L'ALGORITMO DI VITERBI

Federico Campeotto, matricola n° 80239
Riccardo Vicedomini, matricola n° 80261

Anno Accademico 2009-2010

Introduzione

Figura 1: A. Viterbi

Andrew J. Viterbi è, senza ombra di dubbio, uno dei pionieri nel campo delle comunicazioni wireless. Nato a Bergamo il 9 Marzo 1935, ha successivamente sostituito il nome al momento della naturalizzazione. Il padre, Achille, era dottore a capo di una delle tante famiglie ebree mentre la madre, Maria Luria, era casalinga. Achille chiuse l'attività e lasciò l'Italia, assieme alla famiglia, a causa delle leggi razziali emanate nel 1938 dal regime di Mussolini che colpivano la popolazione ebrea in Italia. Giunse a New York dove viveva un cugino, il 27 Agosto 1939, esattamente 5 giorni prima dell'invasione della Polonia da parte della Germania nazista e, quindi, dell'inizio della seconda guerra mondiale. Due anni più tardi la famiglia si trasferì nuovamente, questa volta a Boston dove il padre poté riprendere l'attività di oftalmologo. Andrew studiò alla "Boston Latin School". Diplomatosi nel 1952, quarto su 255 studenti, si iscrisse alla Massachusetts Institute of Technology, dove seguì i corsi di Claude Elwood Shannon, Norbert Wiener, Bruno Rossi e Roberto Fano per laurearsi in teoria delle comunicazioni. Nel 1956 conobbe Erna Finci, emigrata anche essa, assieme alla sua famiglia, nel 1950 in California. Nel 1957, Andrew Viterbi, assieme alla sua nuova famiglia, si trasferì definitivamente a Los Angeles dove lavorò in un gruppo di ricerca capitanato da Rechtin, guru delle comunicazioni, presso il *Jet Propulsion Laboratoy* (JPL) a Pasadena. In quel periodo i suoi interessi si concentrarono su una tecnica trasmissiva innovativa chiamata "spread spectrum" (a spettro distribuito), che consiste nell'impiegare per la trasmissione una gamma di frequenze più ampia rispetto a quella strettamente necessaria per l'invio dell'informazione desiderata. In questo campo Viterbi ottenne importanti risultati teorici nell'ambito della comunicazione digitale, occupandosi in particolar modo di metodi di codifica e decodifica. Nell'Ottobre del 1957, appena due mesi dopo l'adesione di Viterbi al JPL, l'unione sovietica lanciò il primo satellite nell'orbita terrestre (Sputnik 1). Gli Stati Uniti, per non perdere la supremazia nel dominio dell'orbita spaziale, spronarono nuovamente la ricerca. Nuovamente, Viterbi si trovò nel posto giusto al momento giusto: il gruppo di ricerca di Rechtin passò sotto il controllo dell programma di difesa spaziale americano. Viterbi lavorò a lungo nel gruppo di ricerca e diede così un pesante contributo alla costruzione dell'*Explorer 1*, il primo satellite degli stati uniti, lanciato nel febbraio del 1958. Le idee di Viterbi

diedero un contributo dominante alla soluzione di due importanti problemi che dovettero essere risolti per mantenere i contatti con il satellite: la debolezza dei segnali ricevuti, dovuta alla distanza, ed il continuo cambiamento della frequenza dei segnali stessi, causato dal movimento orbitale. Un anno dopo aver conseguito il dottorato nel 1962 alla *University of Southern California*, Viterbi intraprese la carriera accademica presso la *University of California* a Los Angeles, con un corso di comunicazioni digitali e teoria dell'informazione. E' proprio di quegli anni il percorso di studio ed intuizione che porterà all'elaborazione di un "algoritmo probabilistico non sequenziale", presentato a maggio del '66 con il titolo "Limiti di errore per codici convoluzionali e un algoritmo di decodificazione asintoticamente ottimo". L'algoritmo di Viterbi, usato per codificare trasmissioni digitali, è presente nelle trasmissione dati GSM, nei sistemi telemetrici che hanno permesso il lancio dei primi missili americani, nei telefoni cellulari come codice di correzione degli errori, nell'analisi del DNA ed in molte altre applicazioni basate sul modello di Markov nascosto (catene di Markov nelle quali sono osservabili gli eventi ma non gli stati) dove, dati i parametri del modello, si utilizza l'algoritmo suddetto per trovare la sequenza più probabile che potrebbe generare una data sequenza di uscita. Purtroppo, non avendolo brevettato, il suo inventore non guadagnò nulla sui diritti. A renderlo ricco, invece, è stata l'idea del CDMA (Code Division Multiple Access), lo standard di trasmissione dell'UMTS che permetteva prestazioni nettamente superiori rispetto alle tecniche tradizionali. Usando il CDMA, tutte le trasmissioni avvengono contemporaneamente ed utilizzando la stessa banda di frequenze. La caratteristica innovativa ed unica del CDMA è che ad ogni trasmissione dati si assegna uno specifico codice, diverso da tutti gli altri. La struttura matematica del CDMA garantisce che comunicazioni simultanee non si disturbino a vicenda: decodificando tutti i segnali presenti nell'etere con uno specifico codice si ottengono esclusivamente i dati originariamente codificati con quello stesso codice, mentre le comunicazioni degli altri utenti, che usano codici diversi, appaiono come rumore che la decodifica è in grado di scartare tramite, appunto, l'algoritmo di Viterbi. E' grazie a queste idee e all'implementazione del suo algoritmo sia in hardware che in software che Viterbi ha co-fondato diverse compagnie tra cui la *Qualcomm*, leader nelle comunicazioni cellulari. Nel 1969 ha fondato, assieme ad Irwin Jacobs, l'azienda Linkabit, un piccolo fornitore delle forze armate. La tecnologia sviluppata alla Linkabit era l'inizio delle moderne "Wide Area Network" o dei modem WiFi nei computer portatili. Nel 1980 la Linkabit fu acquistata dalla Microwave Associates Communication (M/A-COM) in Burlington. La fusione diede la possibilità di espandere i prodotti della Linkabit anche in ambito commerciale. Nel 1993 il protocollo CDMA per le comunicazioni cellulari venne standardizzato. Nel 1995 la Qualcomm

produsse il primo cellulare con il sistema di comunicazione CDMA, nel 2006 ne furono venduti più di 300 milioni. Nel 2000, all'età di 65 anni, Viterbi si ritirò dalla Qualcomm per occuparsi principalmente di attività universitarie ed investimenti su compagnie emergenti. La "Viterbi Family Foundation" è la fondazione a scopo filantropico tramite la quale Viterbi ha elargito (e continua ad elargire) somme significative di denaro a molte istituzioni, tra le quali il liceo frequentato da ragazzo da Viterbi (Boston Latin School), l'MIT ed il Technion Israel Institute of Technology. Il 2 marzo 2004, la scuola di ingegneria dell'University of Southern California (USC), ove Viterbi aveva conseguito (nel 1962) il dottorato di ricerca è stata rinominata *Andrew and Erna Viterbi School of Engineering*. In quell'occasione, Viterbi ha devoluto alla USC la somma di 52 milioni di dollari. Nel 2000 era classificato 386° per ricchezza tra gli statiunitensi. Dal 2003 è il presidente della società di investimenti *The Viterbi Group*. Infine, nel settembre del 2008, Viterbi si è aggiudicato la "National Medal of Science" per lo sviluppo dell' "algoritmo di Viterbi" e per il suo contributo alla tecnologia wireless CDMA per aver dato un enorme contributo alla teoria e la pratica delle comunicazioni digitali.

Descrizione del problema

Nella sua forma più generale, l'algoritmo di Viterbi può essere visto come una soluzione del problema della stima a massima probabilità a posteriori (MAP) di una sequenza di stati di un processo stocastico markoviano con un numero finito di stati e a tempo discreto (ad ogni istante t è possibile estrarre dal processo una variabile casuale discreta). In altre parole, l'algoritmo di Viterbi viene utilizzato per trovare la miglior sequenza di stati (detta "Viterbi path") in una sequenza di eventi osservati in un processo markoviano.

Un processo stocastico markoviano è un processo stocastico nel quale la probabilità di transazione che determina il passaggio ad uno stato di sistema dipende unicamente dallo stato di sistema immediatamente precedente e non dal *come* si è giunti a tale stato. Entrando più nel dettaglio, il processo di Markov è, generalmente, caratterizzato come segue. Il tempo è discreto. Lo stato x_k al tempo k è un numero m appartenente all'insieme finito di stati M dove $1 \leq m \leq M$; ad esempio, lo spazio degli stati X può essere semplicemente $\{1, 2, \cdots, M\}$. Assumeremo, inizialmente, che il processo avvenga solamente dal tempo 0 al tempo K e che lo stato iniziale e lo stato finale, rispettivamente x_0 e x_k siano noti; la sequenza degli stati viene quindi rappresentata tramite un vettore finito $x = (x_0, \cdots, x_k)$. Nel seguito vedremo che l'estensione a sequenze infinite è immediata.

