Voronoi caratteristiche delle parole di codice, considerando che nella ricerca della sequenza ideale viene cercato un vertice tra tali regioni (vedi Figura 4.1).
La *regione di Voronoi* relativa a una parola di codice è quella porzione di spazio contenente tutti i punti che il decoder ricondurrebbe a tale parola di codice.

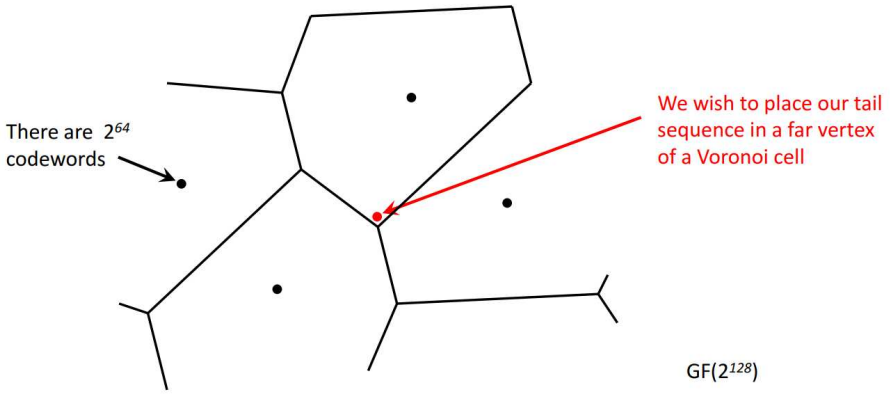

Figura 4.1: Posizionamento su un Vertice di una Regione di Voronoi

La procedura seguita per la ricerca di un *vertice* è descritta di seguito.
Partendo da una parola di codice, ci si allontana da essa un passo alla volta (ovvero complementando un bit alla volta) fino ad arrivare sul bordo di una regione di Voronoi, ossia in un punto in cui si è ugualmente distanti da una parola di codice. A quel punto ci si allontana da entrambe le parole di codice

mantenendosi sul bordo della regione, fino a che non ci si trova ugualmente distanti da una terza parola. A questo punto si può dire di aver trovato un *vertice*.

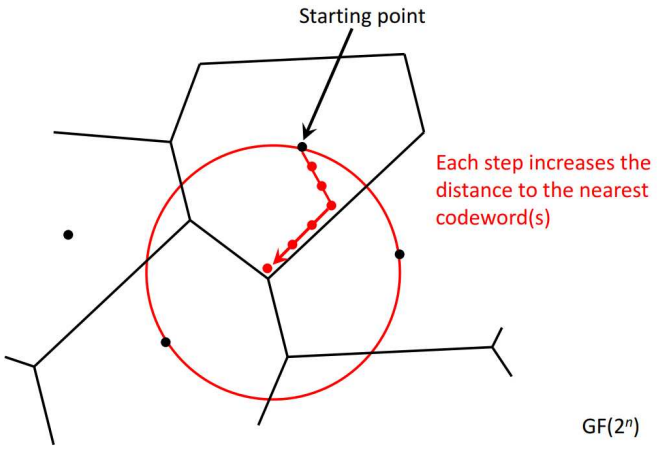

Figura 4.2: Ricerca del Vertice di una Regione di Voronoi

Più nello specifico, in [8] è stato poi proposto un semplice algoritmo per la ricerca della *Tail Sequence* partendo da una sequenza iniziale non nulla. L'algoritmo si può riassumere nei seguenti step:

- si trovano tutte le parole a distanza minima d_{min} dalla sequenza iniziale;
- se possibile, si ricerca un passo che permetta a tutte le parole che si trovano a d_{min} dalla sequenza iniziale di aumentare tale distanza. Trovata la coordinata del bit che permette di fare tale passo, si complementa tale bit nella sequenza iniziale.
- Questo ovviamente va ad aumentare la distanza per il gruppo di parole di codice che si trovavano a distanza minima, ma la va inevitabilmente a diminuire per un altro gruppo di parole di codice.
- Si ripetono nuovamente i passaggi precedenti fintanto che non si trova più un altro bit da complementare all'interno della parola di codice.

