UNIVERSITA' DEL SALENTO

Corso di Laurea in Ingegneria dell'informazione

Tesi di Laurea in Fondamenti di Comunicazioni

Codifica di canale tramite l'utilizzo di codici concatenati

Channel coding through concatenated codes

RELATORE:
Ch.mo Prof. Francesco Bandiera

STUDENTE:
Stefano Rainò
Matricola n° 20034127

Anno Accademico 2019 -2020

Indice

Prefazione

In questo elaborato di tesi, si proverà a mettere in pratica la teoria alla base dei processi di codifica di canale, al fine di poter visionare come codificare opportunamente una sequenza in ingresso e poter rilevare e correggere eventuali errori che possono presentarsi nel processo di trasmissione.

Il capitolo 1 è dedicato all'introduzione della teoria alla base delle principali codifiche utilizzate per la trasmissione su un canale di comunicazione, successivamente, si cercherà di descrivere in maniera opportuna i concetti teorici delle codifiche a blocco e cicliche, utilizzate adeguatamente per i processi rappresentativi dei codici convoluzionali e Reed-Solomon. In particolare, si dimostrerà come molti dei codici di interesse pratico siano sinteticamente descritti da un polinomio generatore.

Il capitolo 2 punterà invece a descrivere il funzionamento dei codici convoluzionali, che hanno dominato la scena per molti anni, e a introdurre la rappresentazione del processo di codifica tramite diagramma degli stati e/o trellis, che rappresenta un'estrazione di un grafo transazionale, ai quali è associata una funzione di trasferimento opportunatamente descritta e utilizzata al fine di riuscire a individuare il numero di passi da percorrere all'interno del diagramma degli stati, dalla prima diramazione, per poter riconciliarsi con la sequenza nulla associata alla codifica. In secondo luogo, si cercherà di visionare una particolare applicazione dei codici convoluzionali, relativi all'utilizzo di specifiche sequenze di ingresso, che generano delle codifiche catastrofiche aventi la particolarità di avere la distanza minima di Hamming, tra parole di codice codificate, che non cresce al crescere di tale distanza nelle sequenze di ingresso al codificatore. Inoltre, al fine di dare un risvolto pratico all'analisi teorica, si implementerà attraverso l'utilizzo di alcune funzioni in linguaggio MATLAB, la codifica

convoluzionale precedentemente descritta cercando di commentare opportunamente l'intero processo di rappresentazione.
Il capitolo 3 introdurrà invece la teoria alla base dello sviluppo dei codici di Reed-Solomon, i quali fanno parte della classe di *codici di Bose-Chaudhuri-Hocquenghem (BCH)*. Questa classe di codici è utilizzata per la correzione di errori casuali, e sfruttano algoritmi di decodifica non troppo complessi volti a riuscire a garantire, nel processo di decodifica associato, di scovare e correggere un numero limitato di errori. Oltre a questo, tali codici sono utili in quanto consentono una notevole flessibilità nella scelta dei parametri quali lunghezza del blocco e rapporto di codifica. All'interno del capitolo si cercherà di descrivere in maniera esaustiva le proprietà associate ai campi di Galois utilizzate per introdurre il polinomio generatore del codice, il quale permetterà di garantire le proprietà di correzione citate prima. Anche in questo caso, si implementerà in MATLAB un opportuno algoritmo di codifica scalabile che si pone come obiettivo quello di codificare opportunamente qualsiasi sequenza in ingresso a seguito della definizione dei parametri generatori del campo di Galois.
Il capitolo 4 è invece dedicato all'implementazione pratica relativa all'utilizzo di una coppia di codici, nello specifico Reed-Solomon e convoluzionale, volti a realizzare concatenazioni tipiche di processi di codifica avanzati che permettono di incapsulare una sequenza codificata in ingresso in modo tale da poter correggere una quantità di errori elevata a seguito dell'inoltro sul canale di trasmissione. Infine, una breve appendice riporterà il risultato relativo all'output fornito dai vari algoritmi MATLAB utilizzati per l'implementazione del processo di codifica.

Capitolo 1: Trasmissioni mediante l'utilizzo di forme d'onda codificate

Il sistema di telecomunicazione digitale tende oramai all'utilizzo di trasmissioni numeriche. Tale sistema di telecomunicazione permette di avere una certa robustezza e affidabilità per quanto riguarda la presenza di distorsioni additive introdotte dal canale di comunicazione stesso su di esso. Questo effetto di distorsione del canale fa sì che il flusso binario di dati da trasportare possa contenere errori. Dunque, la bontà di un sistema di trasmissione numerico viene valutata attraverso una stima della cosiddetta *probabilità di errore,* che rappresenta uno dei più importanti parametri da valutare al fine di ottenere un'ottima realizzazione di una rete comunicativa. Le tecniche di gestione di tale probabilità richiedono che la sequenza di informazione in ingresso risulti essere *ridondante* in modo tale che essa possa ripetere il messaggio informativo più volte sullo stesso canale trasmissivo. Ovviamente il parametro relativo alla ridondanza di un codice, per essere considerato affidabile, deve essere messo a confronto con l'efficienza trasmissiva di quest'ultimo. Per mezzo di tale fattore introdotto dall'operazione di ridondanza, alla sequenza trasmissiva della sorgente binaria, viene aggiunto un numero arbitrario di bit i quali fanno sì che, ad esempio, la parola venga rappresentata non più da *k bit*, bensì da *n bit* ($\boldsymbol{n > k}$). Di conseguenza le 2^k possibili parole di codice della sorgente binaria vengono mappate non più su *k*, ma su *n* bit. Il dispositivo che effettua tale mappatura è chiamato *codificatore di canale*. Se la codifica eseguita da tale codificatore è senza memoria, il codice risultante al termine della trasmissione è un *codice a blocco,* al contrario qualora tale processo avvenga con memoria, il codice risultante sarà un *codice ad albero.* Oltre a questa classificazione preliminare, tali codici possono essere anche classificati in codici a *rivelazione d'errore* e codici a *correzione di errore.* Questa seconda classificazione riguarda principalmente la modalità di

decodifica della sequenza in ingresso che avviene mediante l'utilizzo di un *decodificatore di canale.*

Il processo introdotto precedentemente con il termine di *codifica* dell'informazione è il processo che permette di rappresentare l'informazione trasmessa da un segnale su più simboli. Esistono tre principali tipi di codifica nell'ambito delle telecomunicazioni ovvero:

- La *codifica di sorgente* che viene effettuata qualora si volesse effettuare una compressione mirata a ridurre la ridondanza dei bit della sorgente; inoltre, tale codifica permette di avere lo stesso contenuto informativo andando ad utilizzare un minor numero di simboli.
- La *codifica di canale* che al contrario della codifica di sorgente mira alla generazione di un flusso di simboli non indipendenti andando così ad introdurre ridondanza nella sequenza trasmessa e a diminuire il livello di informazione fornita.
- La *codifica del contenuto* che viene utilizzata invece quando si ha la necessità di nascondere il contenuto informativo di un messaggio trasmesso, in modo tale da impedire l'accesso non autorizzato ai contenuti trasmessi.

I codici di cui si discuterà in questo trattato rientrano nella categoria di codifiche di canale, in particolar modo verranno sviscerati i codici a correzione dell'errore precedentemente menzionati.

1.1 Introduzione

Come detto precedentemente, la codifica di canale volta alla correzione o rivelazione dell'errore, avviene mediante l'esecuzione di un'operazione di ridondanza, la quale aggiunge alla parola binaria, relativa alla sorgente trasmessa, una quantità "*controllata*" di bit aggiuntivi. Questa operazione fa sì che l'algoritmo di decodifica, a cui verrà sottoposta la parola trasmessa, subirà una mole di operazioni aggiuntive volte al recupero dell'effettivo contenuto informativo. Per quanto riguarda il processo di rivelazione dell'errore, il decodificatore elimina una parte di indeterminazione relativa alla sequenza di bit ricevuta. Questa parte corrisponde al termine $H(e)$ presente nella *disuguaglianza di Fano* ($H(X|Y) \leq H(e) + P(e)log\ (N-1)$) che corrisponde alla quantità di informazione necessaria per sapere se si è verificato o meno un errore durante la trasmissione dei dati. Tale procedura è in genere utilizzata per la realizzazione di due sistemi particolari: *indicazione di errore* o *richiesta automatica di ripetizione*. Nel primo caso, viene fornita continuamente una stima sulla qualità della sequenza ricevuta, nel secondo caso invece, vengono ripetute le trasmissioni di eventuali scambi di informazioni falliti.

La seconda strategia riguardante la correzione dell'errore (*FEC, dall'inglese Forward Error Correction*) permette invece di andare ad eliminare un'ulteriore quantità di indeterminazione relativa al termine $P(e)\log\,(N-1)$ presente, anch'esso, nella disuguaglianza di Fano. Questo metodo è utilizzato quando si vuole ripristinare la sequenza effettivamente trasmessa ogni volta che si rivelano errori.

L'immagine sottostante illustra come avviene una effettiva codifica di canale in tutti i suoi passaggi fondamentali:

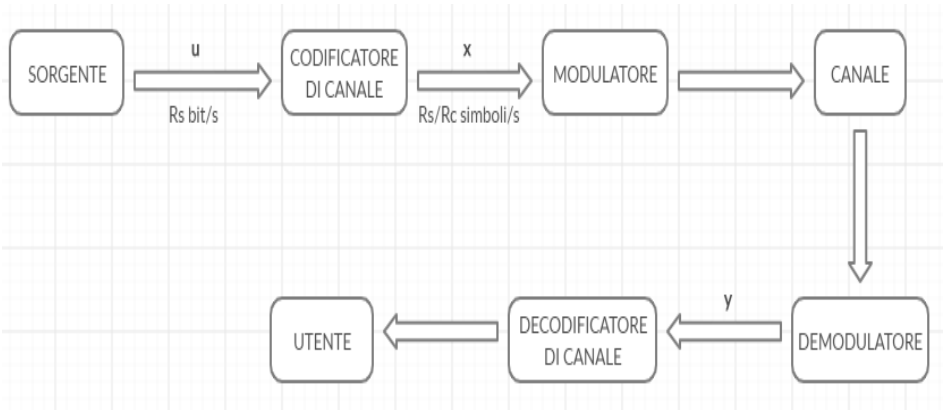


Figura 1: schema a blocchi del processo di codifica

Parafrasando il grafico, dalla sorgente vengono emesse cifre binarie con frequenza pari a $R_s\ bit/s$, successivamente, il codificatore rappresenta le cifre emesse dalla sorgente prendendole a gruppi di k e utilizzando a sua volta $n = {}^{k}/_{R_c}$ cifre binarie, dove è indicata con R_c la frequenza di codifica. Questa modalità di trasmissione necessita di essere aumentata fino ad un valore tipico di ${}^{R_s}/_{R_c}$ simboli al secondo, questo aumento fa sì che venga ridotta l'efficienza di banda per poter privilegiare la trasmissione non codificata. Successivamente i simboli uscenti dal codificatore sono presentati al modulatore e vengono da esso convertiti in delle forme d'onda utilizzando una particolare tipologia di modulazione specifica per ogni singola trasmissione. Supponendo che nel caso di esempio venga utilizzata una modulazione binaria antipodale, con questo tipo di modulazione, ogni simbolo opportunamente codificato è associato ad una forma d'onda di durata $T = T_c = {}^{R_s}/_{R_c}$ secondi. Pertanto indicando con ξ_b l'energia media per bit, che immaginando venga mantenuta costante durante tutto il processo di trasmissione, si può facilmente verificare che la codifica diminuisce l'energia per simbolo ad un valore tipico di $\xi = \xi_b \cdot R_c$, pertanto si avranno

innumerevoli simboli di canale trasmessi in modo errato qualora la trasmissione avvenisse senza alcun processo di codifica opportuno.

Osservando tale processo si può notare che trasmissioni di questo tipo devono scendere a degli opportuni compromessi per quanto riguarda l'efficienza di utilizzo e la riduzione delle varie probabilità di errore, andando ad utilizzare però la medesima potenza trasmissiva. Di conseguenza si possono distinguere tre principali modelli di trasmissione, chiamati modelli di *"decisione"* che permettono di descrivere tre diversi tipi di trasmissioni codificate, ovvero:

- *Modello di codifica a decisione "hard"* che è un modello che considera il caso in cui un eventuale demodulatore, preliminarmente, prende una decisione sull'effettiva corrispondenza tra 0 o 1 di ogni forma d'onda binaria. Di conseguenza si avrà che l'uscita del modulatore è quantizzata su due livelli indicati con 0 ed 1, la sequenza di cifre binarie in uscita da tale demodulatore ricostruisce l'informazione utilizzando algoritmi ridondanti o per rivelare o per correggere errori presenti all'uscita di esso. Tali algoritmi combinano modulatore, canale e demodulatore al fine di creare un *canale binario simmetric*o *(BSC)*. Le prestazioni in relazione all'utilizzo di questo modello dipendono appunto dalla prestanza di tali algoritmi.
- *Modello di codifica non quantizzata a decisione "soft"* che è un modello in cui l'uscita del demodulatore è non quantizzata ed essa viene inviata direttamente al decodificatore. Esso permette di ricostruire dalle *n* uscite, corrispondenti ad *n* differenti forme d'onda binarie, 2^k variabili di decisione. Tale processo permette di andare a scegliere una fra le 2^k possibili sequenze trasmesse, la quale sia maggiormente somigliante all'effettiva sequenza informativa. Ovviamente, tale approccio offre una maggiore affidabilità rispetto

all'approccio descritto precedentemente, dato che mediante l'utilizzo di questo modello decisionale, si possono trarre diversi vantaggi trasmissivi dalla informazione aggiuntiva contenuta in campioni non quantizzati.

- *Modello di codifica quantizzata a decisione "soft"* che è un modello dove il demodulatore quantizza l'uscita su Q livelli con $Q>2$. In questo caso, dunque, la combinazione tra modulatore e canale è la medesima configurazione che si presenta quando si ha un canale discreto con due ingressi e Q uscite. L'algoritmo di decodifica sfruttato da questo modello decisionale si basa su di una manipolazione dei simboli uscenti dal canale per rappresentare la sequenza binaria che trasporta l'informazione in una particolare forma d'onda. Il vantaggio rispetto al caso precedente è che tutta l'elaborazione del segnale informativo può essere effettuata mediante circuiti numerici.