In un processo di Markov, la probabilità $P(x_{k+1} \mid x_0, x_1, \cdots, x_k)$ di essere nello stato x_{k+1} al tempo $k+1$, dati tutti gli stati fino al tempo k, dipende solamente dallo stato x_k al tempo k:

$$P(x_{k+1} \mid x_0, x_1, \cdots, x_k) = P(x_{k+1} \mid x_k)$$

E' conveniente definire la *transizione* ξ_k al tempo k come la coppia di stati (x_{k+1}, x_k):

$$\xi_k \triangleq (x_{k+1}, x_k)$$

Sia A l'insieme delle transizioni $\xi_k = (x_{k+1}, x_k)$ per ogni $P(x_{k+1} \mid x_k) \neq 0$ e $|A|$ la sua cardinalità. Ovviamente $|A| \leq M^2$. Come è evidente, esiste una corrispondenza *uno-a-uno* tra la sequenza di stati $x = (x_0, \ldots, x_k)$ e la sequenza delle transizioni $\xi = (\xi_0, \ldots, \xi_{k-1})$. Si assume che il processo osservato sia privo di memoria, ovvero tale che esista una sequenza z di osservazioni z_k dove z_k dipende in modo probabilistico solamente dalla transizione ξ_k al tempo k:

$$P(z|x) = P(z|\xi) = \prod_{k=0}^{k-1} P(z_k|\xi_k)$$

Si noti che $P(A \cup B) = P(A) \cdot P(B)$, con A e B eventi indipendenti.

Nel linguaggio della teoria dell'informazione, la sequenza di osservazioni z può essere descritta come l'output di un qualche canale privo di memoria il cui input è la sequenza ξ (come illustrato in Figura 2). Questa formulazione

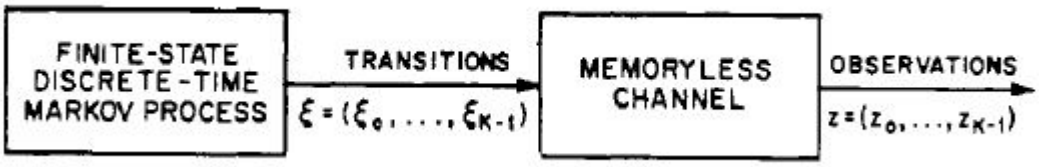

Figura 2: Sequenza di osservazioni z.

sussume i seguenti casi speciali:

1. Il caso in cui z_k dipenda solamente dallo stato x_k:

$$P(z|x) = \prod_{k} P(z_k|\xi_k)$$

2. Il caso in cui z_k dipenda in modo probabilistico da un output y_k del processo al tempo k, dove y_k è a sua volta (una funzione deterministica) dipendente dalla transizione ξ_k o dallo stato x_k.

Esempio Il seguente modello si presenta frequentemente nelle comunicazioni digitali. Si ha una sequenza di input $u = (u_0, u_1, \dots)$, dove u_k è generato indipendentemente secondo qualche distribuzione probabilistica $P(u_k)$ e può prendere uno di un numero finito di valori, ad esempio m. Si ha, inoltre, una sequenza y di segnale priva di disturbo, non osservabile, dove ogni y_k è una qualche funzione deterministica che dipende dall'input presente e dai precedenti v input:

$$y_k = f(u_k, \dots, u_{k-v})$$

La sequenza di osservazioni z è l'output di un canale privo di memoria dove l'input è y. Possiamo chiamare il suddetto processo come processo con registri a scorrimento ("shift-register process"), dato che può essere modellato da un registro a scorrimento di lunghezza v ed input u_k (vedi Figura 3).

Notare che un registro è un circuito digitale che ha due funzioni base: la memorizzazione e lo spostamento dei dati. In particolare, i *registri a scorrimento* (*shift register*) sono componenti utilizzati nell'elettronica digitale costituiti da una catena di celle di memoria ad un bit interconnesse tra loro (solitamente dei flip-flop), ad ogni impulso di clock essi consentono lo scorrimento dei bit da una cella a quella immediatamente adiacente. Lo scorrimento può avvenire verso destra, verso sinistra o in alcuni tipi sia verso destra che verso sinistra in base allo stato della linea di controllo. Per

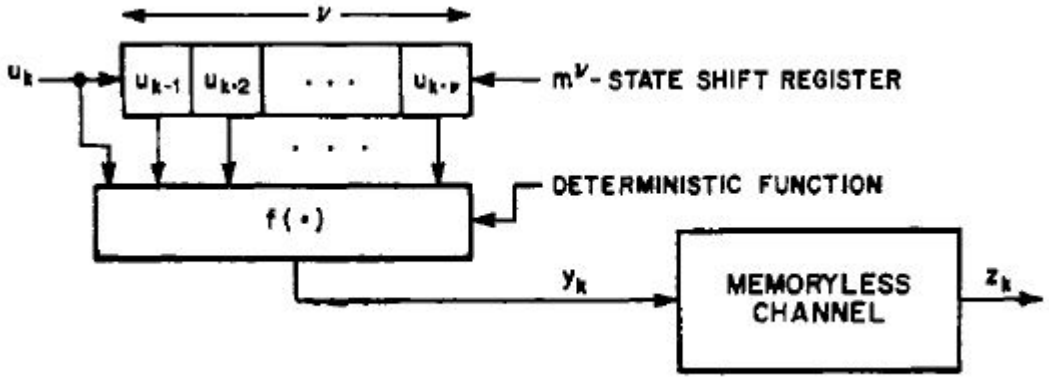

Figura 3: Processo con registri a scorrimento.

completare la corrispondenza con il nostro modello generale definiamo:

1. lo stato

$$x_k \triangleq (u_{k-1}, \dots, u_{k-v})$$

2. la transizione

$$\xi_k \triangleq (u_k, u_{k-1}, \ldots, u_{k-v})$$

Il numero di stati è, dunque, $|X| = m^v$ e le transizioni sono $|A| = m^{v+1}$. Se la sequenza di input "inizia" all'istante 0 e "finisce" all' istante $K - v$, ad esempio:

$$u = (\ldots, 0, u_0, u_1, \ldots, u_{K-v}, 0, 0, \ldots),$$

allora il processo a scorrimento di registro comincia effettivamente al tempo 0 e termina al tempo K con $x_0 = x_K = (0, 0, \ldots, 0)$.

Infine, definiamo il problema per cui l'algoritmo di Viterbi ne è la soluzione: data una sequenza z di osservazioni di un processo di Markov con un numero di stati finiti e a tempo discreto, trovare la sequenza di stati x per cui la probabilità $P(x|z)$ è massima. Alternativamente, trovare la sequenza di transizioni ξ per cui $P(\xi|z)$ è massima (come detto sopra, esiste una corrispondenza uno-a-uno fra x e ξ). Nel modello a scorrimento di registri si tratta di trovare la sequenza più probabile di input u oppure, ancora, trovare la sequenza più probabile di segnale y (corrispondenza uno-a-uno fra x e $u - y$). E' ben noto che questa regola MAP minimizza la probabilità di errore (riducendo la distanza di Hamming) nel determinare l'intera sequenza (blocco, messaggio, ecc.) ed è, quindi, ottima in questo senso.

Esempi applicativi

Daremo ora degli esempi che mostreranno che il problema suddetto si applica in numerosi e diversi campi, come l'interferenza intersimbolica, la FSK, il riconoscimento di testo e la codifica convoluzionale.

Interferenza intersimbolica

L'interferenza intersimbolica è una forma di distorsione del segnale nella quale un simbolo interferisce con la sotto sequenza di simboli. Si tratta di una specie di disturbo dove un dato trasmesso sul canale ha la forma d'onda le cui code interferiscono con il segnale del dato successivo. Entrando più nel dettaglio, prendiamo in esame le trasmissioni digitali su canali analogici (non ideali), dove si incontra frequentemente la seguente situazione. La sequenza di input u, con valori discreti e a tempo discreto come nel modello dei registri a scorrimento, è usata per modulare (trasmettere un segnale elettrico per mezzo di un altro segnale elettrico con caratteristiche di frequenza e/o fase diverse) una qualche forma d'onda continua che è trasmessa attraverso un canale e successivamente campionata. Idealmente, i campioni z_k dovrebbero essere uguali ai corrispondenti u_k, tuttavia molto spesso non è così. In effetti

i campioni z_k vengono disturbati sia dal rumore presente sul canale sia dagli input vicini $u_{k'}$. Il secondo è chiamato, appunto, interferenza intersimbolica. Alcune volte l'interferenza intersimbolica viene introdotta deliberatamente per sagomare lo spettro del segnale trasmesso in modo da venire incontro alle caratteristiche del canale di trasmissione (tecniche di *spectral shaping*) nei cosiddetti sistemi a risposta parziale. In questi casi l'output campionato può essere spesso modellato come

$$z_k = y_k + n_k$$

dove y_k è una funzione deterministica con un numero finito di input, ad esempio $y_k = f(u_k, \ldots, u_{k-v})$ e n_k è il rumore gaussiano bianco additivo (AWGN) ovvero il rumore termico (equivale ad avere, sul canale, un segnale di fondo incognito). Questo è precisamente il modello con registri a scorrimento riportato in Figura 3.