4.1 Esempio di Ricerca della Tail Sequence

In questa sezione si propone un toy example, in cui si va a ricercare la *Tail Sequence* per il codice a ripetizione (4,1).

Si considerano le parole di codice $c_0 = [0000]$ e $c_1 = [1111]$ e il vettore iniziale $v = [1000]$.

Viene calcolata la distanza minima delle parole di codice dal vettore iniziale facendo una somma *XOR*. Si calcola che la d_{min} alla prima iterazione è pari a 1 e la parola che si trova a questa distanza da $v = [1000]$ è $c_0 = [0000]$.
Per allontanarsi da c_0 è necessario complementare una tra la seconda, la terza e la quarta coordinata di v, che in questo lavoro viene scelta in maniera random.
Supponendo che sia stato fatto il complemento a 2 del bit corrispondente alla seconda coordinata, ora il vettore iniziale risulta essere uguale a $v = [1100]$.

Ricalcolando ora la distanza minima di c_0 e c_1 da v, si osserva che ora $d_{min} = 2$ ed entrambe le parole di codice si trovano a questa distanza.
A questo punto non esiste una coordinata comune ad entrambe le parole, per cui complementando il bit corrispondente si può allontanare il vettore iniziale v da una parola senza avvicinarsi all'altra.
Si può quindi affermare di aver trovato un *vertice*, che in questo caso corrisponde alla sequenza [1100].

Capitolo 5

Conclusioni

In questa tesi sono state analizzate le prestazioni dello standard proposto dal CCSDS [6, 7] nel caso in cui si adotti l'approccio di riconoscimento della *Tail Sequence* basato sul trigger di un errore nel processo di decodifica usufruendo dei risultati di simulazioni Monte Carlo, variando il decoder utilizzato tra SPA-LLR, MinSum e Normalised MinSum.

Si è osservato che, se si adotta questo approccio, l'utilizzo di un *randomizer* (consigliato nello standard) prima dell'incapsulamento dei dati codificati non permette di sfruttare al massimo delle sue possibilità la sequenza proposta in [8] come *Tail Sequence*, perchè essa non era stata pensata per essere randomizzata.
Dal resoconto dell'analisi è emerso che in fase di decodifica, a prescindere dal tipo di decoder impiegato, vi era una sorta di "polarizzazione" verso le tre parole a distanza minima (pari a 15) dalla *Tail Sequence Randomizzata*.

Al contrario, dall'analisi dei risultati delle simulazioni effettuate risulta che, nel caso con *Tail Sequence Non Randomizzata*, i decoder non propendono verso nessuna parola in particolare e quindi le prestazioni attese, conseguenti all'utilizzo della sequenza proposta in [8], sono effettivamente verificate.

Appendice A

Codice Sviluppato

In questo capitolo è riportato il codice sviluppato per questo lavoro di tesi. Le sezioni sono ordinate in questo modo:

- Script del Main per la Valutazione delle Prestazioni del Sistema
- Funzione per il Calcolo della Distanza di Hamming fra Parole Ricevute e la Tail Sequence
- Funzione per il Calcolo delle Occorrenze delle Parole di Codice
- Script per la Verifica delle Parole Ricevute
- Funzione per Estrazione Parole Ricevute dal File di Testo
- Funzione per Generazione Matrice di Parità
- Script Toy Example
- Funzione per la Ricerca della Tail Sequence
- Funzione per la Ricerca della Coordinata del bit da Complementare