Questi sistemi di trasmissione codificata hanno il vantaggio rispetto ai sistemi non codificati di avere un alto *guadagno di codifica*. Questa quantità è definita come la differenza (in decibel) relativa al valore di ξ_b/N_0 necessario al fine di avere una probabilità di contrarre errori di trasmissione rispetto ad una trasmissione binaria antipodale ideale e non codificata. Concludiamo questo breve preambolo introduttivo osservando che le trasmissioni codificate utilizzando uno dei tre metodi sopra citati permettono di avvicinare la capacità del canale al valore limite ξ_b/N_0 per valori molto piccoli in corrispondenza di regioni aventi una potenza verosimilmente limitata. Il proposito di questo capitolo è dunque quello di descrivere le caratteristiche prestazionali di questi sistemi di codifica che cercano di ottenere guadagni di codifica teorici. Per far questo si svilupperà il concetto di *codici ad albero* e *codici a blocco* che permettono di effettuare

o meno un processo di acquisizione dati per l'operazione di codifica. Solo una particolare classe di essi verrà esaminata e implementata in modo tale da sviscerarne i principi di funzionamento, ovvero i cosiddetti *codici convoluzionali.*

1.2 Codici a blocco

La base dell'utilizzo di una codifica a blocchi è quello di, ad esempio, prendere un blocco di *n* cifre binarie codificate e fare riferimento solo ad un corrispondente blocco di *k* cifre della sorgente emessa. Questa particolare codifica è dunque una codifica senza memoria. La teoria dell'utilizzo dei codici a blocco è rappresentata dall'operazione di *controllo della parità.* Quest'operazione prende una sequenza di *k* cifre binarie e aggiunge in ultima posizione una nuova cifra, tenendo conto però che a seguito di questa nuova aggiunta, la nuova sequenza di uno presente nella parola deve risultare in numero pari. In questo modo ogni qualvolta vi sia la presenza di un errore, che cambia la parità della sequenza binaria, esso è rivelato dal decodificatore.

Questa operazione appena descritta, viene utilizzata in diverse classi particolari di codici a blocco, dato che, in un codice a blocco la parola binaria è costituita da un insieme di *n* controlli di parità, eseguiti su tutte le cifre (*k*) della sorgente informativa. Tale codice può assumere il significato di codice *sistematico* quando le prime *k* cifre sono una replica delle cifre della sorgente binaria e le restanti $n - k$ (con *n* lunghezza effettiva della parola decodificata) rappresentano dei controlli di parità sulle *k* cifre di informazione. Un esempio di un codificatore a blocchi funzionale è rappresentato dalla figura sottostante:

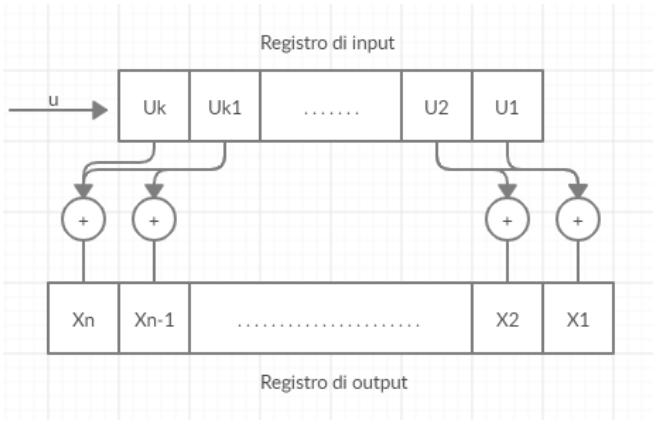


Figura 2: registri di un codificatore a blocco

Esso è rappresentato da un registro a scorrimento con *k* stadi in ingresso, *n* sommatori, e un registro a scorrimento con *n* stadi in uscita. Ovviamente ogni sommatore è collegato ad un sottoinsieme di registri che eseguono il controllo di parità, ovvero: il registro in ingresso viene caricato con le *k* cifre e, quando tale registro viene caricato, ogni uscita del sommatore è inviata serialmente all'ingresso di ogni singolo blocco del registro di uscita, che emette le parole di codice relative alla sorgente codificata. Nel mentre viene emessa una parola il registro in ingresso viene ricaricato in modo tale da poter ripetere la sequenza di operazioni, calibrando in maniera differente la frequenza di uscita delle parole tra i due registri.

In questo tipo di codici l'informazione da dover codificare può essere rappresentata sotto forma di matrice. Si definisce con il termine di *matrice binaria* **G** di dimensione $k \times n$ una matrice dove l'elemento (i, j) è uno se e solo se l'i-esima cifra posizionale del registro in input è connessa al j-esimo sommatore, in tutti gli altri casi sarà zero. Questa matrice **G** sarà chiamata *matrice generatrice del codice a blocco.* Utilizzando una notazione vettoriale si può indicare una generica sequenza codificata *x* come: $\boldsymbol{x} = \boldsymbol{u} \cdot \boldsymbol{G}$, dove *u* è il vettore corrispondente alla sorgente binaria da codificare. Un metodo veloce per ottenere la parola codificata è quello di

sommare in aritmetica binaria le righe della matrice **G** che corrispondono agli uno della sorgente informativa. Ad esempio, se nella sequenza informativa le cifre u_i, u_j e u_l sono uno e tutte le altre zero, allora la parola di codice codificata si ottiene sommando in aritmetica binaria le righe della matrice generatrice corrispondenti agli indici *i, j* e *l*.

Consideriamo ora a titolo di esempio un codice sistematico definito dalle relazioni:

$$x_i = u_i, i = 1,2,3,4$$
$$x_5 = u_1 + u_2 + u_3$$
$$x_6 = u_2 + u_3 + u_4$$
$$x_7 = u_1 + u_2 + u_4$$

La matrice generatrice **G** può essere ottenuta sfruttando le relazioni descritte sopra tra cifre del registro di input e cifre del registro di output. Tale matrice risulta essere:

$$\mathbf{G} = \begin{array}{c} \begin{matrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \end{array} \begin{matrix} u1 \\ u2 \\ u3 \\ u4 \end{matrix}$$

Dunque qualora si volesse la parola di codice alla quale corrisponde la sequenza di input ***u****=(1011)* si procederà nel sommare la prima, con le due ultime righe di **G** e si avrà come risultato:

1000101 +

0010110 +

<u>0001011 =</u>

1011000

Quando un codice è sistematico, la matrice **G** ha la forma: **G = [Ik : P]**, in questo caso, **Ik** rappresenta la matrice identità di dimensioni $k \times k$ mentre **P** è una matrice di dimensioni $k \times (n - k)$ che contiene l'informazione riguardo ai codici a controllo della parità, ovviamente una volta conosciuta la matrice **P** si possono definire le regole di codifica per un codice sistematico. I codici a controllo di parità hanno cinque diverse proprietà che devono essere rispettate:

Proprietà 1: Ogni parola codificata è una somma di righe della matrice generatrice.

Proprietà 2: Il codice a blocco è l'insieme di tutte le combinazioni di somme tra le righe della matrice generatrice.

Proprietà 3: Sommare due parole già codificate darà vita ad una nuova parola codificata.

Proprietà 4: La sequenza di tutti zeri è sempre una parola codificata a controllo di parità.

Proprietà 5: La distanza minima di un codice a blocco *lineare* è il peso minimo delle sue parole codificate non nulle.

Per quanto riguarda la comprensione di quest'ultima proprietà è necessario introdurre il concetto di *distanza di Hamming* (o più comunemente chiamata distanza). Qualora due sorgenti di codice informativo $\boldsymbol{x_i}$ e $\boldsymbol{x_j}$ differiscano in alcune posizioni per il bit ad esse associato nelle posizioni d'esame, saranno definite distanti di una quantità pari al numero di bit per le quali esse differiscono. Questa quantità è chiamata *distanza di Hamming* $d_{i,j}$. Ovviamente questa quantità sarà sempre compresa tra zero e il numero *n* di bit di cui sono composte le parole binarie. La più piccola di queste distanze, tra tutte le parole che formano il nostro codice a blocco, è chiamata *distanza*

minima del codice d_{min}. A seguito della stesura di queste proprietà, i codici a controllo di parità sono anche chiamati *codici a blocco lineari*. Tali codici sono interpretabili come un sottospazio dello spazio contenente tutte le *n-uple* binarie della matrice generatrice di un codice a blocco. Infatti, dal punto di vista algebrico, le righe della matrice **G** sono una base del sottospazio e contengono *k* parole di codice linearmente indipendenti, infatti, tutte le loro 2^k combinazioni lineari generano l'effettivo codice a blocco. Pertanto qualsiasi matrice generatrice di un codice a blocco, mediante operazioni algebriche elementari, può essere ridotta alla forma sistematica: **G = [Ik : P].** Ogni codice a blocco (n, k) può essere dunque considerato come un codice sistematico (n, k).

In una trasmissione mediante l'utilizzo di un codice a blocco sistematico, la sequenza ***x*** di cifre binarie trasmesse è ricevuta dal decodificatore in forma ***y*** dove *y* è sempre una sequenza di *k* cifre binarie associate con $(n - k)$ cifre di controllo. Questa sequenza può essere affetta da errori casuali, memorizzati all'interno di $e = [e_1, \cdots, e_n]$ che rappresenta il vettore d'errore che ad ogni sua componente è associato uno se l'i-esima cifra trasmessa è stata alterata, altrimenti varrà zero. Il compito del decodificatore sarà dunque quello di confrontare tutte le $(n - k)$ cifre ricevute, che sono relative al controllo di parità effettuato, con la combinazione lineare delle cifre che formavano il controllo di parità emesse dal trasmettitore. Se esse coincidono la parola ricevuta fa parte del codice. Il vettore di errore, dunque, racchiude un numero di uno pari al numero di volte che tale confronto da esito negativo. Se si chiama con ***s*** *(sindrome)* il vettore contenente la configurazione dei fallimenti, si può verificare che tale sindrome è ottenuta dall'equazione

$\boldsymbol{G \cdot H^T = 0}$**,** dove **H** è chiamata *matrice di controllo di parità* ed è definita come: $\boldsymbol{H = [P' : I_{n-k}]}$, che rappresenta una matrice $(n - k) \times n$ dove le righe rappresentano il risultato dei controlli di parità effettuati dal decodificatore.

Associate alla definizione di sindrome si possono evincere due proprietà importanti:

Proprietà 6: La sindrome di una sequenza *y* di bit decodificata è nulla se e solo se *y* è una parola di codice

Proprietà 7: Il decodificatore può rivelare tutti gli errori del canale sotto forma di vettori binari.

Queste due importanti proprietà, insieme alla caratteristica precedentemente introdotta riguardante il fatto che non esistono parole di codice aventi una distanza inferiore a d_{min}, permettono di verificare che un codice a blocco lineare avente d_{min} come distanza minima, ha la possibilità di rivelare tutti i vettori di errore con peso non superiore a $(d_{min} - 1)$. Usando le espressioni precedentemente ottenute per il processo di decodifica, si può riscrivere l'equazione riguardante il calcolo della sindrome come:

$\boldsymbol{s} = \boldsymbol{y} \cdot \boldsymbol{H^T} = (\boldsymbol{x} + \boldsymbol{e}) \cdot \boldsymbol{H^T}$, di conseguenza, vi saranno 2^k parole di codice codificate corrispondenti ad errori diversi, aventi la stessa sindrome. Per poter scoprire quale effettivamente tra questi errori è l'errore trasmesso a seguito della trasmissione di una parola, è necessario utilizzare un appropriato algoritmo di decodifica basato sulle probabilità di errore.

Un approccio di questo tipo può essere raggiunto mediante l'utilizzo di *decodifiche a massima verosimiglianza*. Questo tipo di approccio fa sì che a pari probabilità trasmissive di parole di codice, la parola che effettivamente sarà ricevuta, sarà la parola avente minima probabilità di errore. Di conseguenza, supponendo di effettuare una decisione di tipo hard, si avrà: $P(y|x_i) = p^{d_i}(1-p)^{n-d_i}$, dove *n* è la lunghezza del blocco, d_i è la distanza fra la sequenza ricevuta *y* e la i-esima parola trasmessa x_i, e *p* è la probabilità di avere una transizione equiprobabile su di un canale *BSC*.

Questa probabilità è una probabilità che descrive una funzione decrescente e monotona dato che $p < {}^{1}/_{2}$, pertanto si può concludere che l'algoritmo di decodifica a minima distanza permette di verificare se il vettore di errore *e*, effettivamente trasmesso durante la comunicazione, è il vettore avente peso minimo che è presente nel calcolo dei 2^k vettori di errore, che permettono di restituire una medesima sindrome associata ad una parola di codice ricevuta e da decodificare. Questo algoritmo di decodifica assegna ad ogni parola di codice una regione di decisione in cui sono presenti tutte le sequenze di parole ricevibili che sono più vicine ad essa rispetto alle altre. Pertanto, un codice a blocco lineare (n, k) avente come distanza minima d_{min}, può correggere al massimo $t = (d_{min} - 1)/2$ errori presenti all'interno di un medesimo vettore di errore che cade nella regione di decisione corretta.

Un obiettivo progettuale per un codice a blocco è dunque quello di utilizzare algoritmi ridondanti per ottenere una distanza minima più elevata possibile che, finora, non hanno un riscontro relativo all'utilizzo di soluzioni di tipo generalizzato.

1.3 Codici ciclici

I *codici ciclici* rappresentano dei particolari codici che effettuano controlli di parità, così come i codici a blocco descritti nella sezione precedente, di conseguenza ne condividono con essi tutte e sette le proprietà descritte. Oltre a queste, i codici ciclici presentano delle proprietà specifiche che permettono di effettuare facili operazioni di codifica. Si definisce un codice ciclico un codice a blocco lineare (n, k) tale per cui ogni traslazione ripetuta delle cifre binarie costituenti una parola di codice è anch'essa una parola di codice. A titolo di esempio possiamo verificare come il codice definito dalle relazioni: $x_i = u_i, i = 1,2,3,4;\ x_5 = u_1 + u_2 + u_3;\ x_6 =$

$u_2 + u_3 + u_4;\ x_7 = u_1 + u_2 + u_4$ (chiamato anche *codice di Hamming*), è un codice ciclico. Di fatti prendendo come esempio la parola di codice 1011000, tutte le possibili traslazioni cicliche di questa parola di codice, ovvero:

0110001 1100010 1000101 0001011 0010110 0101100

Appartengono tutte al paniere di parole di codice, e questo è vero per ogni singola parola di codice.

