Diversity

Federico Casu

Novembre 28, 2023

Diversity

Come studiato in precedenza, anche se esistono delle tecniche di trasmissione (vedi OFDM) che ci permettono di superare i fenomeni di disturbo, in generale le prestazioni del canale multipath in termini di `BER` sono più scarse, a parità di SNR, rispetto al modello di canale gaussiano.

Come possiamo aumentare le prestazioni del sistema se il canale di trasmissione è di tipo multipath[1]?

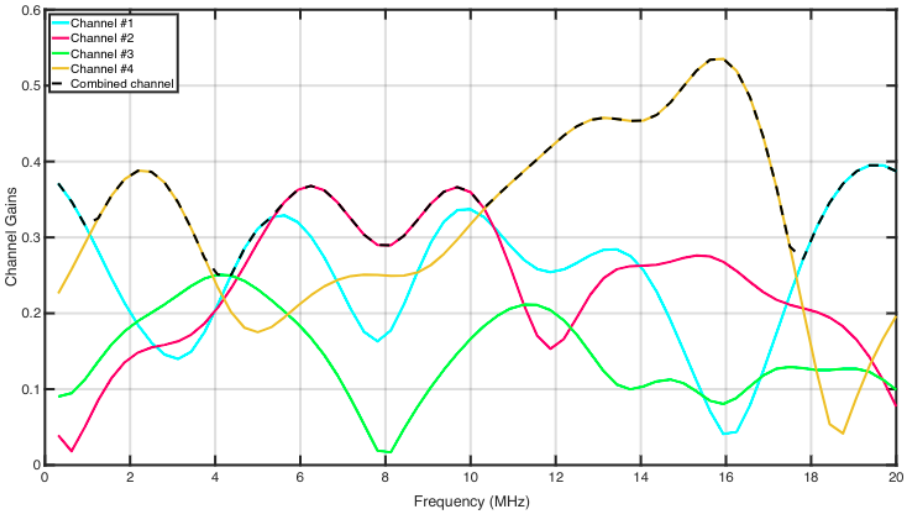

Figure 1: Esempio di *diversity* nel dominio della frequenza.

Esistono 3 diversi principali tipi di diversity:

[1] Diamo per scontato che il canale debba essere *flat fading* altrimenti avremo interferenza intersimbolica.

1. Time diversity. Esempio: un protocollo di trasmissione in cui si trasmette in slot temporali. Si noti che la distanza tra due slot di trasmissione assegnati alla sorgente debbono essere distanti più del *coherence time* del canale: se quest'ultima condizione non fosse verificata allora non si sfrutterebbe il concetto di diversity.
2. Frequency diversity. Esempio: la trasmissione avviene su più canali definiti su intervalli di frequenza non sovrapposti. Qui si deve prestare attenzione alla *coherence bandwidth*.
3. Spatial diversity. Esempio: più antenne sullo stesso dispositivo. Le antenne devono essere disposte in modo tale che distano l'una dall'altra più della *coherence distance* del canale.

Time Diversity: Coding and Interleaving

Nelle trasmissioni wireless che sfruttano il concetto di *time diversity*, lo stesso segnale viene trasmesso in istanti temporali differenti. In alternativa, viene aggiunto un codice di *forward error correction* e il messaggio è diffuso nel tempo mediante la codifica dei bit prima della trasmissione. In questo modo, si evitano i *burst* di errori, semplificando la correzione di quest'ultimi.

Block Codes

I block codes sono molto semplici: il processo di encoding consiste nell'espandere i k bits informativi in modo tale che il decoder, in ricezione, possa sfruttare l'espansione per:

1. Trovare gli errori commessi in trasmissione.
2. Correggere gli errori commessi in trasmissione.

Dal punto di vista matematico, i block codes possono essere rappresentati come segue:

$$\mathsf{d} = \mathsf{uG},\ \mathsf{d} \in \{0,1\}^{\mathsf{n}},\ \mathsf{u} \in \{0,1\}^{1\times\mathsf{k}},\ \mathsf{G} \in \{0,1\}^{\mathsf{k}\times\mathsf{n}}$$

La matrice G è detta *generatrice*.