A.1 Script Main per Analisi delle Prestazioni del Sistema

In questa sezione è riportato il codice dello script da cui sono state richiamate tutte le funzioni e gli script necessari per analizzare le prestazioni del sistema (si vedano Figure A.1,A.2, A.3, A.4, A.5 e A.6).

```
clear all; clc; close all;

%Tail sequence presa dal paragrafo sugli LDPC nel blue book
Tail_seq = [0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 1 ...
    0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1];

Tail_sequence = reset_set_seq(Tail_seq);

%Idle sequence definita sulla base di quanto scritto nel blue book
%(alternanza di 0 e 1)
Idle_seq = [0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 ...
    1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1];

Idle_sequence = reset_set_seq(Idle_seq);

%% Randomizzazione delle sequenze di Idle e Coda

SequenceLength = 128;
InitialSeed = 2^8-1;
feedbacks=[1,2,3,4,5,7];
N=8;

[Seq,Final_Seed] = LFSRseq(N,InitialSeed,feedbacks,SequenceLength);

Tail_seq_rand = xor(Seq,Tail_seq);
Tail_sequence_rand = reset_set_seq(Tail_seq_rand);

Idle_seq_rand  = xor(Seq,Idle_seq);
Idle_sequence_rand = reset_set_seq(Idle_seq_rand);
```

Figura A.1: Parte 1 dello script del main

```
%% Simulazione Prob Start Sequence

%Start sequence presa dal paragrafo sugli LDPC nel blue book
Start_seq = [0 0 0 0 0 0 1 1 0 1 0 0 0 1 1 1 0 1 1 1 0 1 1 0 1 1 0 0 0 1 1 1 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 0 1 0 0 1 0 1 0 1 1 0 1 1 0 0 0 0];

EbN0_vec = 0:7; % vettore del EbN0 con valori in dB

M=2;
Pb = berawgn(EbN0_vec, 'psk', M, 'nondiff');

%Pmd calcolata con la formula [7.1]

Pmd = 1 - (1-Pb).^64;

for i = 1:7
    Pmd = Pmd - nchoosek(length(Start_seq),i).*(Pb.^i.*(1-Pb).^(64-i)); % nchoosek funzione per il calcolo del coefficiente binomiale
end

figure;
semilogy(EbN0_vec, Pmd, '-xr');
grid on;
title('Probabilità di Miss Detection');
xlabel('Eb/N0'); ylabel('Probabilità');

legend('Pmd calcolata');
```

Figura A.2: Parte 2 dello script del main

```
%% CER per Decoder (Tesi Nicola)

%valori presi dalla tesi di dottorato di Nicola

CER_LDPC_SPA = [1,0.8,0.4,0.06, 0.0023, 3*10^-5, 1.6*10^-7];

CER_LDPC_MS = [1,0.8,0.5,0.12,0.0062,8.2*10^-5];

CER_LDPC_NMS = [1,0.8,0.35,0.07, 0.0033, 3.2*10^-5];

%% CER per Decoder (Simulazione)

%valori risultanti dalle simulazioni (100 iterazioni, 100 FeMax)

CER_SPALLR = [9.709*10^-1, 7.937*10^-1, 3.559*10^-1, 7.022*10^-2,2.034*10^-3, 2.354*10^-5,1.340*10^(-7),0];

CER_MINSUM = [9.615*10^-1, 8.772*10^-1, 4.274*10^-1,1.134*10^-1,5.121*10^-3, 6.008*10^-5, 0, 0];

CER_NORMMS = [9.346*10^-1, 7.752*10^-1, 3.289*10^-1, 6.207*10^-2, 1.905*10^-3, 1.600*10^-5, 0, 0];

figure_Nicola(EbN0_vec, CER_SPALLR, CER_LDPC_SPA, CER_MINSUM, CER_LDPC_MS, CER_NORMMS, CER_LDPC_NMS);
```