Manipolando questo tipo di codici, molto spesso, si utilizza una notazione polinomiale per la rappresentazione della sequenza binaria associata alla parola. Di conseguenza la struttura di un codice, e le proprietà relative ad esso, sono legate ad alcune proprietà polinomiali. Una parola di codice può essere rappresentata nel seguente modo:

$$x(D) = x_{n-1}D^{n-1} + x_{n-2}D^{n-2} + \cdots + x_1 D + x_0$$

I coefficienti del polinomio $x(D)$ sono tutti coefficienti binari e vi è una corrispondenza biunivoca tra polinomi di codice e parole del codice a blocco lineare (n, k). La notazione utilizzata, invece, per indicare una traslazione ciclica i-esima (a destra o a sinistra) di una parola di codice associata ad un polinomio $x(D)$, rispetto alla prima cifra, è: $x^{(i)}D$. Questi tipi di traslazioni cicliche sono regolate da un'importante proprietà applicativa. Infatti, si può verificare come il polinomio di codice $x^{(i)}D$ rappresenta il resto della divisione di $D^i x(D)$ per $(D^n - 1)$, in altri termini si avrà che: $D^i x(D) = q(D)(D^n + 1) + x^{(i)}(D)$. Il polinomio $q(D)$ in questo caso rappresenta il quoziente della divisione il cui grado non è superiore a $(i - 1)$. Per dimostrare ciò si prenderà in esame la parola di codice dell'esempio precedente: 1011010. Il polinomio ad essa associato è: $x(D) = D^5 + D^3 + D^2 + 1$. Traslando questa parola di codice tre volte verso sinistra avremo generato il polinomio $x^{(3)}(D)$, il quale è possibile ottenerlo andando a dividere $D^3 x(D)$ per $(D^7 + 1)$ in questo modo:

D^7+1	$D^8+D^6+D^5+D^3$	D
	D^8+ $\quad D$	Quoziente
	$D^6+D^5+D^3+D$	Resto

Il resto in questo caso sarà rappresentato dal polinomio: $D^6 + D^5 + D^3 + D$ (ovvero trasformato in sequenza binaria sarà 0101011). Questo polinomio rappresenta una traslazione ciclica di tre posizioni verso sinistra del polinomio generatore originario.

Un altro importante risultato riguardante la manipolazione di codici ciclici è basato sulle loro matrici generatrici. Infatti si può dimostrare come, dato un codice ciclico del tipo (n, k), esiste uno ed un solo polinomio tale per cui vale la seguente uguaglianza: $g(D) = D^{n-k} + \cdots + 1$, inoltre, i restanti $2^k - 1$ polinomi appartenenti al codice sono multipli di $g(D)$ e tutti gli altri polinomi avente grado $(n-1)$ o inferiore, divisibili per $g(D)$, sono dei polinomi di codice. La dimostrazione analitica di questo risultato può essere effettuata considerando il codice di Hamming precedentemente utilizzato, il quale è già stato classificato come codice ciclico. Riscrivendo la sua matrice generatrice in forma polinomiale si avrà:

$$\boldsymbol{G(D)} = \begin{bmatrix} D^6+ & & & & D^2+ & & 1 \\ & D^5+ & & & D^2+ & D+ & 1 \\ & & D^4+ & & D^2+ & D+ & \\ & & & D^3+ & & D+ & 1 \end{bmatrix}$$

Se si consider ora la penultima riga di questa matrice generatrice, ovvero il polinomio $g(D) = D^3 + D + 1$, tale polinomio è l'unico polinomio di grado $(n - k) = 3$ del codice. Qualora vi fosse stata la presenza di un'altra parola di codice di grado tre, essa sarebbe potuta essere sommata a $g(D)$ in modo tale da dare vita ad una nuova parola di codice che avrebbe avuto parte dell'informazione pari a zero e una sezione della parola, relativa al controllo di parità, non nulla, cosa impossibile a causa della definizione di codice ciclico. Di conseguenza si può affermare che esiste un unico polinomio di grado $(n - k) = 3$ e ha sempre una forma del tipo: $g(D) = D^3 + \cdots + 1$. Per ricavare le altre righe della matrice generatrice si effettua un'elaborazione prendendo in considerazione il polinomio

$g(D) = D^3 + D + 1$, di fatti: l'ultima riga è rappresentata proprio dal polinomio $g(D)$*;* la penultima riga è rappresentata da una rotazione ciclica verso sinistra del medesimo polinomio (che in termini analitici significa moltiplicare per *D* il polinomio $g(D)$); la terzultima riga invece può essere ricondotta dal polinomio $g(D)$ andando a moltiplicarlo con il polinomio $D^2 + 1$, stando attenti a notare che, a seguito dell'operazione analitica di moltiplicazione polinomiale il risultato porterebbe alla somma binaria di due componenti relative alla presenza di due uno in terza posizione, cosa che porterebbe ad annullare la presenza di un uno in quella posizione come conseguenza delle proprietà relative alla somma modulo-2; infine, la prima riga della matrice, può essere ricondotta dal polinomio $g(D)$ andando a moltiplicare analiticamente questo polinomio con se stesso, questo produrrebbe un risultato pari a $D^6 + 2D^4 + 2D^3 + D^2 + 2D + 1$ che per quanto detto precedentemente riguardo alle regole di algebra binaria, darebbe un contributo pari a $D^6 + D^2 + 1$ che rappresenta appunto la prima riga della matrice. Di conseguenza si avrà:

$$\mathbf{G(D)} = \begin{bmatrix} (D^3 + & D + & 1) & g(D) \\ (D^2 + & 1) & & g(D) \\ & D & & g(D) \\ & & & g(D) \end{bmatrix}$$

Come si può ben notare tutte le righe della matrice generatrice del codice sono multiple di $g(D)$. Pertanto, si può concludere che tutte i polinomi di codice sono multipli del polinomio $g(D)$. Tale polinomio è detto *polinomio generatore* del codice ciclico e permette di definire completamente un codice ciclico ad esso associato. Inoltre, è sempre possibile ricavare da un codice ciclico il suo corrispondente polinomio generatore e questa osservazione è regolata dal fatto che il polinomio generatore di un codice ciclico è divisore del polinomio $(D^n + 1)$, viceversa dunque, ogni divisore del polinomio $(D^n + 1)$, di grado $(n - k)$ genera un codice ciclico $(n - k)$. Infatti, si consideri ad esempio il polinomio $D^k g(D)$ di grado *n* e lo si divida per $(D^n + 1)$, ciò che si ottiene è che $D^k g(D)$ può essere espresso come segue: $D^k g(D) = (D^n + 1) + g^{(k)}(D)$. Dove ovviamente $g^{(k)}(D)$ rappresenta il resto della divisione ovvero un polinomio di grado non superiore a $(n - 1)$. Per le proprietà precedentemente descritte, e ricordando la nomenclatura utilizzata, risulta che $g^{(k)}(D)$ è un polinomio di codice ottenuto al seguito di *k* rotazioni verso sinistra di $g(D)$, dunque risulta essere un polinomio multiplo di $g(D)$ (si scriverà tale polinomio sotto forma di $m(D)g(D)$). Pertanto, si avrà:

$D^k g(D) = (D^n + 1) + m(D)g(D) \rightarrow g(D)(D^k + m(D)) = D^n + 1 = h(D)g(D)$.

Dunque, la prima implicazione è stata dimostrata. Per quanto riguarda la seconda implicazione invece, considerando $g(D)$ come un divisore di $(D^n + 1)$ avente grado $(n - k)$ e considerando i *k* polinomi $g(D)$, $Dg(D)$, … ,$D^{k-1}g(D)$. L'insieme di questi polinomi forma un codice lineare (n, k),

infatti se $x(D)$ è uno di questi polinomi di codice, allora come visto precedentemente si ha: $D^i x(D) = q(D)(D^n + 1) + x^{(i)}(D)$, affinché sia $x(D)$ sia $(D^n + 1)$ siano multipli di $g(D)$, $g(D)$, allora anche $x^{(i)}(D)$ deve esserlo, in quanto può essere espresso come combinazione lineare dei polinomi $g(D)$, $Dg(D)$, … ,$D^{k-1}g(D)$. In conclusione, $x^{(i)}(D)$ è una parola di codice e il codice è ciclico.

Capitolo 2: Codici convoluzionali

Un *codice convoluzionale* è un particolare tipo di codice a blocco che genera uno o più bit di codice ogni volta che all'ingresso viene posto un nuovo bit di informazione. Il codificatore di questa struttura è tipico di alcuni filtri digitali. Un codificatore convoluzionale binario tipico è un sistema avente una memoria finita che genera un numero n_0 di cifre binarie per ogni k_0 cifre informative presenti in ingresso. Lo stato del codificatore viene rappresentato in ogni istante dal contenuto delle sue celle di memoria, inizialmente si suppone nullo (ovvero che tutte le celle di memoria contengono uno zero). Ogni bit di informazione presente nella sorgente informativa fa sì che la memoria venga alterata e vengano fatte scorrere le celle per analizzare lo stato alterato della cella successiva. Il collegamento tra i vari bit di codice è subordinato non solo dai bit in ingresso, ma anche dalla condizione del codificatore negli istanti precedenti. Il numero di celle di memoria di un codice convoluzionale è detto *lunghezza di vincolo*. Questa lunghezza è pari a Nk_0 che rappresenta il numero di posizioni del registro a scorrimento in ingresso. Il principio di funzionamento fa sì che quando un blocco di k_0 cifre è entrato nel registro, gli n_0 sommatori inviano al registro presente in uscita tali cifre in modo tale da poterle trasmettere una alla volta. Successivamente viene inviato dal registro di input un nuovo blocco di k_0 cifre, dopo aver traslato il precedente blocco verso destra. In questo modo il messaggio più vecchio inviato andrà perso, di conseguenza, un codice convoluzionale genera le n_0 cifre codificate tenendo conto sia delle k_0 cifre contenenti il messaggio sia delle $(N-1)k_0$ cifre precedenti. Il diagramma a blocchi che descrive il funzionamento di un codificatore di codici convoluzionali (n_0, k_0), è il seguente:

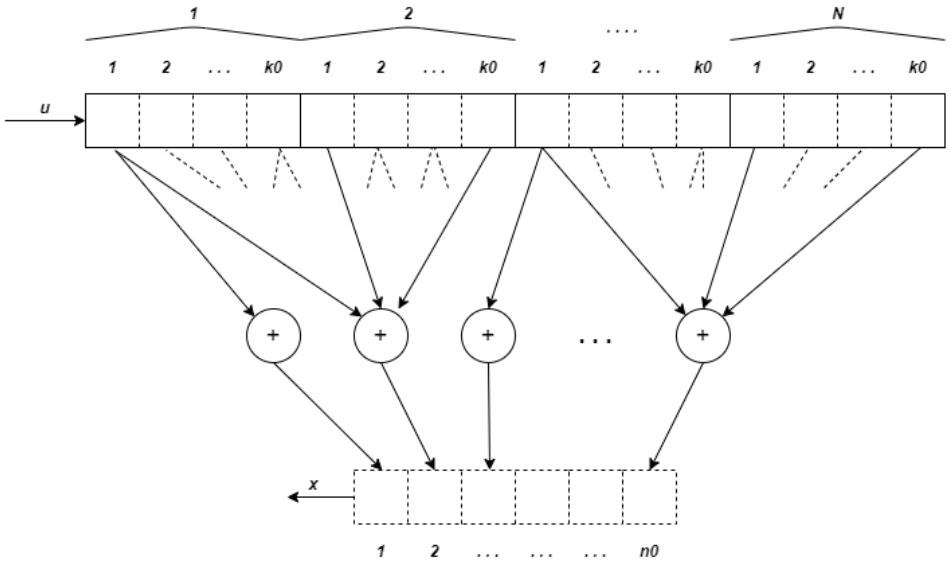


Figura 3: diagramma a blocchi codice convoluzionale

Per capire il funzionamento del codificatore in figura, si pensi di utilizzare *N* sottomatrici **G1, G2, …, Gn** che combinate danno vita ad una matrice avente k_0 righe e n_0 colonne. Ogni matrice i-esima permette di individuare le relazioni tra l'i-esimo elemento appartenente a k_0 spazi di memoria del registro in ingresso con gli n_0 elementi del registro in uscita. Pertanto, tutta la prima riga della matrice i-esima permette di individuare tutte le connessioni tra l'i-esimo spazio di memoria del registro di ingresso con tutti gli n_0 elementi del registro di uscita, la presenza di un uno in una posizione della matrice i-esima significa che vi è connessione tra essa e l'elemento corrispondente nel registro di uscita, uno zero invece indica assenza di connessione. La matrice cosidetta *generatrice di un codice convoluzionale* è dunque la seguente:

$$\boldsymbol{G}_{\infty} \triangleq \begin{bmatrix} \boldsymbol{G}_1 & \boldsymbol{G}_2 & \dots & \boldsymbol{G}_N & & & & \\ & \boldsymbol{G}_1 & \boldsymbol{G}_2 & \dots & \boldsymbol{G}_N & & & \\ & & \boldsymbol{G}_1 & \boldsymbol{G}_2 & \dots & \boldsymbol{G}_N & & \\ & & & \boldsymbol{G}_1 & \boldsymbol{G}_2 & \dots & \boldsymbol{G}_N & \\ & & & & \boldsymbol{G}_1 & \boldsymbol{G}_2 & \dots & \boldsymbol{G}_N \end{bmatrix}$$

Gli elementi non trascritti sono pari a zero. Questa matrice ha una costruzione simile alla matrice *generatrice di un codice a blocco*, infatti ne eredita le proprietà pur estendendosi arbitrariamente essendo una matrice semi-infinita. Di conseguenza si può generalizzare la scrittura di un vettore codificato mediante codifica convoluzionale, partendo da un messaggio semi infinito $\boldsymbol{u}$**,** come segue: $\boldsymbol{x} = \boldsymbol{u} \cdot \boldsymbol{G}_{\infty}$. Per chiarire meglio il funzionamento di un codice convoluzionale si considerino i seguenti due esempi.