Esempio: **Parity Check Code**. L'encoder aggiunge un bit di parità a ogni parola di 7 bit: il bit di parità è `1` se la parola contiene un numero dispari di `1`, `0` se la parola contiene un numero pari di `1`. L'output dell'encoder è una parola di 8 bit che contiene sempre un numero pari di `1`.

La matrice generatrice è la seguente:

$$[\mathsf{I}_7, 1_7]$$

Siamo sicuri che la matrice generatrice sia corretta? Dimostriamolo:

1. Supponiamo di avere un parola di 7 bit $\mathsf{u} = [\mathsf{u}_0, \mathsf{u}_1, \mathsf{u}_2, \mathsf{u}_3, \mathsf{u}_4, \mathsf{u}_5, \mathsf{u}_6]$.

2. La matrice generatrice è la seguente:

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

3. Il prodotto uG è il seguente:

$$\begin{aligned} \mathsf{u} \cdot \mathsf{G} &= [\mathsf{u}_0, \mathsf{u}_1, \mathsf{u}_2, \mathsf{u}_3, \mathsf{u}_4, \mathsf{u}_5, \mathsf{u}_6] \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \\ &= [\mathsf{u}_0, \mathsf{u}_1, \mathsf{u}_2, \mathsf{u}_3, \mathsf{u}_4, \mathsf{u}_5, \mathsf{u}_6, \sum_{\mathsf{i}=0}^{6} \mathsf{u}_\mathsf{i}] \end{aligned}$$

4. Se $\sum_{\mathsf{i}=0}^{6} \mathsf{u}_\mathsf{i}$ è pari allora sappiamo che in $\mathsf{GF}(2)$ la somma sarà nulla ($\sum_{\mathsf{i}=0}^{6} \mathsf{u}_\mathsf{i} = 0$). Nel caso opposto, la somma sarà non nulla e, conseguentemente, il numero di bit della parola trasmessa diventa pari.

Non tutti gli errori vengono rilevati. Le trasmissioni che introducono un numero pari di errori passano senza destare alcun sospetto perchè il controllo sulla parità è corretto!

Fino ad ora non abbiamo fatto altro che definire un sistema che introduce delle informazioni ridondanti: dove si sfrutta la diversity che tanto abbiamo acclamato? **Data Retransmission**:

1. Il ricevitore invia un `ACK` se la parolaricevuta è corretta, un `NACK` in caso contrario.
2. Dopo aver ricevuto un `NACK`, il trasmettitore ritrasmette la parola associata al `NACK` ricevuto.

Il protocollo appena descritto è detto ARQ (**A**utomated **R**epeated re*Q*uest). Tale protocollo sfrutta la diversità temporale del canale ritrasmettendo i dati dopo un intervallo di tempo $\mathsf{T}_{\mathsf{ARQ}}$, un periodo più lungo rispetto al tempo di coerenza del canale T_C. La nuova trasmissione sperimenterà un canale diverso e, si spera, migliore.

Dato un canale di comunicazione con larghezza di banda B, Shannon ha dimostrato che la *capacità del canale* C può essere calcolata come

$$\mathsf{C} = \mathsf{B} \cdot \log_2(1 + \mathsf{SNR}) \quad \text{b/s}$$

- Per qualsiasi trasmissione con rate $\mathsf{R} < \mathsf{C}$, è possibile trovare un codice di correzione degli errori in modo che la probabilità di errore sia $\mathsf{P_e} < \epsilon >$, con ϵ arbitrariamente piccolo.
- Al contrario, se $\mathsf{R} > \mathsf{C}$ non è possibile trovare un codice che possa rendere la probabilità di errore arbitrariamente piccola.

Il ragionamento è il seguente: una volta che ho stimato la capacità del canale, adopero un rate di trasmissione che sia minore della capacità del canale e cerco un codice che mi permetta di diminuire la probabilità d'errore fino alla soglia desiderata.

Decoder: Optimal Strategy

In $\mathsf{GF}(2)$, la distanza tra le parole è calcolata come il numero di bit diversi nelle due stringhe di bit (distanza di Hamming).