Figura A.3: Parte 3 dello script del main

```
%% Confronto con Tesi

%valori risultanti dalle simulazioni (150 nr. iterazioni, 3000000 FeMax)

CER_SPALLR_3mil = [1,1,1,0.9999, 0.9998, 0.9998, 0.9997, 0.9995];

CER_MINSUM_3mil = [1, 1,1,1,1,0.9999, 0.9998, 0.9997];

CER_NMS_3mil = [1,1,1,1, 0.9999, 0.9997, 0.9994, 0.9989];

%Valori P_term calcolati per i diversi valori di EbN0 con la formula [7.2]
%della tesi di Andrea con i dati ottenuti dalle simulazioni (nr. iterazioni
%150, FeMax 3000000)

P_term_SPALLR_3mil  = [6/2999993, 45/2999955, 129/2999871, 162/2999838, 136/2999864, 114/2999886, 68/2999932, 34/2999966];

P_term_MS_3mil = [16/2999983, 12/2999988, 13/2999987, 20/2999987, 24/2999976, 19/2999981, 18/2999982, 11/2999989];

P_term_NMS_3mil = [118/2999881, 128/2999872, 128/2999872, 127/2999873, 160/2999840, 171/2999829, 161/2999839, 128/2999872];

%Valori P_term estratti dai grafici [7.5, 7.6, 7.7] della tesi di Andrea

P_term_SPALLR_AB = [2*10^(-6),2*10^(-5),5*10^(-5),6*10^(-5),5.5*10^(-5),5*10^(-5), 2.5*10^(-5), 10^(-5)];

P_term_MS_AB = [7*10^(-6),6*10^(-6),7*10^(-6),7.5*10^(-6),8*10^(-6), 8*10^(-6),9*10^(-6),7*10^(-6)];

P_term_NMS_AB = [5.5*10^(-5),5.5*10^(-5),5.9*10^(-5),6*10^(-5),6.2*10^(-5),6.5*10^(-5), 7*10^(-5), 6*10^(-5)];
```

Figura A.4: Parte 4 dello script del main

```
%Valori P_rej estratti dai grafici [7.5, 7.6, 7.7] della tesi di Andrea

P_rej_SPALLR_AB = [1,0.99,0.54,0.1,0.4*10^-2,0.69*10^-4,0.25*10^-4,1*10^-5];

P_rej_MS_AB = [1,0.99,0.65,0.25,0.89*10^-2,0.89*10^-4,0.65*10^-5,0.4*10^-5];

P_rej_NMS_AB = [1,0.99,0.65,0.08,0.5*10^-2,0.95*10^-4,0.7*10^-4,0.6*10^-4];

%Valori P_rej calcolati con la formula [7.3] della tesi di Andrea

P_rej_SPALLR = P_term_SPALLR_3mil + (1-P_term_SPALLR_3mil).*(Pmd + ((1-Pmd).*[CER_LDPC_SPA 0]));

P_rej_MS = P_term_MS_3mil + (1-P_term_MS_3mil).*(Pmd + ((1-Pmd).*[CER_LDPC_MS 0 0]));

P_rej_NMS = P_term_NMS_3mil + (1-P_term_NMS_3mil).*(Pmd + ((1-Pmd).*[CER_LDPC_NMS 0 0]));


run("figure_Bertuccini.m");
```