Si prenda in esame un codice convoluzionale (3,1) con lunghezza di vincolo pari a 3 ($N = 3$). Il codice è definito dalle tre sottomatrici:

G1*=[1 1 1],* ***G2****=[0 1 1],* ***G3****=[0 0 1].* La matrice generatrice sarà dunque:

$$\boldsymbol{G}_{\infty} = \begin{bmatrix} 111 & 011 & 001 & 000 & \dots & \dots & \dots \\ 000 & 111 & 011 & 001 & 000 & \dots & \dots \\ 000 & 000 & 111 & 011 & 001 & 000 & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{bmatrix}$$

Di conseguenza, qualora si volesse codificare, ad esempio, la sequenza informativa $\boldsymbol{u}$*=(101001…),* essa sarà codificata nella sequenza $\boldsymbol{x}$*=(111011110011001…)* dato che (prodotto matriciale):

$$x_1 = (1 \cdot 1) + \sum 0 = 1, \quad x_2 = (1 \cdot 1) + \sum 0 = 1, \quad x_3 = (1 \cdot 1) + \sum 0 = 1,$$

$$x_4 = (1 \cdot 0) + (0 \cdot 1) + \sum 0 = 0, \; x_5 = (1 \cdot 1) + (0 \cdot 1) + \sum 0 = 1,$$

$$x_6 = (1\cdot 1)+(0\cdot 1)+\sum 0 = 1, \quad x_7 = (1\cdot 0)+(1\cdot 0)+(1\cdot 1)+\sum 0 = 1,$$

$$x_8 = (1\cdot 0)+(0\cdot 1)+(1\cdot 1)+\sum 0 = 1, \quad x_9 = (1\cdot 1)+(0\cdot 1)+(1\cdot 1)+\sum 0 = 0 \text{ ecc.}$$

Questo tipo di codice è anche chiamato *codice convoluzionale sistematico*. Un codice convoluzionale si definisce sistematico quando a seguito del processo di codifica, le prime k_0 cifre delle n_0 cifre codificate sono le medesime k_0 cifre, replicate, del messaggio originale. In termini matriciali la proprietà di sistematicità è dovuta alla presenza delle seguenti sottomatrici generatrici:

$$\boldsymbol{G_i} = \begin{matrix} \\ \\ \\ \end{matrix} \begin{matrix} \overset{k_0}{} & & & & \overset{n_0 \text{ - } k_0}{} \\ \end{matrix} \begin{bmatrix} 0 & 0 & \ldots & 0 & . \\ 0 & 0 & \ldots & . & P_i \\ \ldots & \ldots & \ldots & \ldots & . \\ 0 & 0 & \ldots & 0 & . \end{bmatrix} \Big\} k_0, \quad i = 1,2,3,\ldots,N$$

Lo schema a blocchi del codificatore dell'esempio precedente invece, è il seguente:

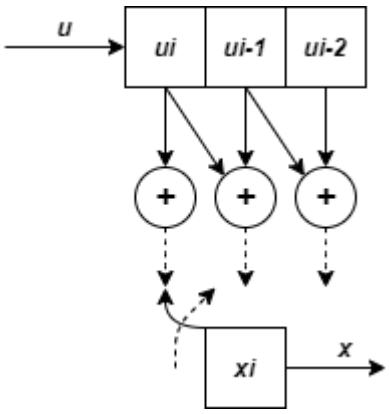


Figura 4: schema a blocchi codice convoluzionale (3,1)

Si consideri ora un codice convoluzionale $(3,2)$ avente due celle di memoria (lunghezza di vincolo *N=2*), generato dalle sottomatrici: $\boldsymbol{G_1} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ e $\boldsymbol{G_2} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$. La matrice generatrice del codice sarà la seguente:

$$\boldsymbol{G_\infty} = \begin{bmatrix} 101 & 001 & 000 & ... & ... \\ 010 & 001 & 000 & ... & ... \\ 000 & 101 & 001 & 000 & ... \\ 000 & 010 & 001 & 000 & ... \\ 000 & 000 & 101 & 001 & 000 \\ 000 & 000 & 010 & 001 & 000 \\ ... & ... & ... & ... & ... \end{bmatrix}$$

In questo caso invece la medesima sequenza informativa di prima, ovvero ***u****=(101001…),* sarà codificata come: ***x****=(101100011001…)* dato che (prodotto matriciale):

$$x_1 = (1\cdot 1) + \sum 0 = 1, \quad x_2 = (1\cdot 0) + (0\cdot 1) + \sum 0 = 0, \quad x_3 = (1\cdot 1) + \sum 0 = 1,$$

$$x_4 = (1\cdot 0) + (0\cdot 0) + (1\cdot 1) + \sum 0 = 1,$$

$$x_5 = (1\cdot 0) + (0\cdot 0) + (1\cdot 0) + (0\cdot 1) + \sum 0 = 0,$$

$$x_6 = (1\cdot 1) + (0\cdot 1) + (1\cdot 1) + \sum 0 = 0,$$

$$x_7 = (1\cdot 0) + (0\cdot 0) + (1\cdot 0) + (0\cdot 0) + (0\cdot 1) + \sum 0 = 0,$$

$$x_8 = (1\cdot 0) + (0\cdot 0) + (1\cdot 0) + (0\cdot 0) + (0\cdot 0) + (1\cdot 1) + \sum 0 = 1,$$

$$x_9 = (1\cdot 0) + (0\cdot 0) + (1\cdot 1) + (0\cdot 1) + (0\cdot 1) + \sum 0 = 1 \text{ ecc.}$$

Anche in questo caso siamo in presenza di un codice *sistematico* e lo schema a blocchi del codificatore è il seguente:

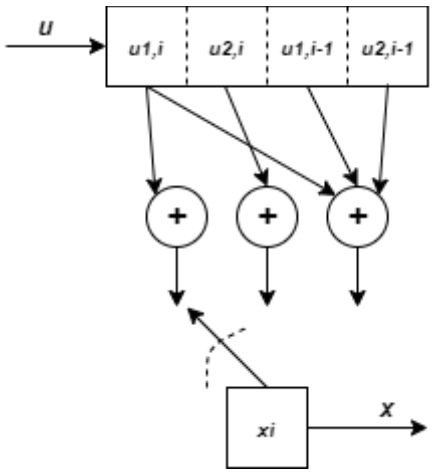


Figura 5: schema a blocchi codice convoluzionale (3,1) simmetrico

Questi due esempi permettono di dimostrare che, innanzitutto, per un codice convoluzionale di tipo $(n_0, 1)$ l'operazione di codifica consiste nel generare le cifre della sequenza di uscita andando proprio ad effettuare la convoluzione discreta delle matrici ***G1, G2, … , GN***, viste come vettori binari, con le cifre $u_1, u_2, \cdots, u_n$ della parola originaria da codificare. Da questa operazione, che ha le medesime proprietà di un prodotto matriciale, ne scaturisce il nome *convoluzionale* per questo tipo di codici. Inoltre, in questi codici, il numero dei sommatori è più piccolo della lunghezza di vincolo del codice, di conseguenza, senza utilizzare *N* sottomatrici ***G***, è più opportuno e immediato andare a rappresentare i collegamenti espressi dal codificatore con dei vettori generatrici $\boldsymbol{g_i}$ (e non più delle matrici), a partire dalle matrici precedenti e considerando il valore di k_0 uguale a uno. Questo metodo permette dunque di semplificare le operazioni di codifica in quanto,

ad esempio, andando a considerare un codice (5,2) con lunghezza di vincolo pari a 7, nel caso in cui esso fosse sviluppato andando ad utilizzare la notazione matriciale, il risultato sarebbe stato quello di andare ad utilizzare sette matrici generatrici di dimensione 5×2. Utilizzando la proprietà precedentemente descritta invece, si possono utilizzare cinque vettori di lunghezza quattordici. Ovviamente questa descrizione permette di utilizzare un minor numero di vettori di lunghezza maggiore, pertanto risulta essere un'alternativa ottima nel processo di descrizione del codice.

2.1 Rappresentazione mediante diagramma degli stati e trellis

Gli esempi analizzati in precedenza dimostrano che lo stato del registro di uscita di un codificatore convoluzionale dipende anche dagli stati precedenti rispetto a quello preso in esame. Di conseguenza ogni nuovo bit da trasmettere, produce una *transizione* dello stato del nostro codificatore. Ad ogni stato del codificatore sono associate due differenti transizioni in base al bit successivo da trasmettere (o zero o uno). L'insieme delle transizioni che subisce un codice rappresenta una descrizione esaustiva per la realizzazione di un codificatore convoluzionale. Il diagramma utilizzato per effettuare questa descrizione è chiamato *diagramma degli stati* del codice convoluzionale, esso ha la forma di un grafo orientato. Il codificatore di un generico codice convoluzionale ha memoria pari a $L = N - 1$ e, definendo con σ_l lo stato del codificatore al tempo *l* che contiene il contenuto della memoria nel medesimo istante *l*, avrà $\sigma_l = (u_{l-1}, \cdots, u_{l-L})$. Gli stati di ingresso e uscita possibili sono invece 2^l.

Considerando il codice convoluzionale (3,1) definito dalle tre sottomatrici ***G1****=[1 1 1],* ***G2****=[0 1 1],* ***G3****=[0 0 1]* e analizzato nell'esempio precedente, i possibili stati sono $2^l = 2^{N-1} = 2^2 = 4$, ovvero: 00, 01, 10 e 11.

2.1 – Rappresentazione mediante diagramma degli stati e trellis

Ipotizziamo che il nostro codificatore sia nello stato 00 (stato precedente alla ricezione della prima cifra della parola da codificare ***u****=(101001…)),* successivamente alla ricezione della prima cifra, ovvero uno, lo stato del codificatore si porterà su 10, la sequenza delle prime tre cifre ricevute sarà 100 e la codifica dell'uno ricevuto darà come output la sequenza 111.
L'algoritmo di codifica segue le seguenti regole operative:

1. Qual ora la cifra in ingresso al codificatore fosse uno, la codifica della cifra sarà data dall'uno in ingresso seguito dall'opposto delle prime due cifre relative alla codifica della precedente cifra binaria, cioè ad esempio: se lo stato del decodificatore fosse stato 00, al seguito della ricezione dell'uno la codifica sarà 111 e il codificatore si sarebbe portato nello stato 10; se lo stato invece fosse stato 10 ciò significa che la potenziale codifica che ha portato il codificatore a posizionarsi in tale stato è o 111 oppure 110, di conseguenza avremo che a seguito della ricezione di un uno la codifica dell'informazione sarà in ambedue i casi pari a 100.
2. Qual ora la cifra in ingresso al codificatore fosse zero, la codifica della cifra sarà data dallo zero in ingresso seguito dalle prime due cifre relative alla codifica della precedente cifra binaria, cioè ad esempio: se lo stato del codificatore fosse stato 00, al seguito della ricezione dello zero la codifica sarà 000 e il codificatore sarebbe rimasto nello stato 00; se lo stato invece fosse stato 10 ciò significa che la potenziale codifica che ha portato il codificatore a posizionarsi in tale stato è o 111 oppure 110, di conseguenza avremo che a seguito della ricezione di uno zero la codifica dell'informazione darà in ambedue i casi pari a 011

Il funzionamento analitico del diagramma degli stati di un codice convoluzionale (3,1), è rappresentato dallo schema sottostante e dal relativo diagramma ad albero.

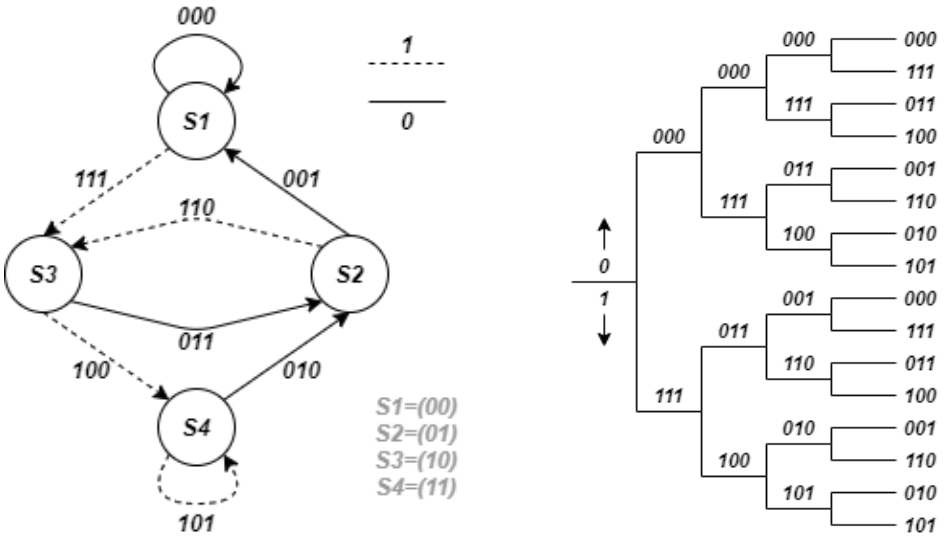


Figura 6: diagramma degli stati e ad albero di un codice convoluzionale (3,1)

Di conseguenza si può verificare che la sequenza in ingresso ***u****=(101001…)* verrà codificata in ***x****=(111 011 110 011 001 111 011 001…)* e il codificatore attraverserà il cammino relativo agli stati *S1→S3→S2→S3→S2→S1→S3.*

Dall'analisi di questo esempio se ne possono scaturire diverse conclusioni utili, innanzitutto si noti che al crescere del parametro *L*, il diagramma degli stati cresce con corrispondenza esponenziale e di conseguenza sarà difficile riuscire a tenere nota del cammino percorso dal codificatore, pertanto, viene aggiunto un nuovo parametro relativo alla valutazione di un codice convoluzionale, ovvero la valutazione relativa alla replica di ogni stato in tutti gli istanti di tempo del processo di codifica. Tale analisi viene effettuata mediante l'utilizzo di un grafico orientato chiamato *traliccio (o trellis in inglese)*. Il traliccio è un grafico in cui in ogni colonna sono presenti gli stati del codificatore, il numero di colonne è pari al numero di cifre della sequenza binaria in ingresso al codificatore e le transizioni tra stato del codificatore (e corrispondente codifica dell'informazione) dall'istante *l* all'istante *l+1*, sono regolate dalle proprietà descritte in precedenza relative alla scrittura del diagramma degli stati. In questo modo gli archi del grafo

tratteggiati rappresentano la transizione degli stati, relativa ad un istante di tempo successivo rispetto a quello in analisi, dovuta alla necessità di codificare un uno in ingresso. Viceversa, gli archi continui all'interno del grafo rappresentano la transizione degli stati, relativa ad un istante di tempo successivo rispetto a quello in analisi, dovuta alla necessità di codificare uno zero in ingresso. Mediante l'utilizzo dei tralicci il cammino percorso da un codificatore risulta essere di più chiara lettura anche nel caso della presenza di un numero di stati elevato.