Per essere più specifici, nella modulazione ad ampiezza di impulso (PAM) la sequenza di segnale y può considerata come una convoluzione della sequenza di input u con una qualche sequenza di risposte impulsive $(h_0, h_1, \ldots)$

$$y_k = \sum_i h_i u_{k-i}.$$

Se $h_i = 0$ per $i > v$ (risposta impulsiva finita), allora si ottiene il modello con registri a scorrimento. La Figura Figura 4 mostra un modello di interferenza intersimbolica che copre tre unità di tempo ($v = 2$).

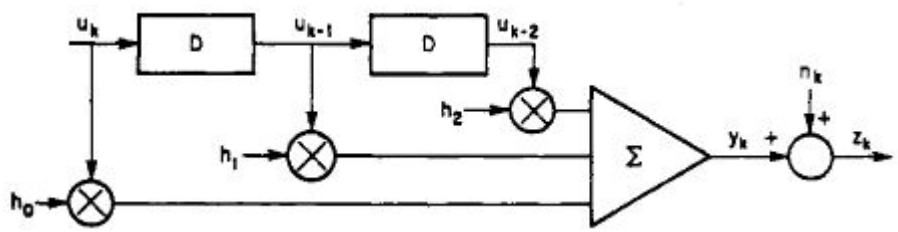

Figura 4: Modello di un sistema PAM soggetto a interferenza intersimbolica e rumore gaussiano bianco.

MSK o Continuous Phase FSK

Questo esempio è citato non per la sua importanza pratica ma perché, primo, tramite esso è possibile definire un semplice modello che useremo come esempio nel seguito della trattazione e, secondo, perché mostra come l'algoritmo

di Viterbi può essere usato nelle situazioni che non sono le più tradizionali. Prima di tutto diamo una breve spiegazione di cosa si intende per FSK. Nella *frequency-shift keying* (FSK) si ottiene una modulazione di frequenza in cui il segnale modulante sposta la frequenza della portante in uscita da uno all'altro di due valori predeterminati, quando la modulazione FSK è binaria (esempio in Figura 5). Si ha quindi che una sequenza digitale u di input se-

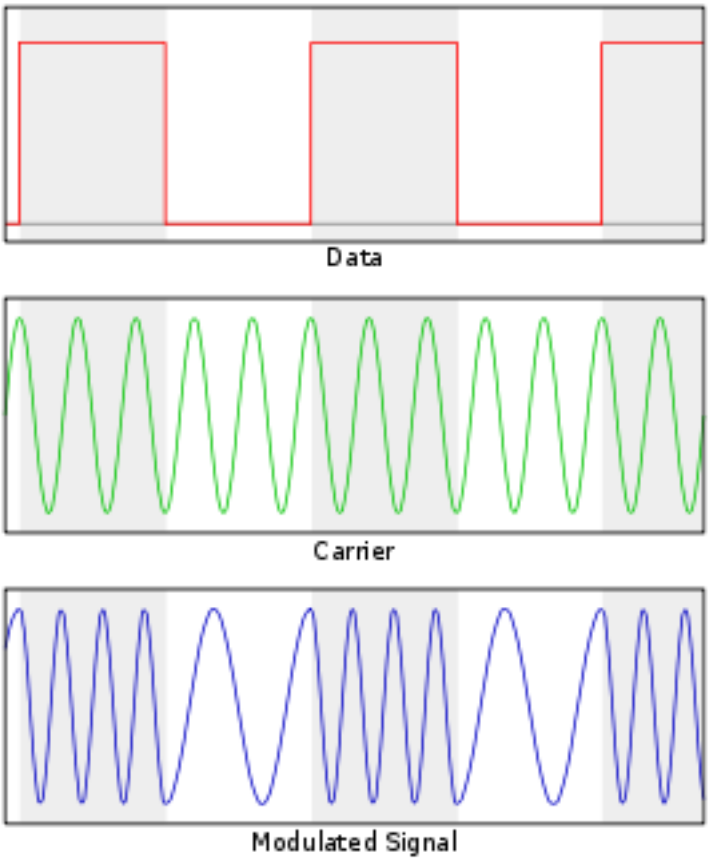

Figura 5: Esempio di modulazione FSK binaria.

leziona una delle m frequenze (se u_k è m-ario) per ogni intervallo di lunghezza T (durata del simbolo); questo significa che il segnale trasmesso $\eta(t)$ è

$$\eta(t) = \cos[\omega(u_k)t + \theta_k], \ kT \leq t < (k+1)T$$

dove $\omega(u_k)$ è la frequenza selezionata da u_k e θ_k è la fase del segnale. Dato che le frequenze selezionate sono scelte a caso non è assicurata la continuità del segnale tra un simbolo ed il successivo. Questo inconveniente causa notevoli

problemi in quanto una brusca variazione del segnale crea componenti ad alta frequenza che disturberebbero altri canali di trasmissione. Per rimediare a questi inconvenienti si fa in modo che le frequenze scelte abbiano una distanza pari a $1/T$. Inoltre, se vi è continuità di fase tra i diversi simboli (cioè nell'intervallo di transizione) come segue

$$\omega(u_{k-1})kT + \theta_{k-1} \equiv \omega(u_k)kT + \theta_k \bmod 2\pi$$

allora si parla di CPM o continuous-phase FSK.

La continuità di fase introduce memoria nel processo di modulazione (per esempio, il segnale attualmente trasmesso nel k-esimo intervallo dipende dal precedente segnale). Esaminiamo il caso più semplice possibile. Sia u una sequenza di input binaria e siano $\omega(0)$ e $\omega(1)$ scelti in modo tale che $\omega(0)$ compia un numero intero di cicli in T secondi e $\omega(1)$ compia un numero semidispari di cicli; ad esempio $\omega(0)T = 0$ e $\omega(1)T = \pi \bmod 2\pi$. Quindi $\theta_0 = 0$, $\theta_1 = 0$ o π, a seconda che u_0 valga 0 o 1 e, similmente, $\theta_k = 0$ o π, a seconda che siano stati trasmessi un numero pari o dispari di 1.

In questa situazione si hanno due stati del processo, con $X = \{0, \pi\}$. Il segnale trasmesso y_k è una funzione di entrambi gli input correnti u_k e x_k:

$$y_k = \cos[\omega(u_k)t + x_k] = \ \cos\, x_k \cos\, \omega(u_k)t,\ kT \leq t < (k+1)T$$

Dato che le transizioni $\xi_k = (x_{k+1}, x_k)$ sono funzioni uno-a-uno dello stato corrente x_k e l'input u_k, è possibile, alternativamente, vedere y_k come determinato da ξ_k.
Se prendiamo $\eta_0(t) \triangleq \cos\, \omega(0)t$ e $\eta_1(t) \triangleq \cos\, \omega(1)t$, come basi del segnale, allora possiamo scrivere

$$y_k = y_{0k}\eta_0(t) + y_{1k}\eta 1(t)$$

Dove le coordinate (y_{ok}, y_{1k}) sono date

$$\begin{cases} (1,0) \text{ se } u_k = 0\ x_k = 0 \\ (-1,0) \text{ se } u_k = 0\ x_k = \pi \\ (0,1) \text{ se } u_k = 1\ x_k = 0 \\ (0,-1) \text{ se } u_k = 1\ x_k = \pi \end{cases}$$

Infine, se il segnale ricevuto $\xi(t)$ è $\eta(t)$ a cui si aggiunge il rumore bianco gaussiano $v(t)$, allora correlando il segnale ricevuto con $\eta_0(t)$ e $\eta_1(t)$ su ogni intervallo, è possibile arrivare all'output discreto nel tempo e senza perdita di informazione

$$z_k = (z_{0k}, z_{1k}) = (y_{0k}, y_{1k}) + (n : 0k, n_{1k})$$

dove n_0 e n_1 sono sequenze indipendenti di rumore bianco gaussiano (Figura 6).

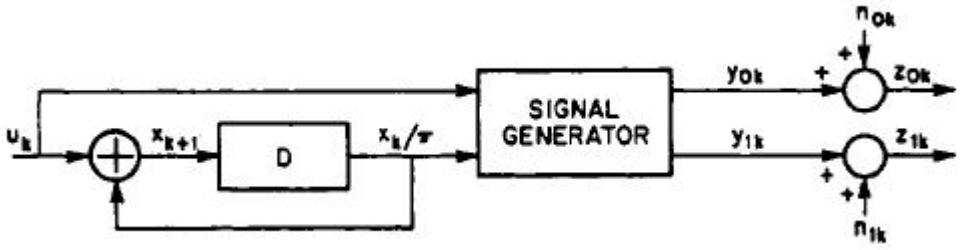

Figura 6: Modello per la continuous-phase FSK binaria

Riconoscimento di testo

Questo esempio è stato incluso per mostrare che l'algoritmo di Viterbi non si limita alle comunicazioni digitali. Nelle letture per il riconoscimento ottico dei caratteri (Optical-Character-Recognition OCR), vengono scannerizzati singoli caratteri, le caratteristiche salienti vengono isolate e vengono prese delle decisioni per capire quale lettera o quale carattere si sta leggendo. Quando il carattere attualmente letto fa parte di un testo in linguaggio naturale, allora è possibile estrarre da quest'ultimo, man mano che viene letto, delle informazioni contestuali che aiutino a risolvere le ambiguità. Un modo per modellare i vincoli contestuali consiste nel trattare un linguaggio naturale, come ad esempio l'italiano, come se fosse un processo discreto di Markov. Per esempio, possiamo supporre che la probabilità di occorrenze di ogni lettera dipenda dalle v precedenti lettere. Così facendo è possibile stimare queste probabilità dalle frequenze delle combinazioni di $(v+1)$ lettere.