Figura A.5: Parte 5 dello script del main

```
%% Distanza di Hamming dalla tail sequence randomizzata

distH_perc_rand_SPALLR = get_perc(Tail_seq_rand, 'Risultati Simulazioni/Risultati Simulazione 3000000 SPA-LLR/Rx_Codeword.txt');
% distanza in percentuale dalla tail sequence randomizzata di tutte le parole di codice decodificate (nella prima colonna l'indice identifica
% la parola nella matrice RXwords, nella seconda colonna c'è il valore della distanza in percentuale)
distH_perc_nrand_SPALLR = get_perc(Tail_seq, 'Risultati Simulazioni/Risultati 3000000 non rand SPA-LLR/Rx_Codeword.txt');

disp("Utilizzando il decoder SPA-LLR, la distanza minima dalla Tail Sequence nel caso randomizzato è pari a "+num2str(min(distH_perc_rand_SPALLR(:,2)))+ ...
    ", mentre nel caso non randomizzato la distanza è pari a "+num2str(min(distH_perc_nrand_SPALLR(:,2))));

distH_perc_rand_MS = get_perc(Tail_seq_rand, 'Risultati Simulazioni/Risultati Simulazione 3000000 MS/Rx_Codeword.txt');
% distanza in percentuale dalla tail sequence randomizzata di tutte le parole di codice decodificate (nella prima colonna l'indice identifica
% la parola nella matrice RXwords, nella seconda colonna c'è il valore della distanza in percentuale)
distH_perc_nrand_MS = get_perc(Tail_seq, 'Risultati Simulazioni/Risultati 3000000 non rand MS/Rx_Codeword.txt');

disp("Utilizzando il decoder MinSum, la distanza minima dalla Tail Sequence nel caso randomizzato è pari a "+num2str(min(distH_perc_rand_MS(:,2)))+ ...
    ", mentre nel caso non randomizzato la distanza è pari a "+num2str(min(distH_perc_nrand_MS(:,2))));

distH_perc_rand_NMS = get_perc(Tail_seq_rand, 'Risultati Simulazioni/Risultati Simulazione 3000000 NMS/Rx_Codeword.txt');
% distanza in percentuale dalla tail sequence randomizzata di tutte le parole di codice decodificate (nella prima colonna l'indice identifica
% la parola nella matrice RXwords, nella seconda colonna c'è il valore della distanza in percentuale)
distH_perc_nrand_NMS = get_perc(Tail_seq, 'Risultati Simulazioni/Risultati 3000000 non rand NMS/Rx_Codeword.txt');

disp("Utilizzando il decoder Normalised MinSum, la distanza minima dalla Tail Sequence nel caso randomizzato è pari a "+num2str(min(distH_perc_rand_NMS(:,2)))+ ...
    ", mentre nel caso non randomizzato la distanza è pari a "+num2str(min(distH_perc_nrand_NMS(:,2))));
```

Figura A.6: Parte 6 dello script del main

A.2 Funzione per il Calcolo della Distanza di Hamming fra parole ricevute e la Tail Sequence

Questa funzione richiama la funzione che calcola le occorrenze delle parole ricevute (*countoccorrenze*) e calcola la distanza di Hamming delle singole parole di codice dalla Tail Sequence (si veda FiguraA.7).

```
function [distH] = get_dist(seq, filename)
%Questa funzione identifica la distanza di Hamming fra le parole ricevute e
%la sequenza data per terminare la decodifica

[Mu, ic]= count_occorrenze(filename);

ic = unique(ic, 'stable');

s = size(Mu);

for i = 1:s(1)
    ind = ic(i);
    distH(i,1) = sum (xor(Mu(i,:), seq));
end

distH = [ic distH];
```

Figura A.7: Funzione per calcolare la distanza di Hamming fra parole ricevute e la sequenza data