- Dopo aver ricevuto la stringa $\hat{\mathsf{x}}$ (lunga n bits), il decodificatore seleziona la parola $\hat{\mathsf{d}}$ che ha la distanza minima da $\hat{\mathsf{x}}$:

$$\hat{\mathsf{d}} = \mathsf{argmin}(\mathsf{d}, \hat{\mathsf{x}})$$

- L'errore si verifica quando, a causa del rumore, la parola $\hat{\mathsf{x}}$ è più vicina a una parola di codice diversa da quella trasmessa.
- Maggiore è la distanza di Hamming tra le parole di codice, maggiore è la capacità di correzione degli errori di un codice a blocchi.

Le proprietà di correzione e rilevamento di errori di un codice a blocchi dipendono dalla distanza minima di Hamming $\mathsf{d_{min}}$ tra le parole di codice, calcolata come la distanza minima tra tutte le parole di codice di un dato codice.

- Un codice a blocchi con distanza minima $\mathsf{d_{min}}$ può rilevare fino a $\mathsf{d_{min}} - 1$ errori e correggere fino a $\lfloor \frac{\mathsf{d_{min}} - 1}{2} \rfloor$ errori.
- I codici in cui la distanza tra le parole è grande sono più robusti contro il rumore e il fading rispetto a codici in cui la distanza è piccola.

Vediamo un intuizione di quest'ultimo concetto:

- Supponiamo di ricevera una parola $\hat{\mathsf{x}}$. Visto che ogni parola del codice che stiamo usando dista almeno $\mathsf{d_{min}}$ da una qualsiasi altra parola del codice, se $\hat{\mathsf{x}}$ contiene $\leq \mathsf{d_{min}} - 1$ errori allora $\hat{\mathsf{x}}$ non è una parola valida e siamo certi che la trasmissione ha introdotto degli errori.

- Supponiamo di inviare la parola $d^{(i)}$. Al ricevitore arriva la parola $\hat{x}$. In particolare, $d_H(d^{(i)}, \hat{x}) > d_H(d^{(i+1)}, \hat{x})$ e di conseguenza il decoder sceglierebbe la parola $d^{(i+1)}$, commettendo un errore. Ciò è accaduto perchè la parola ricevuta $\hat{x}$ è affetta da più di $\lfloor \frac{d_{min}-1}{2} \rfloor$ errori quindi, durante il processo di error correction, il decoder sceglie la parola del codice che risulta essere più vicina alla parola inviata.

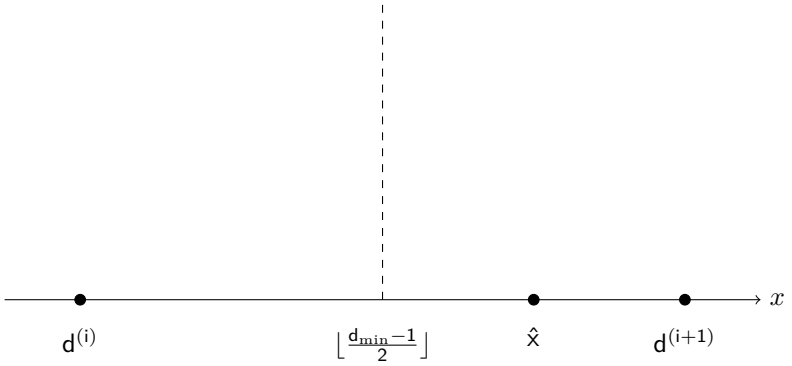

Figure 2: $\hat{x}$ non può essere corretto perchè dista più di $\lfloor \frac{d_{min}-1}{2} \rfloor$ dalla parola inviata dal trasmettitore, in questo caso specifico $d^{(i)}$.

Avendo fissato il rapporto $R = \frac{k}{n}$, d_{min} è più grande quando k e n sono grandi; sfortunatamente, anche la complessità del ricevitore cresce con k.

Convolutional Codes

Contrariamente ai codici a blocchi in forma sistemica, i codici convoluzionali in generale non sono *sistematici*[2]. In un codice convoluzionale, il mittente invia solo i bit di parità. L'encoder utilizza una finestra scorrevole per calcolare $n > 1$ bit di parità combinando vari sottoinsiemi di bit nella finestra, in modo che il processo di codifica possa essere visto come una convoluzione in $GF(2)$.