Con questo modello, le lettere dell'alfabeto italiano sono viste come l'output di un processo di Markov a m^v stati dove m è il numero di caratteri dell'alfabeto italiano. Se, inoltre, si assume che l'output z_k del processo OCR dipende solamente dal corrispondente carattere di input y_k, allora la lettura OCR può essere vista come un canale privo di memoria alla cui sequenza di output è possibile applicare l'algoritmo di Viterbi per sfruttare i vincoli contestuali. Questo esempio è schematizzato in Figura 7.

Alcuni esperimenti e ulteriori riferimenti si possono trovare in [3] e in [4].

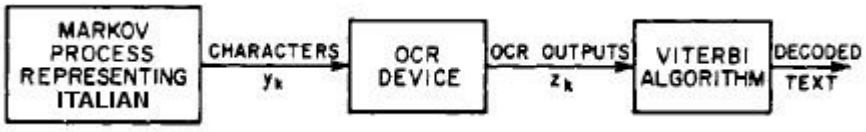

Figura 7: Riconoscimento di testo.

Codifiche Convoluzionali

Un codificatore convoluzionale con rapporto di codifica $1/n$ è un circuito con registri a scorrimento come quello mostrato precedentemente, dove gli input u_k sono bit di informazione e gli output y_k sono blocchi di n bit, $y_k = (p_{1k}, \ldots, p_{nk})$, dove ogni blocco y_k è un bit di parità (mod 2) di un sottoinsieme dei $v+1$ bit di informazione $(u_k, u_{k-1}, \ldots, u_{k-v})$. Quando la sequenza codificata y (parola di codice) è spedita attraverso un canale privo di memoria si ha, precisamente, il modello visto in precedenza (Figura 3). Ma entriamo più nel dettaglio. La codifica convoluzionale può essere considerata come un caso di codifica con memoria in quanto l'influenza di un blocco di bit in ingresso si protrae sulla codifica dei blocchi successivi. Può essere schematizzata nel modo seguente. Il flusso di simboli di ingresso u viene "segmentato", mediante una conversione serie-parallelo, in una successione di parole di ingresso u_k, $k = 0, 1, 2, \ldots$ di lunghezza k (abusiamo della notazione per le parole di ingresso u_k e la lunghezza k delle parole stesse, ovviamente le due cose non sono correlate). Questa sequenza viene trasformata in una sequenza y di parole di codice di lunghezza n mediante una trasformazione

$$y_k = f(u_k, u_{k-1}, \ldots, u_{k-v})$$

con memoria v (misurata in parole di sorgente). Successivamente, la sequenza di parole di codice viene convertita in un flusso di simboli d'uscita mediante una conversione parallelo-serie. Una codifica siffatta viene indicata con la notazione (n, k, v) dove n è la lunghezza delle parole di output e k la lunghezza delle parole di input. Notare che se $v = 0$ si ritrova la codifica a blocchi (senza memoria). Quindi, la differenza sostanziale rispetto ai codici a blocchi sta nel fatto che la trasformazione che dà le parole di uscita y_k è una trasformazione con memoria. Inoltre, nel caso considerato ovvero il caso i cui gli alfabeti di ingresso e di uscita sono coincidenti, il rapporto

$$R = \frac{k}{n}$$

è il rapporto di codifica del codice o tasso di trasmissione. Chiaramente, per garantire la decodificabilità del codice è necessario che sia $R \leq 1$, o anche, in termini di lunghezza delle parole, $k \leq n$. Il codificatore convoluzionale può essere pensato come una macchina a stati in cui

$$x_k = (u_{k-1}, u_{k-2}, \ldots, u_{k-v})$$

è il vettore di stato (tante locazioni quante sono le variabili di stato) e

$$x_{k+1} = \delta(x_k, u_k)$$

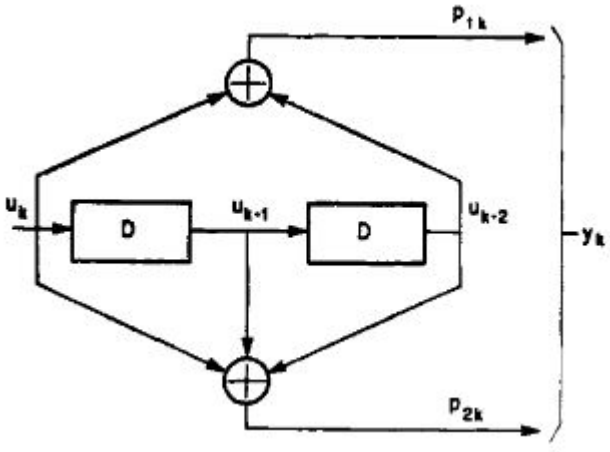

Figura 8: Codificatore convoluzionale con tasso di trasmissione $\frac{1}{2}$.

$$y_k = f(u_k, x_k)$$

sono rispettivamente l'equazione di transizione di stato e quella di uscita.

La macchina a stati può essere quindi rappresentata da un diagramma di stato. In particolare, per il codificatore illustrato in Figura 8, si ha il diagramma della Figura 9.

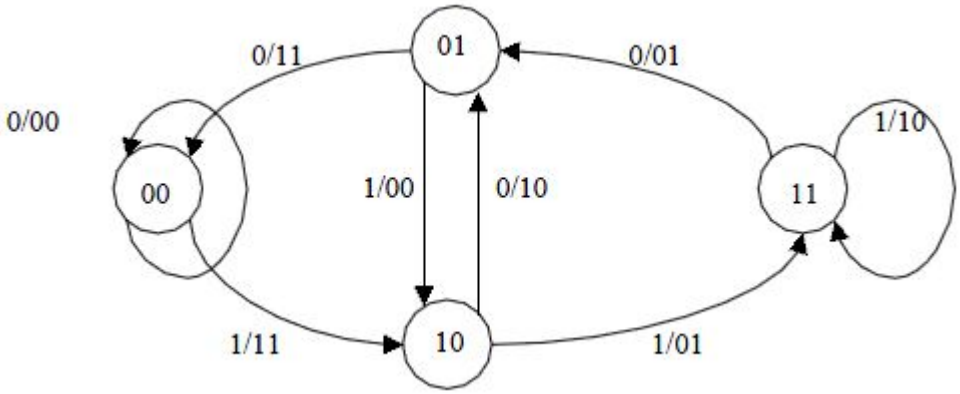

Figura 9: Diagramma di stato relativo al codificatore della Figura 8

Il seguente esempio, invece, mostra come i blocchi di output y_k di due bit siano un controllo di parità (mod 2) dei $v + 1$ bit di ingresso.

Esempio $n = 1, k = 2, v = 2, R = \frac{1}{2}$
input: 1 1 1 0 0 1
output macchina a stati: 11 10 01 10 11 11
se consideriamo i primi $v + 1 = 3$ bit, ovvero 111, di input, l'output corrispondente è 01_{bin} ovvero 1_{dec} mod $2 = 1$, se consideriamo la seconda terna di bit 110 l'output è 10_{bin} ovvero 2_{dec} mod $2 = 0$, per la terna di bit 100 si ha 11_{bin} e cioè 3_{dec} mod $2 = 1$ e così via.

Diagrammi a traliccio

Il problema della decodifica di una sequenza emessa da un codificatore convoluzionale, eventualmente affetta da errori di trasmissione, può essere impostato nel seguente modo. Si consideri il caso in cui una sequenza finita $u = u_0 u_1 \cdots u_{L-1}$ di L parole viene trasformata da un codificatore (n, k, v) (a registri inizialmente azzerati) in una sequenza di parole codificate $y = y_0 y_1 \cdots y_{L+v-1}$ lasciando alla fine i registri del codificatore nuovamente azzerati. Si supponga, inoltre, che la sequenza sia affetta da errori indipendenti secondo il modello di un canale discreto senza memoria. All'uscita del canale si ha una sequenza $z = z_0 z_1 \cdots z_{L+v-1}$ che può non coincidere con la sequenza y delle parole codificate trasmesse. Il problema della decisione relativa alla sequenza trasmessa sulla base della sequenza ricevuta, come già accennato in precedenza, si basa sul criterio della massima verosimiglianza. La soluzione infatti consiste nel scegliere, tra tutte le possibili sequenze di parole codificate, quella $\hat{y}$ che rende massima la probabilità che sia stata ricevuta z. Nel caso di un canale simmetrico, in particolare, si adotta il criterio della minima distanza, considerando la sequenza $\hat{y}$ avente la minima distanza di Hamming dalla sequenza z. Si noti che non tutte le sequenze di parole codificate sono possibili, se si assume che lo stato iniziale e lo stato finale coincidano con quello in cui tutti i registri del codificatore sono azzerati (indicato con S_0). Il problema è pertanto quello di individuare tutte le sequenze di parole codificate ammissibili, calcolare per ciascuna di esse la funzione verosimiglianza $P(y|z)$ rispetto alla parola ricevuta z (o la distanza di Hamming da z) e determinare quale delle sequenze possibili dia luogo alla massima verosimiglianza (o alla minima distanza di Hamming). Ovviamente, il numero delle sequenze possibili cresce in modo esponenziale con la lunghezza L della sequenza di ingresso. Più precisamente, quest'ultimo è pari a q^{kL}, il quale è anche il numero delle possibili sequenze di ingresso. Pertanto la complessità computazionale diventa ben presto insostenibile a meno che non si adottino dei meccanismi che consentano di ridursi ad una complessità di calcolo limitata.