A.3 Funzione per il Calcolo delle Occorrenze delle Parole di Codice

Questa funzione richiama la funzione *scriptmatriceH* e conta le occorrenze delle singole parole di codice (si veda FiguraA.8).

```
function [Mu, ic] = count_occorrenze(filename)
%Script per graficare l'occorrenza delle diverse parole di codice, variando
%e non variando il valore di Eb/N0

[RXwords, EbN0] = script_matriceH(filename);

str_ebn0 = ["Eb/N0:  0.0000E+00", "Eb/N0:  1.0000E+00","Eb/N0:  2.0000E+00", "Eb/N0:  3.0000E+00", "Eb/N0:  4.0000E+00", ...
    "Eb/N0:  5.0000E+00", "Eb/N0:  6.0000E+00", "Eb/N0:  7.0000E+00"];
fin = 0;

% Mu = matrice unica, %ia = indici delle colonne di RXwords, %ic = numeri
% di ripetizioni delle righe di RXwords

[Mu,ia,ic] = unique(RXwords, 'rows', 'stable');
h = accumarray(ic, 1);  % Conta le occorrenze
s = size(Mu);
bar(1:s(1), h); %definisce l'istogramma l'istogramma
title("Istogramma Occorrenze non differenziato per EbN0");
xlabel('RX Word Nr.'); ylabel('Occorrenze');


for i = 1:length(EbN0)

    [Mu_ebn0, ia_ebn0, ic_ebn0] = unique(ic(1+fin:EbN0(i)+fin), 'rows', 'stable');

    h_ebn0 = accumarray(ic_ebn0,1);

    figure;
    bar(Mu_ebn0, h_ebn0); %definisce l'istogramma l'istogramma
    title("Istogramma Occorrenze per " + str_ebn0(i));
    xlabel('RX Word Nr.'); ylabel('Occorrenze');

    fin = fin+EbN0(i);
 end
```

Figura A.8: Funzione per calcolare le occorrenze delle singole parole di codice

A.4 Script per la Verifica delle Parole Ricevute

In questo script vengono richiamate le funzioni *getrxwords* e *matriceH* e viene calcolata la sindrome di ogni parola ricevuta per verificare che sia una parola di codice (si veda Figura A.9).

```
function [RXwords, EbN0] = script_matriceH(filename)

%Questo script estrae le parole ricevute dal file di testo creato durante
%la simulazione  e verifica che siano effettivamente parole di codice

[RXwords, EbN0] = get_rx_words(filename);

H = matriceH();
check_ind = 0;

s= size(RXwords);

for i = 1:s(1)
    c = RXwords(i,:);
    p = mod(c*H',2);
    if (isequal(p,zeros(1,64)==false))
        disp ('La '+ num2str(i) + '-esima parola ricevuta non è una parola di codice')
        check_ind = 1;
    end
end

if(check_ind == 0)
    disp('Tutte le parole ricevute sono parole di codice');
end
```

Figura A.9: Script per verificare che le parole ottenute siano effettivamente parole di codice

A.5 Funzione per Estrazione Parole Ricevute dal File di Testo

Con questa funzione vengono estratte le parole ricevute dal file di testo riportato in figura 3.1 (si vedano Figure A.10 e A.11).

```
function [Rx_words,EbN0] = get_rx_words (file_name)

file = fopen(file_name, 'r');

cell_str = textscan(file,'%s',  'Delimiter', '\n');

fclose(file);

ind = 1;
EbN0 = zeros(1,8);
lun = length(cell_str{1,1});

for i = 1:lun
    if strcmp(cell_str{1,1}(i),"Eb/N0:  0.0000E+00")
       EbN0(1) = EbN0(1) + 1;
    end
    if strcmp(cell_str{1,1}(i),"Eb/N0:  1.0000E+00")
       EbN0(2) = EbN0(2) + 1;
    end
    if strcmp(cell_str{1,1}(i),"Eb/N0:  2.0000E+00")
       EbN0(3) = EbN0(3) + 1;
    end
    if strcmp(cell_str{1,1}(i),"Eb/N0:  3.0000E+00")
       EbN0(4) = EbN0(4) + 1;
    end
    if strcmp(cell_str{1,1}(i),"Eb/N0:  4.0000E+00")
       EbN0(5) = EbN0(5) + 1;
    end
    if strcmp(cell_str{1,1}(i),"Eb/N0:  5.0000E+00")
       EbN0(6) = EbN0(6) + 1;
    end
    if strcmp(cell_str{1,1}(i),"Eb/N0:  6.0000E+00")
       EbN0(7) = EbN0(7) + 1;
    end
```