La differenza più importante rispetto ai codici a blocchi è che ora abbiamo un sistema con memoria. La memoria L dell'encoder è il numero ingressi di k bit che influenzano i n bit in uscita. Di conseguenza, l'uscita dell'encoder dipende sia dall'input corrente $u^{(i)}$ sia dai $L-1$ ingressi precedenti $u^{(i-1)}, \ldots, u^{(i-L+1)}$. La risposta impulsiva di ciascuna delle n convoluzioni è data da un diverso vettore generatore di lunghezza $k \cdot L$. Essendo il risultato di una convoluzione, i bit codificati non sono organizzati in blocchi, ma costituiscono un flusso continuo di dati.

[2]Un codice è detto *sistematico* se la stringa di bit trasmessa è composta dai bit del messaggio seguiti dai bit di parità.

Un encoder convoluzionale può essere descritto mediante la tripla (n, k, L). Facciamo un esempio: $(n = 2, k = 1, L = 3)$.

- $k = 1$: ogni T, l'encoder convoluzionale riceve un bit in input.
- $n = 2$: ogni T, l'encoder convoluzionale produce **due** bit in output.
- $L = 3$: ogni bit prodotto dipende dal bit in ingresso all'istante $i \cdot T$ e dai due bit immediatamente precedenti $u^{(i-1)}$ e $u^{(i-2)}$.

L'output dell'encoder può essere descritto come il prodotto scalare tra due vettori. Prendiamo in considerazione l'esempio precedente $(n = 2, k = 1, L = 3)$:

$$d_n^{(i)} = \sum_{j=0}^{3-1} u^{(i-j)} \cdot g(j)_n,\ n \in \{0, 1\}$$

Per capire a fondo il funzionamento dei codici convoluzionali, analizziamo nel dettaglio il seguente esempio:

$$(n = 2, k = 1, L = 3),\ g_1 = [1, 1, 1],\ g_2 = [1, 0, 1]$$

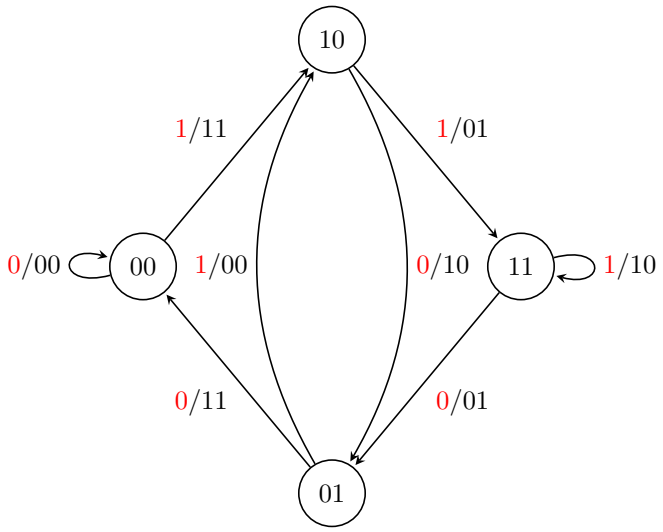

Figure 3: Macchina a stati finiti dell'encoder $(n = 2, k = 1, L = 3)$.

Ad ogni T, l'output dell'encoder dipende dal bit in ingresso e dai due bit precedenti. I due bit appena menzionati rappresentano la memoria (o stato) dell'encoder. Di seguito elenchiamo ogni possibile output in funzione del *1)* bit

in ingresso e *2)* lo stato dell'encoder:

$$u^{(i)} = 0,\ (0,0) \rightarrow d_1^{(i)} = \sum_{j=0}^{2} u^{(i-j)} \cdot g_1(j) = u^{(i)} + u^{(i-1)} + u^{(i-2)} = 0 + 0 + 0 = 0$$

$$u^{(i)} = 0,\ (0,0) \rightarrow d_2^{(i)} = \sum_{j=0}^{2} u^{(i-j)} \cdot g_1(j) = u^{(i)} + u^{(i-2)} = 0 + 0 = 0$$

$$u^{(i)} = 1,\ (0,0) \rightarrow d_1^{(i)} = \sum_{j=0}^{2} u^{(i-j)} \cdot g_1(j) = u^{(i)} + u^{(i-1)} + u^{(i-2)} = 1 + 0 + 0 = 1$$