Nel caso della decodifica, per prima cosa è necessario determinare tutte le possibili sequenze di parole codificate di lunghezza $L + v$, con il vincolo che la sequenza degli stati del decodificatore inizi e termini nello stato S_0. La cosa potrebbe essere fatta sulla base del diagramma di transizione. Si tratterebbe, quindi, di individuare su di esso tutti i percorsi di lunghezza $L + v$ che iniziano e terminano nello stato S_0. Per ogni percorso, la sequenza delle parole codificate che etichettano i successivi rami è una delle sequenze codificate possibili. É facile tuttavia rendersi conto che, al crescere di L, il metodo diventa poco pratico, dato che diviene presto impossibile rappresentare sul diagramma i vari percorsi in modo non ambiguo. Per facilitare la comprensione del problema si introduce il diagramma a traliccio del codice (detto anche *trellis*). Da un punto di visto concettuale, si tratta semplicemente di un'espansione bidimensionale del diagramma di transizione lungo un asse dei tempi. Gli stati vengono rappresentati in corrispondenza di ogni istante t e le transizioni sono riportate come archi dallo stato di partenza all'istante t allo stato d'arrivo all'istante successivo $t + 1$.

Esempio Si consideri il codificatore convoluzionale binario $(3, 1, 2)$ avente relazioni ingresso-uscita

$$\begin{aligned} y_{r1} &= u_r + u_{r-1} \\ y_{r2} &= u_r + u_{r-2} \\ y_{r3} &= u_r + u_{r-1} + u_{r-2} \end{aligned}$$

Si tratta del codificatore illustrato in Figura 10, il cui diagramma di transizione è illustrato nella figura stessa. Lo stato del codificatore è dato da $s_r = (u_{r-2}, u_{r-1})$, ovvero dal contenuto del registro di scorrimento del codificatore. Gli stati possibili sono pertanto $S_0 = (0, 0)$, $S_1 = (0, 1)$, $S_2 = (1, 0)$ e $S_3 = (1, 1)$. Se si vuole tracciare il diagramma a traliccio corrispondente ad una sequenza di parole d'ingresso di lunghezza $L = 5$ (ognuna lunga 1 bit in questo esempio), si procede nel seguente modo. Si indica in corrispondenza dell'istante 0 lo stato S_0. Si osserva poi che dallo stato S_0 si può rimanere nello stesso stato, in corrispondenza del simbolo d'ingresso $u_0 = 0$ (con emissione della parola codificata $w_0 = 000$), oppure, in corrispondenza del simbolo d'ingresso $u_0 = 1$, passare allo stato S_1 (con emissione della parola codificata $w_0 = 111$). Le transizioni verso gli stati S_0 ed S_1 sono etichettate con le parole codificate corrispondenti. I simboli di input, invece, non sono riportati dato che, nel caso in cui sia $k = 1$, si può seguire la convenzione che il ramo superiore indichi il simbolo d'ingresso 1 e quello inferiore il simbolo d'ingresso 0. Si ripete lo stesso procedimento per i nuovi stati trovati e, in corrispondenza dell'istante 2, tutti gli stati del codificatore compaiono nel diagramma. Il procedimento, quindi, viene ulteriormente iterato. É immediato

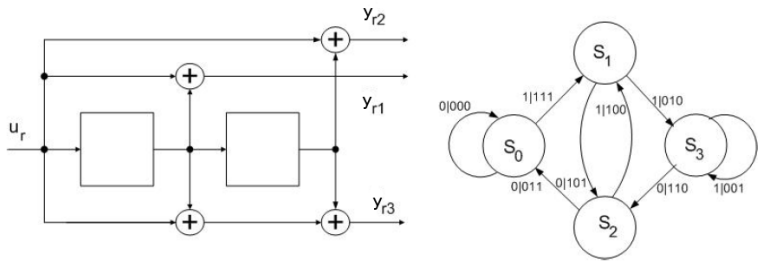

Figura 10: Codificatore convoluzionale binario $(3, 1, 2)$ e relativo diagramma delle transizioni.

che, una volta che sono comparsi tutti gli stati, la struttura del *trellis* rimane inalterata. Tuttavia, una volta giunti alla fine della sequenza d'ingresso, la struttura del traliccio considera soltanto le transizioni che possono portare allo stato finale $S_7 = S_0$, in seguito all'ingresso di simboli nulli. É chiaro che i percorsi dallo stato S_0 allo stato S_0 in $L + v = 7$ passi sono facilmente individuabili e interpretabili sul diagramma a traliccio. Ad esempio, il percorso evidenziato in Figura 11 corrisponde alla sequenza d'ingresso 11001 e alla sequenza codificata

$$111\ 010\ 110\ 011\ 111\ 101\ 011.$$

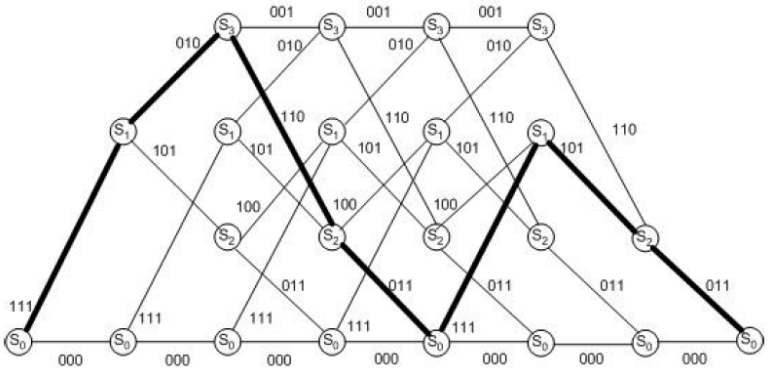

Figura 11: *Trellis* di esempio.

Nel caso generale di un codice q-ario di tipo (n, k, v) con una parola d'ingresso di lunghezza L gli istanti rappresentati nel diagramma a traliccio sono ancora $L+v+1$. Nelle prime v transizioni si passa esponenzialmente dal singolo stato iniziale S_0 ai q^{kV} stati del diagramma di transizione completo. Nelle $L - v$ transizioni successive, fino al completamento della sequenza di

input, il diagramma a traliccio fornisce altrettante repliche del diagramma di transizione. In questa fase da ogni stato escono q^k archi, quanti sono i possibili valori della parola d'ingresso u_r. Nelle ultime v transizioni necessarie all'annullarsi del contenuto dei registri di scorrimento si ritorna allo stato S_0. In questa fase, da ogni stato esce un unico arco corrispondente alla parola d'ingresso $u_r = 0$.

L'algoritmo di Viterbi

Mostreremo ora che il problema della stima a massima probabilità a posteriori, illustrato in precedenza, è formalmente identico al problema di ricerca del cammino minimo su di un certo tipo di grafo. L'algoritmo di Viterbi verrà quindi definito come una naturale soluzione ricorsiva. Assoceremo ad un processo stocastico Markoviano (con stati finiti e a tempo discreto) un diagramma di stato del tipo illustrato in Figura 12(a) per un modello con registri a scorrimento a quattro stati come quello di Figura 8. In particolare,

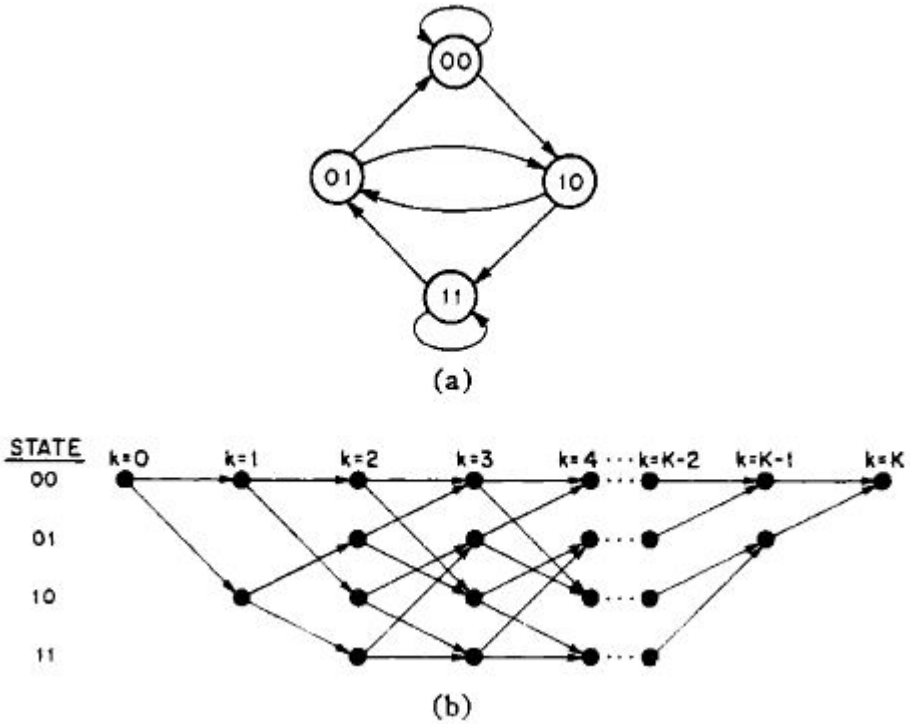

Figura 12: (a) Diagramma di stato di un processo con registri a scorrimento di quattro stati. (b) *Trellis* relativo a tale processo.