Il traliccio relativo all'esempio precedente ($\boldsymbol{u} = (101001)$) è mostrato nella figura sottostante:

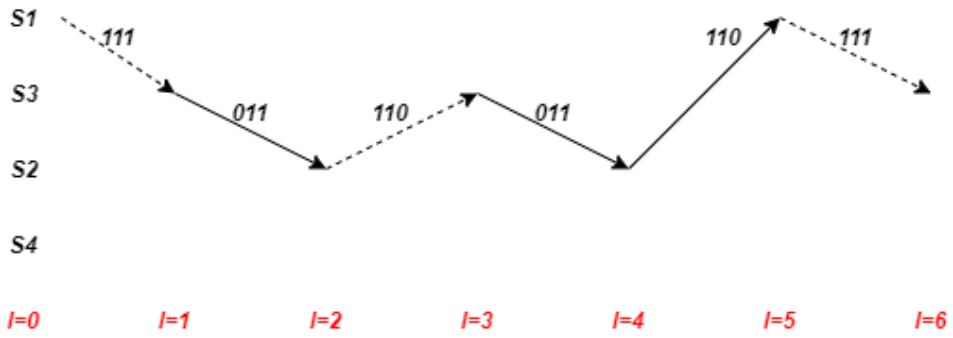


Figura 7: traliccio dell'ingresso u=101001 relativo alla codifica convoluzionale (3,1)

2.2 Proprietà relative alla funzione di trasferimento e alle distanze dei codici convoluzionali

La rappresentazione di un codice convoluzionale mediante un traliccio (*trellis*) è auto esaustiva ed è possibile risalire al codificatore direttamente analizzando esso. Così come nella rappresentazione mediante il diagramma degli stati però, può essere una rappresentazione di difficile lettura qualora il numero di stati dovesse crescere. Anche per i codici convoluzionali, esattamente come visto con i codici a blocco, la capacità di

rilevare e correggere gli errori nel codice è dipendente dalle distanze di Hamming delle sequenze che si vogliono codificare. Al fine di calcolare le prestazioni di un generico codice convoluzionale, si consideri una coppia di sequenze (di cui una è rappresentata da una sequenza di soli zeri) che siano state codificate fino alla profondità *l* del traliccio (*trellis*) e che divergano alla prima diramazione del diagramma ad albero del codice convoluzionale. Si definisce con il termine di *distanza colonna* ($d_c(l)$) dell'*l*-esimo ordine, la distanza minima di Hamming fra tutte le coppie di generiche sequenze binarie. Nel caso preso in analisi, dunque, si considereranno tutte le sequenze codificate fino alla profondità *l* del *trellis*, che divergono alla prima diramazione del diagramma ad albero del codice, rispetto alla sequenza nulla. Pertanto, la distanza colonna, rappresenta una funzione relativa alla profondità di *l* non decrescente. Al fine di trovare il valore di *l* associato alla distanza colonna $d_c(l)$, si definisce con il termine di *distanza minima* d_{min} di un codice convoluzionale, la distanza colonna del codice ottenuta quando *l*=*N* con *N* lunghezza di vincolo del codice, ovvero: $d_{min} = d_c(l)$, mentre si indica con il termine di *distanza libera* d_f del codice convoluzionale la distanza minima di Hamming che vi è tra sequenze di codice infinitamente lunghe, ovvero è il minimo peso che hanno all'interno del processo di codifica le sequenze di codice che dopo essersi staccate dalla sequenza nulla vi riconfluiscono, in formule: $d_f = \lim_{l\to\infty} d_c(l)$. Al fine esemplificativo volto al calcolo di tali distanze, si consideri un codice convoluzionale (3,1) descritto dettagliatamente nel capitolo precedente. Prendendo in esame il *traliccio* (*trellis*) di tale codice e raffigurando solo i cammini relativi alle sequenze che divergono dalla sequenza nulla alla prima diramazione, si ottiene:

2.2 – Proprietà relative alla funzione di trasferimento e alle distanze dei codici convoluzionali

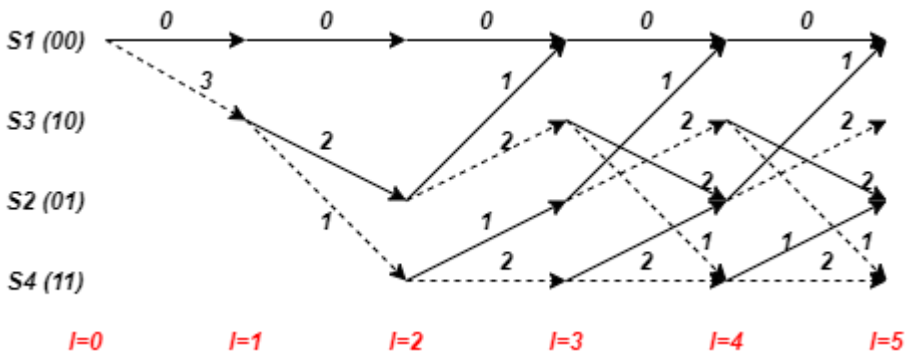


Figura 8: traliccio del codice convoluzionale (3,1) per il calcolo della distanza colonna

Dove si è indicato sopra ogni cammino la distanza della sequenza codificata rispetto alla sequenza nulla. La distanza colonna del codice, in funzione della profondità *l*, è la seguente:

l	$d_c(l)$
1	3
2	4
3	5
4	6
5	6

Dove, dato che la lunghezza di vincolo è pari a *N=3*, la distanza minima del codice sarà pari a 5, come si può ben notare però, alla profondità *l* del *trellis* il cammino si ricongiunge con la sequenza nulla, ma dato che la sequenza congiungente non restituisce un valore relativo alla distanza pari alla distanza colonna relativa alla profondità 3 del *trellis*, al fine di ricercare la distanza libera, è necessario analizzare un valore di *l* superiore. Di fatti per *l=4* si ha una sequenza congiungente la quale ha come distanza pari alla distanza colonna relativa alla profondità 4 del *trellis*, che in questo caso assume il valore di 6. In conclusione, si avrà: d_{min}=5 e d_f=6.

2.2 – Proprietà relative alla funzione di trasferimento e alle distanze dei codici convoluzionali

Il calcolo della distanza libera risulta essere immediato qualora le sequenze codificate non fossero particolarmente lunghe in termini di bit, ma se tali sequenze risultano essere particolarmente lunghe, è possibile effettuare il calcolo di tale distanza basandosi su degli algoritmi risolutivi volti alla gestione del diagramma degli stati. Pertanto, si definisce con il termine di *funzione generatrice* $T(D)$ di un codice convoluzionale una funzione che non fa altro che andare a contare i pesi (numero di uno presenti a seguito della codifica di una cifra in ingresso generati dall'attraversamento di un cammino corrispondente) dei singoli cammini delle eventuali diramazioni di una codifica, tenendo presente solo la sequenza di tutti zeri e le sequenze che divergono da essa alla prima diramazione, che portano lo stato del codificatore da *S1* ad *S1* nell'istante successivo alla elaborazione dell'ultima cifra in ingresso al codificatore, ovvero all'istante *l+1*. Questa funzione è anche chiamata *funzione di trasferimento* di un codice convoluzionale. Per poter capire l'algoritmo di scrittura di tale funzione, si analizzano di seguito alcune proprietà relative alla teoria dei grafi.

Innanzitutto, un grafo è dotato di vertici e lati, ovvero punti e congiunzioni tra i vari punti, si chiamerà con il termine di *cammino* una sequenza di lati che possono essere indicati andando a scrivere la sequenza dei vertici consecutivi inclusi nel cammino. È possibile assegnare a ciascun lato una etichetta che, nel caso della derivazione della funzione di trasferimento di un codice convoluzionale, rappresenterà il peso della codifica dell'informazione al seguito del passaggio da uno stato al successivo. Si introdurrà con il termine di *trasmissione* tra due vertici, la somma delle etichette relative a tutti i possibili cammini che connettono appunto tali vertici. Di conseguenza, in un qualsiasi grafo, è possibile calcolare la trasmissione tra una coppia di vertici intermedi sul grafo andando a ridefinire le varie etichette. Pertanto, dato che per la scrittura di una funzione di trasferimento di un codice convoluzionale, è necessario andare a

identificare tutti i vari cammini, etichettati con i vari pesi di transizione, che congiungono lo stato *S1* con sé stesso, è utile sfruttare tale proprietà per avere una visione più chiara sia del diagramma degli stati che della funzione di trasferimento stessa. Di conseguenza andando a sviluppare passo-passo quanto detto precedentemente si avrà:

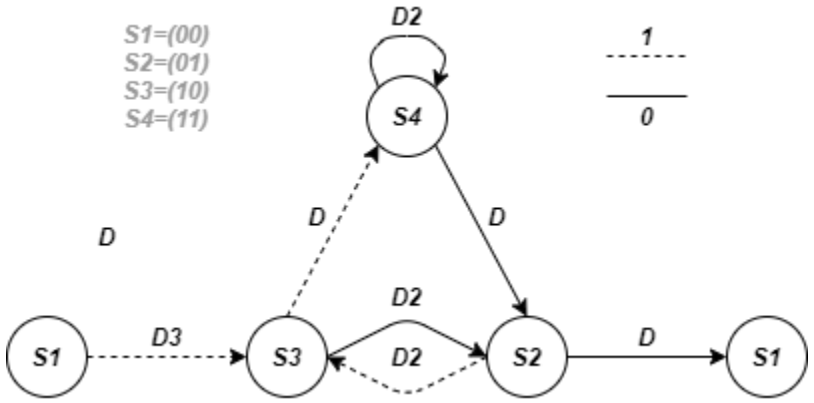


Figura 9: diagramma degli stati di un codice convoluzionale (3,1) raffigurato utilizzando le proprietà dei grafi

Che rappresenta il diagramma degli stati di un codice convoluzionale (3,1) raffigurato in forma differente rispetto all'esempio precedente, in quanto, in questa rappresentazione le etichette dei singoli lati, rappresentanti il peso relativo alla transizione degli stati, permettono di calcolare la funzione generatrice.

Utilizzando ora la proprietà precedente è possibile andare a ridurre il grafo sostituendo il vertice relativo a *S4* come segue:

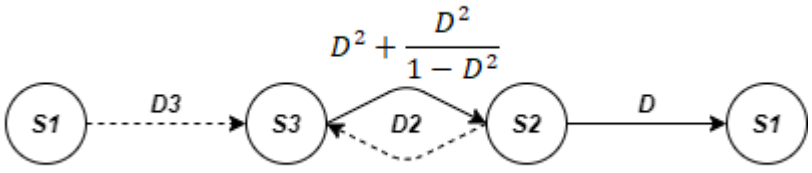


Figura 10: diagramma degli stati di un codice convoluzionale (3,1) raffigurato utilizzando le proprietà dei grafi

Dove l'etichetta dovuta a quel lato scaturisce dalla sostituzione con la funzione $T(S3, S2)$ che regola tutti i possibili cammini che interpongono *S3* da *S2*. Essa sarà pari a:

$$T(S3, S2) = D^2 + D \cdot D \cdot D^2 + D \cdot D \cdot D^4 + \cdots = D^2 + \frac{D^2}{1 - D^2}$$

Continuando a utilizzare la proprietà precedente, avremo che il grafo si ridurrà ulteriormente andando a sostituire il vertice relativo allo stato *S3* come segue:

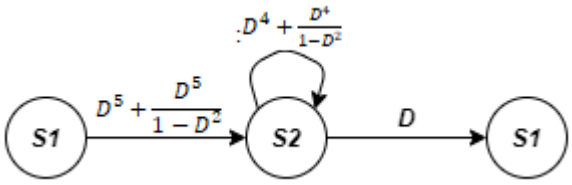


Figura 11: diagramma degli stati di un codice convoluzionale (3,1) raffigurato utilizzando le proprietà dei grafi

Dove l'etichetta dovuta al lato che congiunge *S1* da *S2* scaturisce dalla funzione $T(S1, S2)$ che regola tutti i possibili cammini che interpongono *S1* da *S2*, e medesimo discorso vale per l'etichetta sul lato che congiunge *S2* con sé stesso, in questo caso la funzione presa in considerazione è stata $T(S2, S3)$. L'ultimo step sarà quello di eliminare anche il vertice relativo allo stato *S2*, in modo tale da poter ridurre il grafo in un grafo avente due vertici (*S1* e *S1*) e un unico lato con etichetta pari ai pesi di tutti i possibili cammini che interpongono i due stati. Il grafo risultante sarà il seguente, e la funzione che ne scaturisce come etichetta assegnata al singolo lato presente rappresenta la funzione di trasferimento di un codice convoluzionale *(3,1).*

$$T(D) = \frac{D^{14} - 4D^{12} + 4D^{10}}{1 + D^4 - 2D^2}$$

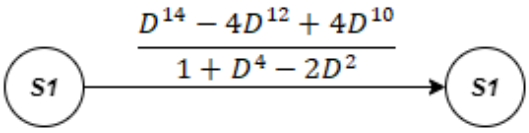


Figura 12: diagramma degli stati di un codice convoluzionale (3,1) raffigurato utilizzando le proprietà dei grafi

2.3 Codici convoluzionali catastrofici

Attraverso l'utilizzo di una codifica convoluzionale si può facilmente notare che, grazie anche agli esempi precedenti, ad una sequenza in ingresso di bit di informazione nulla, il codificatore associa a sua volta una codifica nulla. Questo perché, mediante la rappresentazione degli stati in funzione del tempo di esecuzione, si verifica che conseguenza della proprietà di linearità del codice, è il percorso semi infinito del lato congiungente lo stato *S1* con sé stesso. Oltre a questo, si osservi come la codifica associata ad una sequenza in ingresso pari a $\boldsymbol{u_1}$*=1000…*, per un codice convoluzionale *(3,1),* avente il seguente diagramma degli stati/diagramma ad albero:

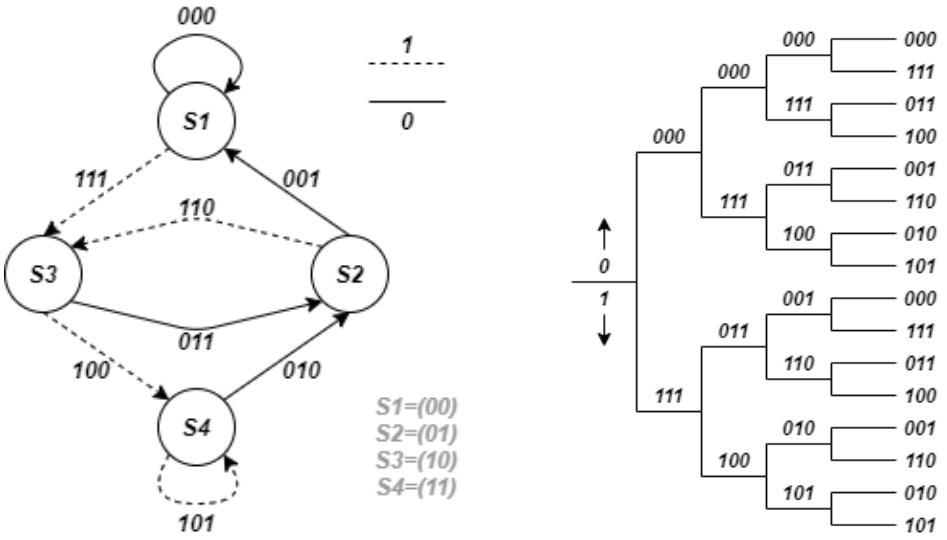


Figura 13: diagramma degli stati e ad albero di un codice convoluzionale (3,1)

Tale ingresso, fa riferimento ad un percorso nel diagramma degli stati pari a: *S1→S3→S2→S1* che darà origine ad una sequenza codificata

$\boldsymbol{x_1}$*=111 011 001 000 …,* di conseguenza notiamo che tale sequenza semi infinita ha peso pari a sei, che rappresenta la *distanza minima di Hamming* tra sequenze codificate infinitamente lunghe, presa nell'insieme di tutti i possibili cammini nel *traliccio (trellis)* del codice che, dopo essersi staccati dalla sequenza nulla, vi riconfluiscono. Tale distanza è chiamata appunto con l'appellativo di "minima", in quanto si può dimostrare come al crescere della distanza minima di Hamming tra le sequenze di ingresso semi infinite, cresca anche quella delle corrispondenti sequenze codificate. Di fatto questo non accade in quanto, ad esempio, prendendo la sequenza $\boldsymbol{u_2}$*=1100…,* che ha distanza minima di Hamming pari a due, diversamente dalla sequenza $\boldsymbol{u_1}$*=1000…* considerata precedentemente che ha distanza minima di Hamming pari a uno, la codifica mediante il medesimo codificatore convoluzionale *(3,1)* porta ad avere: $\boldsymbol{x_2}$*=111 100 010 001 000 …,* che similmente con $\boldsymbol{x_1}$*=111 011 001 000 …,* ha distanza minima pari a sei.

Questo tipo di codici vengono chiamati *codici catastrofici*, proprio perché la distanza minima di Hamming tra parole di codice codificate non cresce al crescere di tale distanza nelle sequenze di ingresso al codificatore. Questo tipo di denominazione del codice risiede nel fatto che il codificatore associa ad una eventuale parola di peso la parola di uscita nulla, pertanto, ciclando su tale ramo il peso della sequenza di uscita non varia. Tale tipo di lati all'interno dei grafi corrispondenti ai diagrammi degli stati dei codici convoluzionali vengono chiamati *selfloop.*

2.4 Implementazione software codice convoluzionale

L'implementazione su Matlab di un codificatore convoluzionale è piuttosto semplice. Si inizi con l'analizzare la funzione *conv_code,* si può notare come essa sia una funzione che prende in ingresso due parametri, *K* che rappresenta anche il numero di cifre differenti rispetto alle quali corrispondono codifiche differenti, e *gen_oct* che rappresenta invece un vettore di *N* sottomatrici generatrici del codice convoluzionale passate in forma ottale. A titolo esemplificativo, nel caso in cui, si volessero passare alla funzione i valori di *K=3* e *gen_oct=[6 4 2]* si avrà che il numero di cifre differenti rispetto alle quali corrispondono codifiche differenti sarà pari a *3,* che *N* sarà uguale a *3* e che le matrici generatrici saranno del tipo *G1=[110], G2=[100]* e *G3=[010]*. Successivamente, viene gestita l'eventualità in cui lo script venisse lanciato senza indicarne gli argomenti, in questo caso viene impostato di default l'approccio di codifica che vede assegnare *K=1* e il vettore relativo alle matrici generatrici pari a: *gen_oct = [7 3 1];*. Andando avanti, si può notare come viene utilizzato una ulteriore variabile *gen_bin* che si occupa di andare a codificare le matrici generatrici del codice passate in input alla funzione, dalla forma ottale alla forma

binaria (passando attraverso la codifica in decimale qualora uno dei numeri passati come input sarà superiore al *7*), attraverso la riga di codice:

gen_bin = dec2bin(oct2dec(gen_oct))-'0'; (il -'0' permette di far passare i numeri presenti in gen_oct sotto forma di numeri e non sottoforma di sctringhe). Dopo aver effettuato la conversione delle matrici generatrici in ingresso, viene implementato il calcolo del valore di *N* ovvero del numero delle matrici generatrici del nostro codice convoluzionale, e del valore della lunghezza di vincolo del codice attraverso il comando *V=ceil(temp/K);* che assegna alla variabile V il valore intero superiore relativo alla divisione della variabile *temp* (numero di righe della matrice G_∞ costruita a partire dalle matrici generatrici passate come input alla funzione). Proseguendo, viene verificata la condizione per la quale si potrebbe avere che le dimensioni delle matrici generatrici del codice siano successivamente incompatibili con le dimensioni relative alla parola correttamente codificata e con le dimensioni dei vari stati del nostro traliccio. Per risolvere eventuali problemi di incompatibilità viene utilizzata l'assegnazione *gen_bin = [zeros(N,V*K-temp),gen_bin];* che attraverso la funzione *zeros(N,V*K-temp)* permette di aggiungere una matrice di *N* righe e *V*K-temp* colonne a partire dalla sinistra della matrice *gen_bin.* Successivamente, viene dichiarato il numero di stati del codice e la matrice corrispondente alle varie transizioni di stato che, inizialmente, viene impostata come una matrice di dimensione *S*2^K* di tutti zeri; medesimo trattamento è riservato alla matrice delle uscite corrispondenti alle varie transizioni tra stati. Vengono poi inizializzati due cicli *for* concatenati per analizzare le risposte del nostro codice all'arrivo in ingresso di tutte le possibili *2^K* sequenze informative relative allo spostamento nei vari stati. Pertanto, per passare dalla valutazione che un generico codice effettua a seguito di eventuali parole binarie in ingresso ad uno stato, rispetto ad un altro, devono essere esplorate tutte le possibili *2^K* combinazioni, decifrate posizionalmente su almeno *K*

bit, di sequenze di ingresso ammissibili ad uno stato. Successivamente nella variabile *temp* vengono concatenati orizzontalmente il vettore corrispondente ai bit dello stato corrente, con quello relativo alla traduzione posizionale della parola binaria in ingresso. Questo vettore *temp* viene poi manipolato per avere un nuovo vettore *s_new_bin* generato dalle prime *K* cifre binarie del vettore *temp* prese a partire da sinistra. Successivamente si effettua la conversione dello stato da stringa di bit a posizione all'interno del grafo, e viene ricalcolato lo stato successivo di transizione al seguito della valutazione delle cifre binarie *u* in ingresso allo stato attuale. Infine, si procede con la valutazione dell'uscita come fatto precedentemente, ovvero, andando prima ad assegnare ad un array *y_bin* il resto della divisione per due del prodotto matriciale trasposto di *temp* con *gen_bin* e successivamente procedendo con la traduzione dell'uscita da stringa di bit a parola binaria con conseguente riassegnazione dello stato di uscita relativo al comportamento del codificatore a seguito di una eventuale cifra di ingresso *u* all'interno di uno stato *s*. Una volta finiti i cicli *for*, vengono impostati i blocchi binari da codificare, ricordando che la lunghezza *L_code* dei blocchi deve essere divisibile per il parametro *K*. Successivamente vengono stampati e visualizzati a video i parametri della codifica convoluzionale come il *code rate nominale* (nel caso *K=1* e *N=3*, *Rc=1/3*) e *code rate effettivo*. Di seguito viene impostata la risposta del codificatore, sotto forma matriciale, relativa allo stato zero di *X* messaggi (Parametro utilizzato precedentemente per la valutazione dei blocchi di un codice). Di seguito, si procede nello impostare un ciclo *for* volto a codificare tutte le parole di lunghezza prefissata *L_code*, che può essere impostata manualmente dal progettista andando a modificare il codice sorgente (nel nostro caso abbiamo scelto di assegnare a tale variabile il valore sei in modo tale da poter eleborare un output conciso e non troppo prolisso riguardante la codifica di piccole parole binarie di esempio), generate sotto forma di matrice identica di dimensione *L_code* x *L_code* (comando *c=eye(L_code)*).

Una volta inizializzato questo ciclo *for* si imposta come stato iniziale del codificatore lo stato numero uno e si procede nell'effettuare un nuovo ciclo *for* volto alla valutazione delle singole cifre binarie presenti all'interno delle parole contenute nella matrice *x.* Per far questo viene effettuata una valutazione dell'ingresso attuale, ovvero, ad *u_bin* viene assegnata come parola binaria di ingresso la parola della matrice *x* corrispondente alla concatenazione delle cifre di riga *x_ind* (che varia da uno a sei nel caso preso in esame con *L_code*=6) e colonne *i, i+1, ... i+K-1.* Successivamente, viene assegnato ad *u* l'indice della prossima cifra da valutare in ingresso, e viene impostata la prima parola di risposta al seguito della valutazione della prima cifra *u_bin,* rappresentato dal valore in binario del contenuto della matrice *Y,* contenente le coppie in ottale di valori ingresso/uscita de visualizzare, alla posizione *(s,u)* decrementato di uno. Tale valore, tradotto su *N* bit come da specifiche di codifica, sarà poi posizionato all'interno della matrice di output alla posizione corrispondente del successivo blocco binario seguente ai blocchi di cifre corrispondenti alle cifre decifrate fino a quel momento. Con il comando *s = S_tran(s,u);* si passa invece allo stato successivo del diagramma degli stati. Infine, viene valutato con un ulteriore ciclo *for* il comportamento del codificatore al seguito dell'ingresso uno come ultima cifra della parola binaria in ingresso, in modo tale da tutelare la normale rappresentazione del codice andando a completare il processo di transizione ad un'uscita di blocchi codificati di dimensione *N* nulla.
Finalmente si può procedere alla rappresentazione del codice andando a visualizzare a video le codifiche assegnate ai vari codici non nulli e con un unico uno in diverse posizioni rispetto alle posizioni di un generico codice di lunghezza *L_code,* in modo tale da poter visualizzare in maniera opportuna e limpida il processo di codifica. All'interno dell'***Appendice A*** si può visionare il codice utilizzato per la realizzazione opportunamente commentato.

Capitolo 3: Codici di Reed-Solomon

I codici di *Reed-Solomon* fanno parte della classe di *codici di Bose-Chaudhuri-Hocquenghem (BCH)*. Questa classe di codici è utilizzata per la correzione di errori casuali, e sfruttano algoritmi di decodifica non troppo complessi. Per ogni coppia di numeri interi, *m* e *t*, esiste un codice binario BCH generato da polinomi che seguono le seguenti equazioni:

$$n = 2^m - 1, n - k \leq m \cdot t, d_{min} \leq 2t + 1.$$

Questi codici sono codici con capacità correttiva *t*, ovvero sono in grado di correggere al più *t* errori di trasmissione. Tali codici sono utili in quanto consentono una notevole flessibilità nella scelta dei parametri quali lunghezza del blocco e rapporto di codifica. Questa flessibilità è data dalla loro proprietà di poter lavorare con blocchi di poche centinaia di simboli, di fatto essi sono tra i meglio utilizzati quando si vuole cercare un approccio di codifica mantenendo la codifica di una sorgente avente pressoché la medesima lunghezza e il rapporto. Un elenco di polinomi generatori BCH è dato dalla seguente tabella, in cui sono elencati anche i valori noti di *n,k* e *t:*

n	k	t	$g(D)$
7	4	1	13
15	11	1	23
	7	2	721
	5	3	2467
31	26	1	45
	21	2	3551
	16	3	107657
	11	5	5423325
	6	7	313365047
63	57	1	103
	51	2	12471
	45	3	1701317
	39	4	166623567
	36	5	1033500423
	30	6	157464165547
	24	7	17323260404441
	18	10	1363026512351725
	16	11	6331141367235453
	10	13	472622305527250155
	7	15	5231045543503271737
127	120	1	211
	113	2	41567
	106	3	11554743
	99	4	3447023271
	92	5	624730022327
	85	6	130704476322273
	78	7	26230002166130115
	71	9	6255010713253127753
	64	10	1206534025570773100045
	57	11	335265252505705053517721
	50	13	54446512523314012421501421

Figura 14: elenco dei polinomi generatori di codici BCH in formato ottale

I polinomi sono rappresentati in forma ottale, con il termine di grado più elevato a sinistra, di conseguenza, la prima riga della tabella va interpretata come: $13 \rightarrow 00110001 \rightarrow g(D) = D^5 + D^4 + 1$.