Figura A.10: Parte 1 della Funzione per Estrarre le Parole Ricevute dal File di Testo

```
    if strcmp(cell_str{1,1}(i),"Eb/N0:  7.0000E+00")
       EbN0(8) = EbN0(8) + 1;
    end
    if strcmp(cell_str{1,1}(i),"Rx Codeword: ")
        RX_String(ind,1) = cell_str{1,1}(i+1);
        ind = ind + 1;
    end
end

Rx_words = [];

for n = 1 : length(RX_String)
    s = RX_String{n};
    word = s - '0';
    Rx_words(n, :) = word;
end
```

Figura A.11: Parte 2 della Funzione per Estrarre le Parole Ricevute dal File di Testo

A.7 Toy Example

In questa sezione è riportato lo script implementato per riprodurre l'esempio dimostrativo inserito in [8] (si veda Figura A.13). In questo script è richiamata la funzione *tailsequencesearch* (si veda Figura A.14) che a sua volta richiama la funzione *findflipposition* (si vedano Figure A.15 e A.16).

```
function [v] = tail_sequence_search (RX, v_init, d_init)
% v = tail sequence in uscita
% RX = matrice di parole di codice da cui distanziarsi; v_init = vettore di partenza; d_init = distanza di inizializzazione, pongo inizialmente ad un valore alto
%per convenzione il vettore iniziale alla prima chiamata della funzione è uguale ad un vettore con un 1 sequito da n 0 a seconda di quanto si vuole lungo il vettore

s = size(RX);

for i = 1:s(1)
 dist(i,1) = (sum (xor(RX(i,:), v_init)));
end

d_min = min(dist);

if (d_min < d_init)
    d_init = d_min;
    r_ind = size(dist);
    index = 1;

    for n = 1:r_ind(1)
        if dist(n) == d_min
            row(index) = n;
            index = index+1;
        end
    end

    pos = findflip_position(v_init, row, RX);

    if pos ~= 0
        v_init(1,pos) = xor(v_init(1,pos),2);
        v = tail_sequence_search(RX,v_init, d_init);
    else
        v = v_init;
    end
else
    v = v_init;
end
```

Figura A.13: Script Toy Example

A.6 Funzione per Generazione Matrice di Parità del codice LDPC

Questa funzione genera la matrice di parità (si veda Figura A.12)

```
function [MatriceH_LDPC_128_64_2] = matriceH()

fID = fopen('NASA_LDPC_128_64.txt','r');
text = textscan(fID, '%s', 'Delimiter', '\n');
fclose(fID);

H = text{1};

MatriceH_LDPC_128_64_1 = zeros (64,128);
MatriceH_LDPC_128_64_2 = zeros (64,128);

for i = 1:128
    pos_1 = str2num(char(H(i,1)));
    for r = pos_1
        MatriceH_LDPC_128_64_1(r,i) = 1;
    end
end

for i = 129:192
    pos_1 = str2num(char(H(i,1)));
    for r = pos_1
        MatriceH_LDPC_128_64_2(i-128,r) = 1;
    end
end

Compare = isequal(MatriceH_LDPC_128_64_1,MatriceH_LDPC_128_64_2);
if (Compare == false)
    disp('Matrici Diverse');
end
end
```

Figura A.12: Funzione per generare la matrice di parità del codice LDPC

A.7.1 Funzione per la Ricerca della Tail Sequence

```
function [v] = tail_sequence_search (RX, v_init, d_init)
% v = tail sequence in uscita
% RX = matrice di parole di codice da cui distanziarsi;
% v_init = vettore di partenza;
% d_init = distanza di inizializzazione, pongo inizialmente ad un valore alto
%per convenzione il vettore iniziale alla prima chiamata della funzione è uguale
% ad un vettore con un 1 sequito da n 0 a seconda di quanto si vuole lungo il vettore

s = size(RX);

for i = 1:s(1)
 dist(i,1) = (sum (xor(RX(i,:), v_init)));
end

d_min = min(dist);

if (d_min < d_init)
    d_init = d_min;
    r_ind = size(dist);
    index = 1;

    for n = 1:r_ind(1)
        if dist(n) == d_min
            row(index) = n;
            index = index+1;
        end
    end

    pos = findflip_position(v_init, row, RX);

    if pos ~= 0
        v_init(1,pos) = xor(v_init(1,pos),2);
        v = tail_sequence_search(RX,v_init, d_init);
    else
        v = v_init;
    end
else
    v = v_init;
end
```