i nodi rappresentano gli stati del processo, le ramificazioni rappresentano

transizioni e, durante il corso del tempo, il processo traccia un certo cammino da uno stato all'altro nel diagramma. In Figura 12(b) è rappresentato lo stesso processo markoviano, utilizzando un diagramma a traliccio. In esso, ogni nodo corrisponde ad uno stato distinto in un dato istante ed ogni arco rappresenta una transizione verso qualche nuovo stato all'istante successivo. Il *trellis* inizia e finisce negli stati noti x_0 e x_K. La sua proprietà più importante è che ad una possibile sequenza di stati x corrisponde un unico cammino attraverso il grafo, e viceversa. Ora mostreremo come, data una sequenza osservata z, ad ogni cammino può essere associata una "lunghezza" proporzionale a $-\ln P(x,z)$, dove x è la sequenza di stati associata a quel determinato cammino. Questo consente di risolvere il problema di ricerca della sequenza di stati per cui $P(x|z)$ è massima, o equivalentemente per la quale $P(x,z) = P(x|z)P(z)$ è massima, trovando il cammino la cui lunghezza $-\ln P(x,z)$ è minima (dato che $\ln P(x,z)$ è una funzione monotona su $P(x,z)$ e c'è una corrispondenza uno-a-uno attraverso i cammini e le sequenze). É sufficiente osservare che, grazie alle proprietà di Markov e di "assenza di memoria", $P(x,z)$ è fattorizzabile nel seguente modo:

$$\begin{aligned} P(x,z) &= P(x)P(z|x) \\ &= \prod_{k=0}^{K-1} P(x_{k+1}|x_k) \prod_{k=0}^{K-1} P(z_k|x_{k+1}, x_k). \end{aligned}$$

Quindi, se assegniamo ad ogni arco (transizione) la lunghezza

$$\lambda(\xi_k) \triangleq -\ln P(x_{k+1}|x_k) - \ln P(z_k|\xi_k),$$

allora la lunghezza totale del cammino corrispondente a qualche x è

$$-\ln P(x,z) = \sum_{k=0}^{K-1} \lambda(\xi_k).$$

Nel campo della ricerca algoritmica, trovare il cammino più corto in un grafo è un problema ben noto. Nel 1957 Minty propose una soluzione molto concisa ma, sfortunatamente, tale algoritmo non si adatta molto bene ai metodi computazionali moderni (si noti che l'algoritmo di Dijkstra è stato pubblicato due anni più tardi). Proprio per questo, l'algoritmo di Viterbi è una valida alternativa. Prima di illustrarlo, però, è necessaria un'ulteriore osservazione. Denotiamo con x_0^k un segmento $(x_0, x_1, \cdots, x_k)$ che consiste degli stati fino all'istante k della sequenza di stati $x = (x_0, x_1, \cdots, x_k)$. Rispettivamente, nel *trellis*, x_0^k corrisponde ad un segmento di un cammino che inizia nel nodo x_0 e termina in x_k. Per ogni nodo x_k, corrispondente ad un certo istante k, in generale, ci sono diversi segmenti di cammino, ciascuno con una certa lunghezza

$$\lambda(x_0^k) = \sum_{i=0}^{k-1} \lambda(\xi_i).$$

Chiamiamo il segmento che corrisponde al cammino minimo come il segmento *sopravvissuto* corrispondente al nodo x_k e lo denotiamo con $\hat{x}(x_k)$. Per ogni istante $k > 0$, ci sono M sopravvissuti, uno per ogni x_k. L'osservazione è la seguente: il cammino minimo completo $\hat{x}$ deve iniziare con uno di questi sopravvissuti. Se per assurdo non fosse così, ma passasse per lo stato x_k all'istante k, allora potremmo rimpiazzare il suo segmento iniziale con $\hat{x}(x_k)$ per ottenere un cammino più corto e ciò porterebbe ad una contraddizione. Quindi, per ogni istante k, è sufficiente memorizzare solo gli M sopravvissuti $\hat{x}(x_k)$ e le loro lunghezze $\Gamma(x_k) \triangleq \lambda[\hat{x}(x_k)]$. Per passare all'istante successivo $(k+1)$, è necessario calcolare le lunghezze dei segmenti estesi e, per ogni nodo x_{k+1}, selezionare il segmento il cui cammino è minimo e termina in x_{k+1}. Quest'ultimo diventerà il sopravvissuto dell'istante temporale $k+1$. La ricorsione procede indefinitivamente senza che il numero di sopravvissuti ecceda M.

Per chiarirne meglio il funzionamento, illustriamo l'algoritmo nel caso di un semplice *trellis* con quattro stati e che copre 5 unità di tempo (Figura 13).

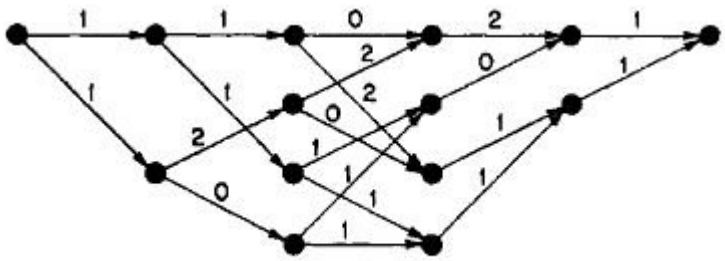

Figura 13: *Trellis* con archi etichettati dalle lunghezze, dove $M = 4$ e $K = 5$.

La Figura 13 mostra il diagramma al completo, dove ad ogni arco è associata una lunghezza (in un'applicazione reale, le lunghezze sarebbero funzione dei dati ricevuti). La Figura 14, invece, mostra i 5 passi di ricorsione, attraverso i quali l'algoritmo determina il cammino minimo dal nodo iniziale a quello finale. In ogni passo sono illustrati al più 4 sopravvissuti e le rispettive lunghezze.

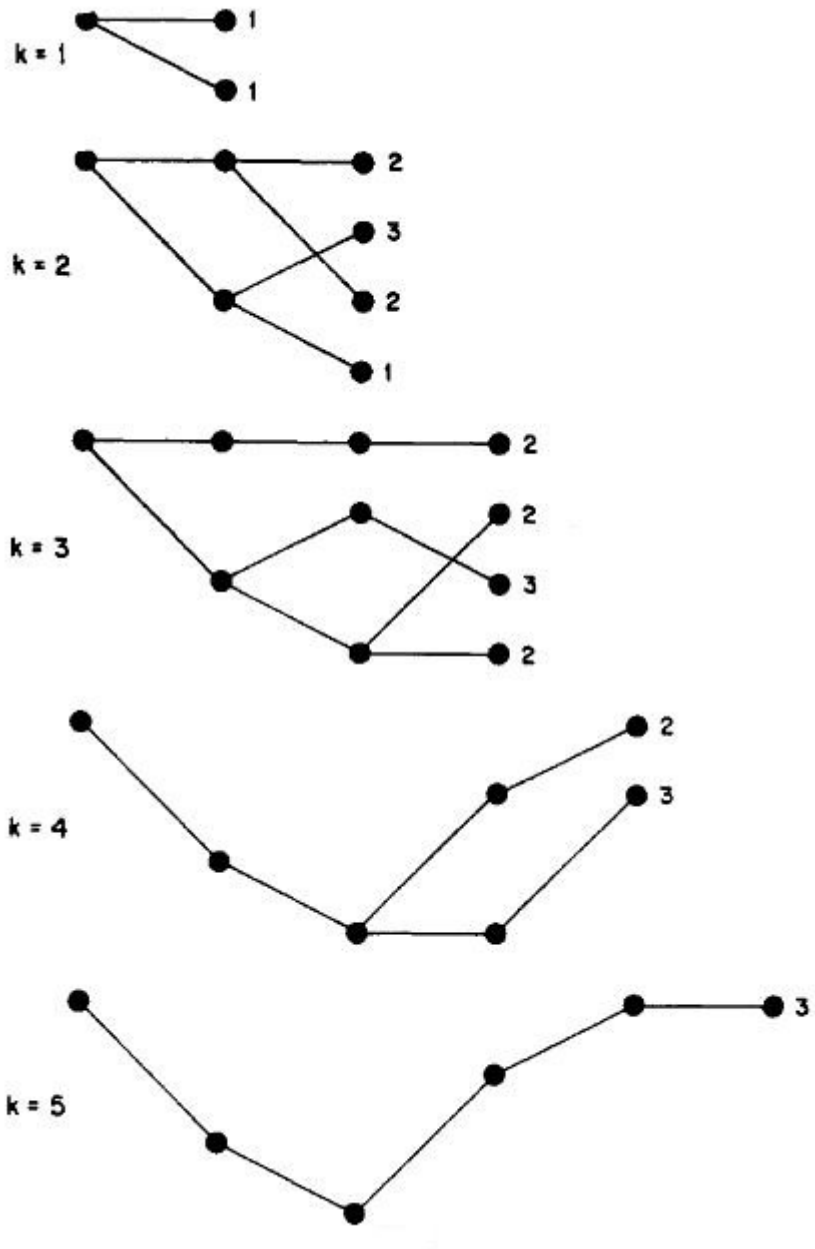

Figura 14: Determinazione ricorsiva del cammino minimo attraverso l'algoritmo di Viterbi.