I codici ciclici di Reed-Solomon differentemente da questi ultimi invece, sono codici generalizzati all'utilizzo di codifiche di simboli di dimensione pari a 2^m. Di conseguenza ogni simbolo viene considerato come una m-upla binaria e pertanto può essere trattato in maniera analoga alla codifica BCH, considerandolo come una particolare estensione di un codice binario. I parametri realizzativi per un codice di Reed-Solomon sono i seguenti:

- Simbolo: m cifre binarie
- Lunghezza del blocco: $n = (2^m - 1)\ simboli = m(2^m 1)\ bit$
- Controlli di parità: $(n - k) = 2t\ simboli = 2m \cdot t$ $cifre\ binarie$

Così come i BCH, i codici di Reed-Solomon sono in grado di correggere al più *t* errori. Questi codici vengono principalmente utilizzati per la codifica concatenata o per la correzione di errori *"burst"*. I cosiddetti errori di tipo "burst" sono degli errori con configurazione posizionale consecutiva. Cioè sono degli errori di lunghezza *b* che possono essere rappresentati nel seguente modo: $e(D) = D^i e_b(D)$ dove D^i rappresenta la posizione del burst nella sequenza di errore mentre $e_b(D)$ è proprio il polinomio di errore. La particolarità di un codice di Reed-Solomon è che esso rappresenta un codice particolarmente efficiente per la risoluzione di questo tipo di errori. Questo perché, una misura di efficienza relativa alla capacità di un codice ciclico di risolvere questo tipo di errori, è regolata dal rapporto: $z = \frac{2b}{n-k}$, in quanto tanto più questo rapporto è prossimo a uno tanto più il codice è efficiente nella correzione. L'utilizzo di questo parametro di decisione è legato al fatto che un codice ciclico (n, k) può rivelare tutti i burst la cui lunghezza non sia superiore a $(n - k)$, inoltre tale codice, volto alla correzione di errori di tipo burst, può correggere eventuali errori burst di lunghezza *b* o inferiore purchè sia soddisfatta la seguente disuguaglianza (*limite di Reiger*): $n - k \geq 2b$.

3.1 I campi di Galois

Al fine di comprendere i principi da attuare al fine di effettuare operazioni di codifica basati sull'utilizzo dei codici di Reed-Solomon è opportuno approfondire la teoria dei campi finiti, detti anche di *Galois*, sui quali si basano i codici ciclici. In particolare, per ogni numero primo *q*, esiste un campo di Galois *GF(q)* formato da *q* elementi, che può essere esteso ad essere formato da q^m con *m* numero intero maggiore di uno. Tipicamente, quando si effettua un'operazione di codifica attraverso un codice di Reed-Solomon, si fa riferimento all'utilizzo di campi di *Galois*

formati da 2^m elementi, i quali sono un'estensione del campo $GF(2)$. In particolare, tale campo è un insieme finito di elementi rispetto al quale sono definite le operazioni di somma e prodotto con proprietà assimilabili a quelle verificate da somma e prodotto sui numeri razionali o reali o anche complessi. Oltre a questo, gode delle seguenti proprietà:

1. È chiuso rispetto alle operazioni di somma e prodotto
2. Prevede le proprietà di associatività, commutatività e distributività
3. Prevede l'esistenza dell'elemento neutro rispetto alla somma (ovvero zero) e rispetto al prodotto (ovvero uno)
4. Prevedono l'esistenza della definizione del concetto di differenza e divisione, dunque, per ogni elemento *u*, esiste il corrispettivo elemento *-u*, e per ogni elemento $u \neq 0$, dell'elemento u^{-1}.

Oltre a tali elementi volti a definire le proprietà stesse del campo $GF(2^m)$, esiste un ulteriore elemento primitivo α non nullo volto a rappresentare qualsiasi elemento del campo come potenza di tale elemento α. Di conseguenza, a partire dalla terna formata dagli elementi *0, 1,* α, è possibile creare un insieme di elementi infinito. Al fine, dunque, di avere uno spazio di Galois finito, di cardinalità 2^m e che abbia tutte e quattro le proprietà precedenti, è necessario stabilire una condizione relativa all'elemento primitivo α del tipo: $\alpha^{(2^m-1)} = 1 = \alpha^0$. Pertanto, qualora esistesse un elemento che risulta essere una potenza maggiore dell'elemento primitivo rispetto a $2^m - 1$ si riduce ad un elemento rappresentato da una potenza minore, ad esempio: $\alpha^{(2^m+4)} = \alpha^{(2^m-1)} \cdot \alpha^5 = 1 \cdot \alpha^5 = \alpha^5$. Si definisce dunque una sesta proprietà dei campi di Galois, relativa alla presenza di tale elemento primitivo, la quale, considerando la presenza di un ordine *e*, definito come il più piccolo esponente per il quale un elemento β, elevato ad *e* faccia uno, allora, si ha che l'elemento primitivo α ha ordine $2^m - 1$ ovvero il maggiore possibile. Oltre a questo, l'elemento primitivo α è una

delle radici del polinomio primitivo $p(x)$ che rappresenta la funzione che descrive il campo di Galois. Il rapporto tra campi di Galois e codici di Reed-Solomon risulta basarsi sul fatto che si può porre in corrispondenza biunivoca i q simboli formati da *m* bit dell'alfabeto del codice, con gli elementi di un campo $GF(2^m)$. Pertanto, da questo momento in poi, si riferirà in maniera equivalente gli elementi di campo con i simboli di codice. Gli elementi presenti all'interno dei campi di Galois possono essere rappresentati in due modalità:

- La prima è una rappresentazione in termini dell'esponente di α, chiamata *index form*, similare alla rappresentazione in decibel dell'elemento stesso, del tipo: $\beta_{if} = log_\alpha \beta$, dove si considera $0 = \alpha^{-1}$
- La seconda è una rappresentazione che utilizza m-ple binarie, chiamata *polynomial form*, avente la seguente rappresentazione: (00..0)=0, (00..1)=1, …, (100..0)= α^{m-1}. Per gli elementi successivi si utilizza la rappresentazione considerando m-ple relative alla divisione in modulo per il polinomio primitivo di tale m-pla esprimendo il resto in funzione degli elementi che vanno da α^0 ad α^{m-1}.

Una volta analizzato questo, è importante precisare come le operazioni aritmetiche su $GF(2^m)$, sono leggermente differenti, in quanto, ad esempio, l'addizione e la sottrazione vengono effettuate andando a manipolare le m-ple attraverso l'utilizzo dell'operatore logico *xor*. Qualora si volesse realizzare un prodotto invece, la forma più conveniente da utilizzare è la *index form* in quanto consente di ricavare il risultato come somma di due costrutti logaritmi. Queste procedure di utilizzo degli elementi di un campo di Galois permettono di ricavare delle tavole che diano una descrizione esaustiva del risultato della somma e del prodotto di ciascun elemento per

ogni elemento del campo, il che è importante in quanto gli algoritmi di codifica e decodifica di un codice di Reed-Solomon si basano su operazioni algebriche compiute su parole di codice che vengono trattate come elementi di un campo di Galois.

3.2 Tecnica di codifica

Come precedentemente accennato, la forma più utilizzata di codici di Reed-Solomon risulta essere quella avente parametri $RS(2^{m-1}, 2^m - 1 - 2t)$, dove $2t$ rappresenta la capacità di correggere il codice in relazione all'utilizzo ridondante di $2t$ simboli. Il polinomio generatore di tale codice è definito come:

$$g(x) = (x - \alpha^{j0}) \cdot (x - \alpha^{j0+1}) \cdot (x - \alpha^{j0+2}) \cdot \cdots \cdot (x - \alpha^{j0+2t-1})$$
$$= g_0 + g_1 x + g_2 x^2 + \cdots + g_{2t} x^{2t}$$

$$\text{con } j \in [0,1]$$

Il grado di tale polinomio è uguale al grado dei polinomi generatori di un generico codice *BCH*, introdotto in questo capitolo, del quale il Reed-Solomon ne rappresenta un sottoinsieme. Si indica con il termine di *dataword* la sequenza formata da $(d_1, d_2, \cdots, d_k)$, in cui i singoli d_i rappresentano una parola binaria composta da *m* bit che rappresenta una parte del messaggio da codificare che può essere interpretato come un termine del campo $GF(2^m)$ e quindi come un coefficiente di un polinomio avente la seguente forma: $d(x) = d_0 + d_1 x + d_2 x^2 + \cdots + d_{k-1} x^{k-1}$. L'algoritmo di codifica permette di ottenere una *codeword*, ovvero un polinomio di grado $n - 1$ che è divisibile per $g(x)$. Tale codeword potrebbe essere ottenuto in maniera semplice andando a moltiplicare il dataword con il polinomio generatore in modo tale da ottenere: $c(x) = d(x) \cdot g(x)$, tale

rappresentazione porta però a dover gestire un codice non sistematico, pertanto al fine di ottenere una codifica sistematica si considera come codeword:

$c(x) = x^{2t}d(x) = x^{2t}d(x) + p(x)\ con\ p(x) = x^{2t}d(x) mod[g(x)]$. Di conseguenza, il polinomio $c(x)$ ha come coefficienti i simboli di parità aventi grado che va da *0* a *2t* e come coefficienti relativi ai simboli di grado successivo, i simboli informativi. In questo modo si è in grado di ottenere un codeword che sia divisibile esattamente per il polinomio generatore e dunque un eventuale decoder potrà valutare la validità di esso andando ad eseguire la divisione per il polinomio generatore $g(x)$, un eventuale resto indicherà la presenza di rumore nel canale.

3.2.1 Implementazione software codice Reed-Solomon

L'implementazione su MATLAB di un codificatore di Reed-Solomon prevede, innanzitutto, la definizione dei parametri del campo di Galois rispetto al quale tale codice fa riferimento, e l'inizializzazione del campo stesso. Esso viene inizializzato attraverso l'utilizzo della funzione *GF_init* la quale prende come input i valori *p* (numero primo) ed *m* in modo tale da poter generare tale campo a partire dalla potenza p^m. In particolare, essi possono essere passati manualmente al programma altrimenti verranno interpretati di default dall'algoritmo con i parametri *p=2* ed *m=5*. Si ricorda che il parametro *p* deve essere un numero primo, altrimenti la costruzione della tabella di divisione fallisce e GF(*p*) non è un campo. Il parametro *prim_poly*, invece, definisce (nella notazione numerica convenzionale di *num2poly* e/o *poly2num*) il polinomio primitivo da utilizzare per la costruzione del campo *GF*(p^m) (al fine di ottenere una moltiplicazione invertibile), dunque, se non viene passato un valore a tale funzione viene

utilizzato il primo (sempre rispetto a *num2poly*) polinomio primitivo di grado pari a *m*. All'interno della funzione *GF_init* si procede innanzitutto con la definizione del numero di elementi del campo, attraverso l'istruzione *q = p^m;*, e successivamente si effettua il calcolo del polinomio primitivo associato al campo qualora esso non fosse passato, come detto prima, come parametro alla funzione. Tale ricerca viene effettuata dalla funzione *first_prim_poly(GF,n,trace)*, la quale va a ricercare il polinomio Monico primitivo relativo al grado e al traliccio assegnato al campo, in particolare si ricorda che un polinomio $p(x)$ è primitivo quando $x^k mod\ p(x)$, al variare di *k*, genera tutti e soli gli elementi non nulli del campo (ovvero ha periodo massimo paria a p^{n-1}. Tale ricerca è ulteriormente affidata ricorsivamente all'utilizzo della funzione *next_prim_poly* la quale effettua la ricerca del polinomio primitivo all'interno dell'insieme dei polinomi irriducibili. Successivamente vengono definite le operazioni aritmetiche sui polinomi a coefficienti in *GF*(*p*). In questa particolare implementazione, i polinomi a coefficienti in *GF*(*p*) sono rappresentati con i vettori dei coefficienti in ordine di grado, ad esempio, il vettore '*a*' rappresenta il polinomio:

$a_1 + a_2 x + a_3 x^2 + \cdots + a_f x^{f-1}$. Nelle funzioni si assume che il coefficiente di grado più alto, a_f, sia diverso da zero e che il polinomio nullo è rappresentato dal vettore vuoto *[]*. Subito dopo vengono implementate varie funzioni volte ad effettuare la ricerca dei possibili polinomi Monici irriducibili (divisibili solo per sé stessi e per 1) fino ad un grado massimo assegnato. In particolare, l'algoritmo di ricerca è basato su quello di *crivello di Eratostene* il quale utilizza una tabella di variabili booleane che vengono azzerate se il polinomio ad esse corrispondente è un multiplo di un polinomio irriducibile. Tale algoritmo porta ad assegnare una valutazione seguente ai primi polinomi:

1. '0': non irriducibile in quanto divisibile con resto nullo da qualsiasi polinomio.
2. '1': irriducibile per definizione.
3. '$x+a$': irriducibile
4. 'x^2+ax+b': riducibile solo nel caso in cui esistono i coefficienti *c* e *d* in *GF*(*p*) con $c+d=a$ e $c\cdot d=b$.
5. 'x^3+ax^2+bx+c': riducibile se esistono *d*, *e* ed *f* con $d+f=a$, $e+(d\cdot f)=a$ ed $e\cdot f=c$.

Una volta definite le variabili e le operazioni aritmetiche tra polinomi all'interno del campo *GF*(*p*), viene effettuata l'inizializzazione delle variabili che descrivono il campo finito *GF*(*q*). Si ricorda che il valore *q* rappresenta il numero di elementi del campo *GF(*p^m*)* a cui appartengono i vari coefficienti. La notazione scelta al fine di rappresentare gli elementi di *GF(*p^m*)* è quella esponenziale/logaritmica, in cui tutti gli elementi non nulli del campo sono rappresentati come potenze di un elemento primitivo (il polinomio *X*). L'elemento nullo del campo (neutro rispetto alla somma) è rappresentato da $\alpha^{-\infty}$ e convenzionalmente dall'esponente -1, mentre gli altri elementi del campo hanno esponenti compresi tra *0* (elemento neutro rispetto alla somma) e $q-2$ (sono *q* in totale). In particolare nella fase di inizializzazione viene costruita la tabella dei logaritmi di *Zech*, per la quale si ha che $\alpha^{z_n}=1+\alpha^n$, dove z_n è memorizzato in $z\cdot(2+n)$ con *n*=*-1,0,1,…,* e $q-2$. Tali logaritmi di Zech sono utili al fine di accellerare la somma, in quanto: $\alpha^n+\alpha^m=\alpha^n(1+\alpha^{m-n})=\alpha^{n+z(m-n)}$. Per costruire tale tabella, si costruiscono le due tabelle dei valori numerici convenzionali delle potenze di α^n (potenze successive del polinomio *X*) e di $1+\alpha^n$ (le stesse sommate al polinomio 1). Infine, a seguito della funzione *poly2xpoly* la quale permette di convertire un polinomio a coefficienti in *GF*(*p*) in un polinomio a coefficienti in *GF*(*q*), viene definito il polinomio generatore di

un codice *BCH* di capacità correttiva *t*, avente *t0* come esponente della prima delle 2^t potenze successive di α in cui il polinomio deve annullarsi, e il polinomio generatore del codice *Reed-Solomon*, il quale è equivalente a quello del codice *BCH* ma ha coefficienti nel campo esteso, di fatto, nel campo esteso il polinomio minimale di α^i è semplicemente $x - \alpha^i$.
Finalmente dunque è possibile andare a visualizzare: il polinomio primitivo del campo *GF*(*p*) in formato polinomiale e decimale, il polinomio generatore dei codici *BCH* e *Reed-Solomon* associato ai parametri dei vari campi, il *dataword* in formato decimale, binario e polinomiale di un codice relativo alla codifica di un testo passato in input al programma in fase di esecuzione, il *codeword* associato al dataword generato, relativo all'utilizzo di una codifica di *Reed-Solomon* in formato polinomiale e decimale per ogni carattere ASCII del testo passato in input al programma e la rappresentazione mediante l'utilizzo della trasformata discreta di Fourier (*DFT*) dei polinomi generatori associati al codice *BCH* e al codice di *Reed-Solomon*. All'interno dell'***Appendice A*** si può visionare l'output associato al codice in linguaggio MATLAB utilizzato per la realizzazione.