Figura A.14: Funzione per la Ricerca della Tail Sequence

A.7.2 Funzione per la Ricerca della Coordinata del bit da Complementare

```
function pos = findflip_position(v_init, row, RX)

%con questa funzione vado a calcolare la posizione del primo bit che complementandolo non mi restituisce un vettore nullo

vect = RX(row, :); %seleziono solo le righe corrispondendi alle parole di codice che si trovano a distanza minima
s= size(vect);

for i = 1:s(1)
    compl_v(i,:) = xor(xor(vect(i,:),v_init),2); % trovo per ogni parola di codice i bit che si possono complementare
end

for i = 1:length(row)
        pos_mat(i,:) = find(compl_v(i,:)); % salvo le posizioni dei bit che si possono complementare in una matrice
end

pos_s = size(pos_mat); pos_ind = 0;
zero_pos = []; %inizializzo il valore a 0
index = 0; %inizializzato a zero e viene mantenuto a 0 fintanto che si trova lo stesso indice su ogni vettore
ind = 2; % inizializzando con la seconda riga della matrice
```

Figura A.15: Script Toy Example

```
if(pos_s(1)>1)
    for i = 1:pos_s(2) %verifico se ho indici in comune
        val = pos_mat(1,i);
        while index == 0 && ind ~= pos_s(1)+1
            zero_pos = find(~bitxor(pos_mat(ind,:),val));
            if isempty(zero_pos) == true
                index = 1;
                pos_ind = zero_pos;
            else
                ind = ind+1;
                pos_ind = zero_pos;
            end

        end
        index = 0;
    end
    if isempty(pos_ind) == false
        pos = pos_mat(1,zero_pos);
    else
        pos = 0;
    end
else
    pos = pos_mat(1,randi(length(pos_mat))); %seleziono in maniera random il bit da invertire
end
```

Figura A.16: Parte 2 della Funzione per la Ricerca della Coordinata del bit da Complementare

Bibliografia

[1] Andrea Bertuccini. Analysis of the tail sequence false alarm probability, 2021. Master thesis at Università Politecnica delle Marche, Ancona, Italy.

[2] METEORED-ITALIA. Quanti satelliti orbitano attorno alla terra e come è possibile che non si scontrino? Available online. https://www.ilmeteo.net/notizie/scienza/quanti-satelliti-orbitano-attorno-alla-terra-e-come-e-possibile-che-non-si-scontrino.html.

[3] IAP Italy. Telecomunicazioni Satellitari. Available online. https://iap-italy.it/applicazioni-satellitari/telecomunicazioni-satellitari-satnav/.

[4] Ministero della Difesa. Comunicazione Satellitare (SatCom). Available online.

[5] CCSDS. Consultative Commitee for Space Data Systems. Available online. https://public.ccsds.org/default.aspx.

[6] CCSDS The Consultative Commitee for Space Data Systems. *TC Synchronization and Channel Coding. Issue. Recommendation for Space Data System Standard (Blue Book).* National Aeronautics and Space Administration. Blue Book CCSDS 231.0-B-4 Washington, DC, USA, July 2021.

[7] CCSDS The Consultative Commitee for Space Data Systems. *TC Synchronization and Channel Coding - Summary of Concept and Rationale. Informational Report. (Green Book).* National Aeronautics and Space Administration. Green Book CCSDS 230.1-G-3 Washington, DC, USA, October 2021.

[8] Kenneth Andrews. Tail Sequence Search for the (128,64) Uplink LDPC Code. Contribution to the CCSDS Meeting, 2017. Jet Propulsion Laboratory, California Institute of Technology.