$$u^{(i)} = 1,\ (0,0) \rightarrow d_2^{(i)} = \sum_{j=0}^{2} u^{(i-j)} \cdot g_1(j) = u^{(i)} + u^{(i-2)} = 1 + 0 = 1$$

$$u^{(i)} = 0,\ (1,0) \rightarrow d_1^{(i)} = \sum_{j=0}^{2} u^{(i-j)} \cdot g_1(j) = u^{(i)} + u^{(i-1)} + u^{(i-2)} = 0 + 1 + 0 = 1$$

$$u^{(i)} = 0,\ (1,0) \rightarrow d_2^{(i)} = \sum_{j=0}^{2} u^{(i-j)} \cdot g_1(j) = u^{(i)} + u^{(i-2)} = 0 + 0 = 0$$

$$u^{(i)} = 1,\ (1,0) \rightarrow d_1^{(i)} = \sum_{j=0}^{2} u^{(i-j)} \cdot g_1(j) = u^{(i)} + u^{(i-1)} + u^{(i-2)} = 1 + 1 + 0 = 0$$

$$u^{(i)} = 1,\ (1,0) \rightarrow d_2^{(i)} = \sum_{j=0}^{2} u^{(i-j)} \cdot g_1(j) = u^{(i)} + u^{(i-2)} = 1 + 0 = 1$$

$$u^{(i)} = 0,\ (0,1) \rightarrow d_1^{(i)} = \sum_{j=0}^{2} u^{(i-j)} \cdot g_1(j) = u^{(i)} + u^{(i-1)} + u^{(i-2)} = 0 + 0 + 1 = 1$$

$$u^{(i)} = 0,\ (0,1) \rightarrow d_2^{(i)} = \sum_{j=0}^{2} u^{(i-j)} \cdot g_1(j) = u^{(i)} + u^{(i-2)} = 0 + 1 = 1$$

$$u^{(i)} = 1,\ (0,1) \rightarrow d_1^{(i)} = \sum_{j=0}^{2} u^{(i-j)} \cdot g_1(j) = u^{(i)} + u^{(i-1)} + u^{(i-2)} = 1 + 0 + 1 = 0$$

$$u^{(i)} = 1,\ (0,1) \rightarrow d_2^{(i)} = \sum_{j=0}^{2} u^{(i-j)} \cdot g_1(j) = u^{(i)} + u^{(i-2)} = 1 + 1 = 0$$

$$u^{(i)} = 0,\ (1,1) \rightarrow d_1^{(i)} = \sum_{j=0}^{2} u^{(i-j)} \cdot g_1(j) = u^{(i)} + u^{(i-1)} + u^{(i-2)} = 0 + 1 + 1 = 0$$

$$u^{(i)} = 0,\ (1,1) \rightarrow d_2^{(i)} = \sum_{j=0}^{2} u^{(i-j)} \cdot g_1(j) = u^{(i)} + u^{(i-2)} = 0 + 1 = 1$$

$$u^{(i)} = 1,\ (1,1) \rightarrow d_1^{(i)} = \sum_{j=0}^{2} u^{(i-j)} \cdot g_1(j) = u^{(i)} + u^{(i-1)} + u^{(i-2)} = 1 + 1 + 1 = 1$$

$$u^{(i)} = 1,\ (1,1) \rightarrow d_2^{(i)} = \sum_{j=0}^{2} u^{(i-j)} \cdot g_1(j) = u^{(i)} + u^{(i-2)} = 1 + 1 = 0$$

Adesso arriva il bello: come avviene la decodifica?

> A differenza dei codici a blocchi, cui ogni blocco poteva essere codificato senza la necessità di conoscere i blocchi precedenti/successivi, i codici convoluzionali posseggono il concetto di memoria: la parola in input all'istante i partecipa alla codifica delle successive $L - 1$ parole!

Invece, per quanto riguarda la strategia di decisione ottima, sia i codici a blocchi sia i codici convoluzionali adottano lo stesso approccio: decisore a minima distanza (di Hamming).