Formalmente, l'algoritmo di Viterbi può essere definito come segue:

- **Memorizzazione:**

k	(indice temporale)
$\hat{x}(x_k),\ 1 \leq x_k \leq M$	(sopravvissuto terminante in x_k);
$\Gamma(x_k),\ 1 \leq x_k \leq M$	lunghezza del sopravvissuto

- **Inizializzazione:**

$$\begin{array}{ll} k = 0; & \\ \hat{x}(x_0) = x_0; & \hat{x}(m) \text{ arbitrario, per } m \neq x_0; \\ \Gamma(x_0) = 0; & \Gamma(m) = \infty, \text{ per } m \neq x_0. \end{array}$$

- **Ricorsione:** Calcola

$$\Gamma(x_{k+1}, x_k) \triangleq \Gamma(x_k) + \lambda\left[\xi_k = (x_{k+1}, x_k)\right]$$

per ogni $\xi_k = (x_{k+1}, x_k)$. Trova

$$\Gamma(x_{k+1}) = \min_{x_k} \Gamma(x_{k+1}, x_k)$$

per ogni x_{k+1}; memorizza $\Gamma(x_{k+1})$ ed il corrispondente sopravvissuto $\hat{x}(x_{k+1})$. Incrementa k e ripeti fino a quando $k = K$.

Con una sequenza finita x di stati, l'algoritmo termina all'istante K, nel quale, il cammino minimo è stato memorizzato in $\hat{x}(x_K)$.

Nella pratica, però, sono necessarie alcune semplici modifiche. Quando la sequenza degli stati è molto lunga o infinita, è necessario "troncare" i sopravvissuti ad una certa lunghezza δ. In altre parole, l'algoritmo, all'istante k, deve prendere una certa decisione sui nodi fino all'istante $k - \delta$. Si noti che in Figura 14 tutti i sopravvissuti all'istante 4 attraversano gli stessi nodi fino all'istante 2. In generale, se un profondità di troncamento δ è stata scelta sufficientemente grande, c'è un'alta probabilità che tutti i sopravvissuti all'istante k attraversino gli stessi stati fino all'istante $k - \delta$. In questo modo il segmento iniziale del cammino a massima verosimiglianza è noto fino all'istante $k - \delta$ e può essere restituito dall'algoritmo; in questo caso il troncamento non costa nulla. Nei rari casi in cui i sopravvissuti non concordano, va bene una qualunque strategia ragionevole: scegliere un nodo arbitrario all'istante $k - \delta$, o il nodo associato al sopravvissuto più corto, ecc.. Se δ è sufficientemente grande, l'effetto sulle prestazioni è trascurabile.

Inoltre, se k diventa grande, è necessario normalizzare le lunghezze $\Gamma(m)$ da un istante all'altro sottraendo una costante da esse.

Infine, lo stato iniziale x_0 potrebbe non essere noto. In questo caso, si può ragionevolmente assegnare, per ogni m, $\Gamma(m) = 0$, oppure $\Gamma(m) = -\ln \pi_m$

se gli stati hanno a priori probabilità π_m. Solitamente, dopo un momentaneo periodo iniziale, c'è un'alta probabilità che tutti i sopravvissuti si riuniscano nel cammino corretto.

La complessità dell'algoritmo si può stimare facilmente. Innanzitutto la memoria richiesta è di M locazioni (una per ogni stato), dove ognuna di queste è in grado di memorizzare una lunghezza $\Gamma(m)$ ed una lista troncata di sopravvissuti $\hat{x}(m)$ di δ simboli. Per quanto riguarda la complessità della computazione, in ogni unità di tempo l'algoritmo deve calcolare $|A|$ addizioni (una per ogni transizione) ed M confronti fra i $|A|$ risultati. Quindi la memoria richiesta è proporzionale al numero degli stati ed il numero di passi è proporzionale al numero di transizioni. Con un processo a registri a scorrimento, $M = m^v$ e $|A| = m^{v+1}$. Quindi la complessità cresce esponenzialmente con la lunghezza v dello *shift-register*.

Nel precedente paragrafo, è stata tralasciata la complessità riguardante la generazione incrementale delle lunghezze $\lambda(\xi_k)$. In un processo di registri a scorrimento, tipicamente $P(x_{k+1}|x_k)$ è $1/m$ oppure 0, in base al fatto che x_{k+1} sia un successore ammissibile di x_k o meno. Quindi tutte le transizioni ammissibili hanno lo stesso valore $-\ln P(x_{k+1}|x_k)$ e questa componente di $\lambda(\xi_k)$ può essere ignorata. Si noti che, in casi più generali, $P(x_{k+1}|x_k)$ è nota a priori e quindi questa componente può essere precalcolata e "cablata" all'interno dell'algoritmo. La componente $-\ln P(z_k|\xi_k)$ è l'unica che dipende dai dati; è tipico che diversi ξ_k portino allo stesso *output* y_k e, quindi, dato z_k, il valore $-\ln P(z_k|y_k)$ può essere calcolato o ricavato dai ξ_k solo una volta. Quando il disturbo è gaussiano, $-\ln P(z_k|y_k)$ è semplicemente proporzionale a $(z_k - y_k)^2$. Infine, tutto questo può esser fatto all'esterno della ricorsione centrale. Quindi la complessità di tale computazione non è significante.

Per di più, si osservi che, una volta calcolati i $\lambda(\xi_k)$, non è più necessario tenere in memoria le osservazioni z_k (buona caratteristica per le applicazioni *real-time*.

Un'osservazione più attenta dei *trellis* di un processo a registri a scorrimento rivela ulteriori dettagli che possono essere sfruttati nell'implementazione. Per un *shift register* binario, le transizioni di un certo istante possono essere suddivise in 2^{v-1} gruppi di quattro, ognuno dei quali ha origine in una coppia comune di stati e termina in un'altra coppia comune. Una tipica cella di questo tipo è rappresentata in Figura 15, dove gli stati dell'istante k sono etichettati $x'0$ e $x'1$ e quelli dell'istante $k+1$ sono etichettati con $0x'$ e $1x'$. Inoltre, x' è una sequenza di $v-1$ bit che è costante all'interno della cella. Per esempio, ogni istante nel *trellis* di Figura 13, è costituito da due di queste celle.

Si noti come il *trellis* di Figura 12(b) ricordi il diagramma delle computazioni della trasformata rapida di Fourier (FFT). Infatti, è identico, ec-

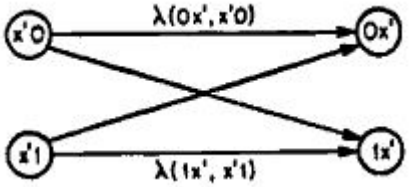

Figura 15: Cella per un trellis di un processo su un registro a scorrimento.

cetto per le lunghezze. Data la sua struttura fortemente parallela ed il solo uso di operazioni di somma, confronto e selezione, l'algoritmo di Viterbi è adatto per applicazioni ad alta velocità.

Analisi delle prestazioni

L'algoritmo di Viterbi non è solo importante per la semplicità dell'implementazione, ma anche per la semplicità con cui le prestazioni possono essere analizzate. In molti casi, inoltre, è possibile derivare dei limiti sia inferiori che superiori (stretti) ben fissati per la probabilità di errore.

Il concetto chiave nell'analisi delle performance è quello di *evento di errore*. Sia x l'attuale stato della sequenza (cammino reale) e $\widehat{x}$ lo stato della sequenza effettivamente preso dall'algoritmo. Nel corso di un lungo periodo di tempo, x e $\widehat{x}$ solitamente divergeranno e riemergeranno un certo numero di volte come illustrato in Figura 16. Ogni distinta separazione di $\widehat{x}$ da

Figura 16: Tipico caso di cammino corretto x (linea spessa) e cammino stimato $\hat{x}$ (linea sottile) nel trellis, mostra gli eventi di errore nell'albero.

x è definita come "evento di errore". Gli eventi di errore possono essere, in genere, di lunghezza illimitata se x è un cammino infinito. Tuttavia, la probabilità di un errore infinito è, di solito, zero.