Capitolo 4: Codici concatenati

I codici concatenati, sono dei particolari codici molto lunghi che permettono di avere, a seguito dell'applicazione di tecniche di codifica particolari, capacità di correzione di errori molto elevate. Lo schema a blocchi relativo alla realizzazione di un codice concatenato è il seguente:

Figura 15: schema a blocchi di una codifica/decodifica concatenata

Il *codice interno* è solitamente è rappresentato da un codice binario (n, k), quale ad esempio un *codice di Hamming (7,4),* capace di correggere un solo errore alla volta. Il *codice esterno* invece è rappresentato da un codice non binario (N, K), quale ad esempio un *codice di Reed-Solomon (16,11),* capace di correggere due errori e operante su byte di quattro bit ciascuno. La codifica mediante un codice concatenato si esegue andando a prendere un blocco di $K \cdot k$ cifre informative (nel caso di esempio saranno $4 \cdot 11 = 44$) che saranno divisi in *K (11)* gruppi di *k (4)* simboli ognuno dei quali binario. Ogni gruppo di *k (4)* simboli è inizialmente codificato dal codificatore esterno in modo tale da fornire una parola di codice formata da *N (15)* bit, nell'esempio considerato verranno di conseguenza aggiunti quattro byte da quattro bit ognuno contenenti informazioni riguardanti la parità del singolo blocco. Successivamente ognuno degli *N (15)* gruppi di *k (4)* bit è codificato dal codificatore interno in una parola binaria di *n (7)* simboli, pertanto, ad ogni gruppo verranno aggiunti tre bit riguardanti il controllo di parità della parola binaria. L'insieme delle *N (15)* parole di codice del codice interno è una parola appartenente ad un codice concatenato di tipo $(N \cdot n, K \cdot k) = (15 \cdot 7, 11 \cdot 4) =$*(105,44),* avente un

rapporto di decodifica pari a $R_c = {K \cdot k}/{N \cdot n}$ *=1/2*,3863…; le prestazioni in questo caso sono prestazioni inferiori rispetto ad un codice *BCH* accorciato *(106,43)*, avente rapporto di decodifica pari a *Rc=1/2.46511…,* ma il processo di decodifica risulta essere molto più semplice. Tale processo di fatti viene spezzato in due parti, la prima parte della decodifica viene eseguita sul codice interno e le *N (15)* parole di codice in *N (15)* gruppi di *k (4)* simboli. Successivamente questi gruppi sono decodificati dal decodificatore esterno da *K (11)* gruppi di *k (4)* simboli binari. Nei codici concatenati tradizionali il codice esterno è quasi sempre un codice di Reed-Solomon. Il codice interno invece è un semplice codice a blocco (con decodifica hard o soft), oppure più spesso un codice convoluzionale con *decodifica di Viterbi (soft)*. In questo caso occorre introdurre un *deinterleaver* per sparpagliare i blocchi di errori prodotti dal decodificatore interno, ed un corrispondente *interleaver* tra i due codificatori. Altrimenti verrebbe facilmente superato il potere correttore del codice esterno.

4.1 Implementazione software codice concatenato

Seguendo quando descritto per la realizzazione di un codice concatenato, la realizzazione su MATLAB risulta essere immediata. Innanzitutto, si procede nel definire i parametri *p* ed *m* relativi alla generazione del campo di Galois per il codificatore esterno, che in questo caso risulta essere un codice di Reed-Solomon, successivamente, dopo aver chiesto all'utente di inserire un testo in modo tale da poterlo tradurre in codice ASCII e codificare, si inizializza il campo di Galois allo stesso modo di come fatto per la realizzazione implementativa del codice di Reed-Solomon , e si genera sia il polinomio generatore del codice BCH associato al codice Reed-Solomon, sia il polinomio generatore del codice di Reed-Solomon stesso. Una volta codificato l'input inserito dall'utente, esso è

memorizzato all'interno della variabile *cxarray*, la quale sarà utilizzata successivamente per pescare le sequenze codificate esternamente e ricodificare attraverso una codifica a blocchi relativa all'utilizzo del codice convoluzionale. Di fatto, si imposta il parametro *K* e i generatori di codice convoluzionale associati ad una specifica codifica (in questo caso si è preso come esempio *K=1* e *gen_oct=[7 3 1]*) e si procede nel codificare la sequenza di bit generata dalla codifica esterna, facendo attenzione a considerare una lunghezza di codifica coerente con l'output generato da quest'ultima. La parte implementativa relativa alla permutazione dei singoli bit associati ad una *dataword* iniziale è la medesima che si effettua andando a considerare le singole codifiche in maniera separata. L'output associato a tale implementazione è presente anch'esso all'interno dell'***Appendice A***.

Conclusioni

Al termine di questo lavoro di tesi si è ottenuto un ambiente di sviluppo stabile, la cui progettazione è stata mirata soprattutto alla creazione di codificatori di facile utilizzo. È stata condotta un'analisi approfondita alla base dell'utilizzo dei codici utilizzati nel processo di codifica del canale e si è cercato di implementare praticamente le nozioni associate attraverso l'utilizzo di algoritmi opportuni utilizzando linguaggio MATLAB. Le principali complicazioni presentatesi nello sviluppo degli algoritmi descritti sono relative principalmente alla gestione del calcolo dei vari polinomi generatori e caratteristici dei codici. In particolar modo, l'utilizzo concatenato di codici differenti, volti a realizzare una codifica di canale adeguata, ha riscontrato la necessità di sincronizzare al meglio i processi di codifica in termini di lunghezza del blocco da codificare, fornito in output dal processo di codifica Reed-Solomon esterno, in relazione al processo di codifica convoluzionale interno. Lo sviluppo futuro più interessante è principalmente legato al miglioramento implementativo volto alla ricerca di un nuovo codice che sia in grado di poter minimizzare i limiti relativi alla distanza minima o la distanza libera per i codici convoluzionali, di fatto, risulterebbe assai utile sapere quanto si sia vicini al miglior codice possibile riuscendo dunque a stabilire la bontà della distanza minima o libera di tale codice. Un altro tipo di limite riguarda ovviamente anche la possibilità intrinseca di rilevazione o correzione di errori da parte del codice, tali limiti permettono di fornire delle indicazioni riguardanti le prestazioni medie di queste classi di codici trattati che possono essere migliorati andando a scegliere parametri più accurati che permettono di avvicinare il processo di codifica ai vari limiti teorici relativi al processo di correzione. Tali sistemi, tuttavia, possono essere utilizzati al meglio non solo in sofisticati sistemi numerici via satellite, ma anche in diversi apparecchi di trasmissione dati

commerciali. Questa tendenza rappresenta il risultato di una diminuzione dei costi della componentistica numerica e della graduale attenuazione della distinzione fra codici a blocco e codici convoluzionali, derivante dallo sviluppo teorico della relativa teoria. L'aspetto di vero interesse però riguarda le prestazioni degli algoritmi di codifica e la loro realizzazione in ogni particolare applicazione. Sotto questo aspetto, la situazione si sta rapidamente evolvendo anche a causa degli sviluppi tecnologici. Ad esempio, nelle comunicazioni fra elaboratori, quando sono disponibili ampie aree di memoria, sistemi ARQ basati su codici ciclici, eventualmente combinati con alcune capacità FEC, rappresentano delle tecniche attraenti per ragioni di prestazioni e flessibilità, invece, quando i protocolli di sistema richiedono la trasmissione di blocchi di dati a lunghezza fissa, i codici a blocco sembrano i più adatti, e in particolare in questa area possono essere ottenute prestazioni elevate con basse potenze utilizzando la tecnica di codifica concatenata. In conclusione, si deve osservare che tutti i confronti in questo campo sono pesantemente influenzati dall'attuale stato dell'arte della tecnologia dei circuiti numerici integrati, di fatto, progressi in questa tecnologia potrebbero modificare in modo significativo sia questi confronti, sia il campo di applicazione.

Appendice A

In questa appendice verranno presentati gli output relativi alle operazioni di codifica effettuate e previste per questo lavoro di tesi. Le operazioni commentate svolte al fine di produrre tali output sono descritte all'interno dei singoli capitoli all'interno dell'eleaborato.

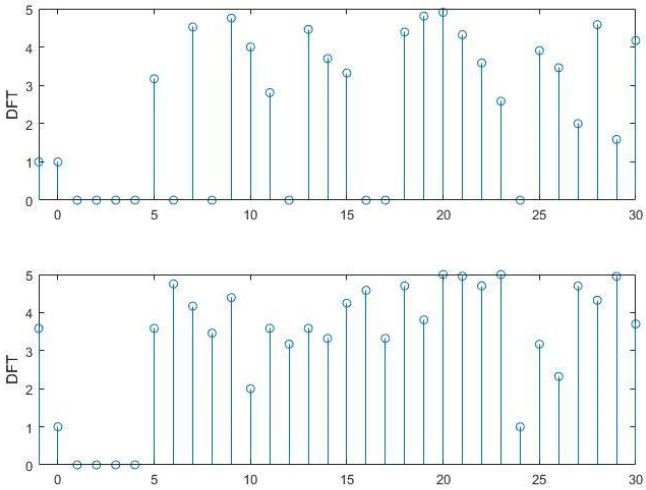
A.1 Output codifica convoluzionale

```
>> code_conv
Codice a blocco convoluzionale con bitrate nominale: 1/3 e rate effettivo: 6/24
Input: 101011 Output: 111011110011110100010001
Input: 000010 Output: 000000000000111011001000
Input: 000100 Output: 000000000111011001000000
Input: 001000 Output: 000000111011001000000000
Input: 010000 Output: 000111011001000000000000
Input: 100000 Output: 111011001000000000000000
>>
```

A.2 Output codifica Reed-Solomon

```
>> code_rs
Polinomio di default [37]: X^5 + X^2 + 1
Generatore di un codice BCH con bitrate (21,31) e capacità correttiva 2 (2 bit): X^10 + X^9 + X^8 + X^6 + X^5 + X^3 + 1
Generatore di un codice Reed-Solomon con bitrate 5 x (27,31) e capacità correttiva 2 (tra 2 e 10 bit): X^4 + alpha^24 X^3 + alpha^19 X^2 + alpha^29 X + alpha^10
Inserisci un testo da codificare: ciao

Dataword (bin): 1  1  0  0  0  1  1     1  1  0  1  0  0  1     1  1  0  0  0  0  1     1  1  0  1  1  1  1
Dataword (dec): 99 105 97 111
Dataword (poly): X^6 + X^5 + X + 1      X^6 + X^5 + X^3 + 1     X^6 + X^5 + 1       X^6 + X^5 + X^3 + X^2 + X + 1
Codeword (bin): 11100100000     11010010000     11011100000     11101010000
Codeword (poly): X^10 + X^9 + X^8 + X^5     X^10 + X^9 + X^7 + X^4      X^10 + X^9 + X^7 + X^6 + X^5        X^10 + X^9 + X^8 + X^6 + X^4
>>
```



A.3 Output codifica concatenata

```
>> conc
Inserisci un testo da codificare: prova

Polinomio di default [11]: X^3 + X + 1
Generatore di un codice BCH con bitrate (4,7) e capacità correttiva 1 (1 bit): X^3 + X + 1
Generatore di un codice Reed-Solomon con bitrate 3 x (5,7) e capacità correttiva 1 (tra 1 e 3 bit): X^2 + alpha^4 X + alpha^3
Dataword (bin): 1  1  1  0  0  0  0     1  1  1  0  0  1  0     1  1  0  1  1  1  1     1  1  1  0  1  1  0     1  1  0  0  0  0  1
Codeword (bin): 000100111       101100111       000000111       010001111       111000011

Codice a blocco convoluzionale con bitrate nominale: 1/3 e rate effettivo: 9/33
Input: 000100111 Output: 000000000111011001111100101010001
Input: 101100111 Output: 111011110100010001111100101010001
Input: 000000111 Output: 000000000000000000011110010101010001
Input: 010001111 Output: 000111011001000111100101101010001
Input: 111000011 Output: 111100101010001000000111100010001
```

Bibliografia

[1] Sergio Benedetto, Ezio Biglieri, Valentino Castellani, “Digital Trasmission Theory”, *Teoria della trasmissione numerica,* Milano 1990.

[2] Stefano Rinauro, “Coifica di Reed-Solomon per un modem digitale a 64 kbps in banda HF”, Roma 2003.

[3] Filippo Ragazzo, “Codici di Reed-Solomon”, Venezia 2013.

[4] Sandro Bellini, “Teoria dell’informazione e codici”, Varese 2015.

[5] Nicola Veracini, “Implementazione software di un algoritmo di codifica e decodifica turbo”, Pisa 2000.

[6] Simone Saviolo, “Simulazione di codici prodotto convoluzionali”, Torino 2007.

[7] Ettore Fonasini, “Codifica convoluzionale”, Padova 2003.

[8] Giovanni Garbo, Stefano Mangione, “Appunti di teoria dell’Informazione e codici”, Viterbo 2013.