Supponiamo di ricevere una sequenza di N parole. Visto che ogni parola è composta da n bits, ciascuno di questi ottenuto tramite convoluzione, la sequenza di bit ricevuta è lunga $n \cdot N$. In realtà, i $n \cdot N$ bits ricevuti sono bit di parità: i bit che effettivamente sono stati trasmessi sono $k \cdot N$. Da questo deduciamo che possono esistere $2^{k \cdot N}$ sequenze di bits diverse che possono essere ricevute.

Un decisore a distanza minima sceglie la sequenza di bit $\bar{d}$, quest'ultima lunga $n \cdot N$, tale che

$$\bar{d} = \operatorname{argmin} \left\{ d_H(d_i, \hat{x}) \right\},\ i = 1, \ldots, 2^{k \cdot N}$$

dove $\hat{x}$ rappresenta la sequenza di bits ricevuta.

Trovare la sequenza di bits $\bar{d}$ tale che la distanza di Hamming da $\hat{x}$ sia minima richiede calcolare la distanza di Hamming tra $\hat{x}$ e tutte le possibili $2^{k \cdot N}$ sequenze valide! La complessità è esponenziale nel numero di bits inviati/ricevuti!

Algoritmo di Viterbi

Prima di introdurre l'algoritmo di Viterbi è comodo imparare ad utilizzare i *diagrammi di trellis*. L'evoluzione temporale dell'encoder può essere rappresentata mediante diagramma di trellis. Tale diagramma illustra l'evoluzione degli stati dell'encoder in funzione del tempo. Qualsiasi sequenza di input e la parola codificata corrispondente possono essere rappresentate come un percorso sul diagramma. Ogni nodo del diagramma rappresenta uno stato dell'encoder.

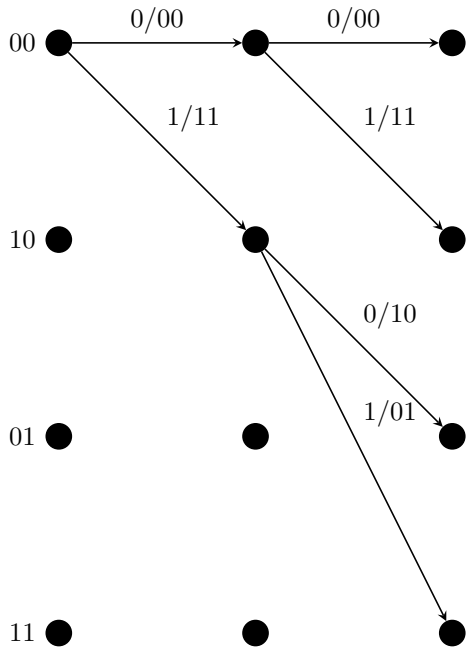

Figure 4: Esempio di diagramma di trellis.

Ora possiamo studiare l'idea alla base dell'algoritmo di Viterbi. Il decoder implementa una strategia di decisione a minima distanza. Se la sequenza di bits ricevuta è composta da N parole di codice, la distanza di Hamming tra una sequenza d e la sequenza ricevuta $\hat{\mathsf{x}}$ è calcolata come segue:

$$\mathsf{d_H}(\mathsf{d}, \hat{\mathsf{x}}) = \sum_{\mathsf{i}=1}^{\mathsf{N}} \mathsf{d_H}(\mathsf{d}^{(\mathsf{i})}, \hat{\mathsf{x}}_\mathsf{i})$$

ovvero la distanza di Hamming è calcolata come la somma delle distanze di hamming tra la parola ricevuta al signaling time i e la parola $\mathsf{d_i}$ ottenuta valutando l'evoluzione temporale della macchina a stati dell'encoder.

Analizziamo la distanza di Hamming tra una sequenza d e la sequenza $\hat{\mathsf{x}}$ ad un certo punto della trasmissione $\mathsf{k} < \mathsf{N}$.

$$\mathsf{d_H}(\mathsf{d}, \hat{\mathsf{x}})|_\mathsf{k} = \sum_{\mathsf{i}=1}^{\mathsf{k}} \mathsf{d_H}(\mathsf{d}^{(\mathsf{i})}, \hat{\mathsf{x}}_\mathsf{i})$$

Supponiamo di avere due sequenze valide, $\mathsf{d_i}$ e $\mathsf{d_j}$, e quest'ultimi al signaling time k convergono nello stato 10. La distanza di Hamming accumulata fino ad ora

dalle due sequenze può essere calcolata come:

$$\Lambda_k(d_i) = d_H(d_i, \hat{x})|_k = \sum_{m=1}^{k} d_H(d_i^{(m)}, \hat{x}_m)$$

$$\Lambda_k(d_j) = d_H(d_j, \hat{x})|_k = \sum_{m=1}^{k} d_H(d_j^{(m)}, \hat{x}_m)$$

Dallo stato 10 si prosegue verso lo stato 01. La distanza di Hamming tra una sequenza valida e $\hat{x}$ al signaling time $k+1$ può assumere due valori differenti (ricorda che ci sono due percosi che convergono nello stato 10 al signaling time k):

$$\Lambda_{k+1}(d) = \Lambda_k(d_i) + d_H(d^{(k+1)}, \hat{x}_{k+1})$$
$$\Lambda_{k+1}(d) = \Lambda_k(d_j) + d_H(d^{(k+1)}, \hat{x}_{k+1})$$

Il decoder sceglierà il percorso che, appunto, minimizza la distanza di Hamming. Visto che la distanza totale associata ai due percorsi ha una componente uguale per entrambi (ovvero la distanza di Hamming a partire da $k+1$), possiamo scartare il percorso con distanza maggiore visto che comunque verrà scartato dal decoder!

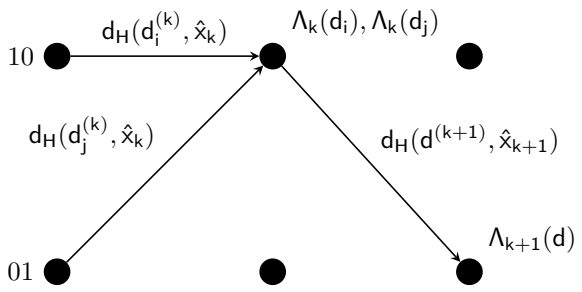

Figure 5: Algoritmo di Viterbi. Esempio pratico.

In generale, assumiamo che ogni parola di codice inizi dallo stato 0 e termini nello stato 0, in modo che tutti i percorsi abbiano origine dallo stato 0 e alla fine convergano nello stato 0. Di conseguenza, il percorso soppravissuto che porta allo stato 0 rappresenta l'output dell'algoritmo e la metrica cumulata finale Λ è il numero di errori corretti. La complessità dell'algoritmo di Viterbi è lineare rispetto a N, non esponenziale!

Interleaving

I codici convoluzionali sono principalmente adatti per canali *memoryless* con eventi di errore distribuiti in modo uniforme e non correlati. I canali con fading

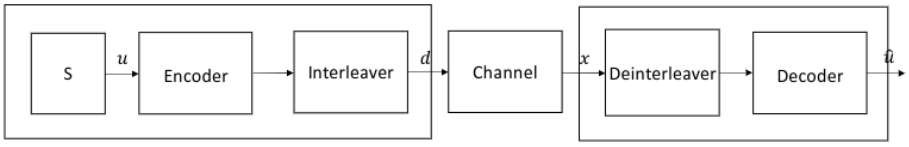

Figure 6: Diagramma a blocchi di un sistema Tx-Rx che adopera il meccanismo di interleaving.

tendono a causare burst di errori. L'interleaving fa sì che il canale sembri un canale memoryless al decoder e tende a decorrelare gli eventi di errore.

L'interleaving viene realizzato diffondendo i simboli codificati nel tempo o nella frequenza prima della trasmissione. In ricezione, si esegue l'operazione inversa (deinterleaving) della sequenza ricevuta. L'interleaving fa sì che gli errori (quest'ultimi molto ravvicinati tra loro a causa del canale fortemente fading in quel punto) sembrino casuali, consentendo così ai codici convoluzionali di ottenere prestazioni migliori. Il costo dell'interleaving è la *latenza*: sia al trasmettitore che al ricevitore è necessario avere l'intero blocco di dati per avviare il processo di codifica/decodifica. C'è un compromesso: maggiore è la profondità dell'interleaver K, maggiormente gli errori sono decorrelati, ma al contempo crescono la latenza e il ritardo.