L'importanza degli eventi di errore è che essi sono, fra di loro, probabilisticamente indipendenti; nel linguaggio della teoria della probabilità si dice che essi sono *ricorrenti* (analisi di sopravvivenza). Inoltre, essi, ci permettono di calcolare la probabilità di errore per unità di tempo la quale si rende necessaria fintanto che la probabilità di un *qualsiasi* errore in una stima di

massima probabilità a posteriori per un blocco di lunghezza K tende a 1 quando K va all'infinito.

Possiamo, quindi, calcolare la probabilità di un evento di errore che inizia in un dato istante di tempo, ammettendo che lo stato della sequenza corrispondente risulti corretto (l'evento di errore non è in corso in quell'istante).

Dato il cammino corretto x, l'insieme E_k di tutti i possibili eventi di errori che iniziano in un certo istante k è simile ad un trellis che inizia in x_k in cui ognuno dei sui rami finisce nel cammino corretto, come illustrato in Figura 17, per il *trellis* di Figura 12(b). Nella teoria dell'informazione, questo è definito *sottoinsieme non corretto* (al tempo k). La probabilità di ogni

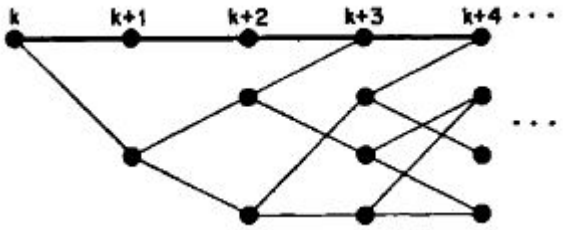

Figura 17: Cammino corretto x (linea spessa e sottoinsieme non corretto al tempo k per il trellis di Figura 12(b).

specifico evento di errore si può calcolare facilmente; essa è semplicemente la probabilità che le osservazioni siano, durante il periodo di tempo in cui $\widehat{x}$ è diversa da x, più simile a $\widehat{x}$ che a x. Se l'evento di errore ha lunghezza τ, quanto detto si traduce in un problema di decisione di due ipotesi fra due sequenze lunghe τ e, tipicamente, ha una soluzione standard.

La probabilità $P(E_k)$ che un *qualsiasi* evento di errore in E_k occorra, può, dunque, essere limitata superiormente da un unione di limiti superiori (ad esempio mediante la somma delle probabilità di tutti gli eventi di errore in E_k). Anche se questa somma potrebbe essere teoricamente infinita, essa è solitamente limitata da uno o più termini che rappresentano gli eventi di errore più probabili, la cui somma è, appunto, una buona approsimazione per $P(E_k)$.

D'altra parte, un limite inferiore alla probabilità degli eventi di errore può essere ottenuta come segue. Prendiamo il particolare evento di errore che ha la più grande probabilità fra tutti quelli in E_k. Supponiamo di avere a disposizione un oracolo che ci dica che la sequenza di stati corretta (reale) è una delle due seguenti (entrambi possibili):

1. l'attuale cammino corretto,

2. il cammino non corretto corrispondente al suddetto evento di errore.

Nonostante questa informazione, continueremmo ugualmente a commettere un errore quando il cammino errato è più simile alla data sequenza di osservazioni z. La probabilità di errore continua a non essere migliore della probabilità del particolare evento di errore. In assenza dell'oracolo, la probabilità di errore deve essere peggiore perchè, una delle possibilità, data l'informazione dell'oracolo, è quella di ignorarla. In altra parole, la probabilità di un particolare evento è un limite inferiore per $P(E_k)$.

Concludendo, la probabilità di ogni evento di errore che inizia al tempo k può essere limitato superiormente ed inferiormente come segue:

$$\max P(\text{evento di errore}) \leq P(E_k) \leq \max P(\text{evento di errore}) + \text{altri termini}.$$

Esempio Prenderemo in esame il calcolo della Continuous Phase FSK presentato in precedenza. Il trellis a due stati del processo è mostrato in Figura 18 mentre la prima parte di un tipo sottoinsieme non corretto è mostrata in Figura 19.

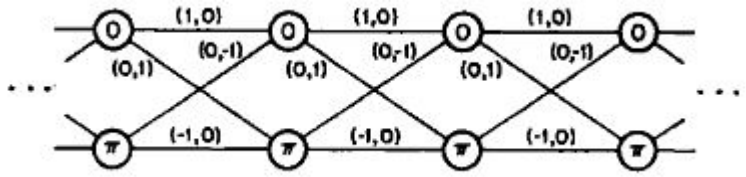

Figura 18: Trellis per la *continuous-phase FSK*.

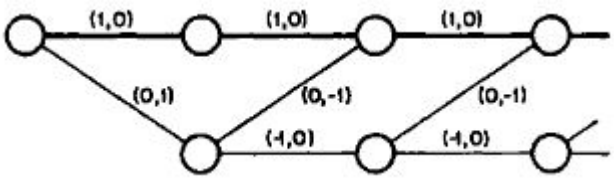

Figura 19: Tipico sottoinsieme non corretto. Linea spessa: cammino corretto. Linee sottili: sottoinsieme incorretto.

L'evento di errore più corto possibile ha lunghezza 2 e consiste nella decisione che il segnale sia $\{\cos\omega(1)t, -\cos\omega(1)t\}$ anziché $\{\cos\omega(0)t, \cos\omega(0)t\}$, o meglio, nella nostra notazione a coordinate, che $\{(0,1),(0,-1)\}$ sia scelto al posto di $\{(1,0),(1,0)\}$. Questo è un problema di decisione a 2 ipotesi in uno spazio del segnale 4-dimensionale. Nel disturbo gaussiano con varianza σ^2 per dimensione, solamente la distanza Euclidea d tra due segnali è rilevante; in questo caso $d = \sqrt{2}$ e, perciò, la probabilità di errore di questo particolare

evento di errore è $Q(d/2\sigma) = Q(1/\sigma\sqrt{2})$, dove $Q(x)$ è la funzione Gaussiana della probabilità di errore definita da

$$Q(x) \triangleq \int_x^\infty \frac{1}{\sqrt{2\pi}} e^{-y^2/dy^2}.$$

Esaminando la Figura 19 notiamo che gli eventi di errore di lunghezza $3, 4, \ldots$ si trovano alle distanze $\sqrt{6}, \sqrt{10}, \ldots$ dal cammino corretto; in questo modo possiamo determinare un limite superiore ed inferiore di $P(E_k)$ di

$$Q(\sqrt{2}/2\sigma) \leq P(E_k) \leq Q(\sqrt{2}/2\sigma) + Q(\sqrt{6}/2\sigma) + Q(\sqrt{10}/2\sigma) + \ldots$$

In vista di un rapido decremento di $Q(x)$ con x, questo implica che $P(E_k$ è stimato come $Q(\sqrt{2}/2\sigma)$ per ogni varianza di disturbo (ragionevole) σ^2. E' da notare come questo risultato sia indipendente dal cammino corretto x o dal tempo k.

Applicazioni

Concludiamo con un riassunto dei risultati, sia teorici che pratici, ottenuti dall'analisi dell'algoritmo di Viterbi.

L'algoritmo di Viterbi è stato originariamente sviluppato per i codici convoluzionali e, naturalmente, è propio in questo caso che si ha il maggior rendimento.

I risultati teorici principali sono contenuti nell'articolo originale di Viterbi [1]. Esso mostra che è possible ridurre esponenzialmente la probabilità di errore con il vincolo di lunghezza v a tutti i tassi di trasmissione R minori della capacità del canale. Per codici convoluzionali binari di lunghezza non asintotica su canali senza memoria (e simmetrici), il principale risultato è che $P(E_k)$ è approssimativamente data da

$$P(E_k) \approx N_d 2^{-dD}$$

dove d è la distanza libera ovvero la minima distanza di Hamming di qualsiasi cammino nel sottoinsieme non corretto E_k dal cammino corretto, N_d è il numero dei suddetti cammini e D è data da

$$D = \log_2 \sum_Z P(z|0)^{1/2} P(z|1)^{1/2}$$

dove la somma è effettuata su tutti gli output z in Z.

In un canale gaussiano (canale in cui è presente del disturbo gaussiano)

$$P(E_k) \approx N_d \exp(-dRE_b/N_0)$$

dove RE_b/N_0 il tasso segnale-disturbo per bit di informazione. Questo è considerato un limite molto basso e, tale limite, viene confermato anche dalle simulazioni.

Inoltre, il tasso di decrescita è considerevolmente più veloce che per i codici a blocchi con complessità di decodifica comparabile.

Se definiamo con *guadagno del codice* il risparmio in rapporto segnale rumore per bit per ottenere le stesse prestazioni di un sistema non codificato (stessa probabilità di errore), al diminuire della probabilità d'errore, il guadagno aumenta:

$$\text{Guadagno codice} \leq 10 \log_{10}(Rd_f)$$

I canali disponibili per le comunicazioni spaziali sono frequentemente modellati tramite canali con rumore gaussiano bianco. L'algoritmo di Viterbi è adatto per questi canali propio perchè egli offre performance superiori a tutti gli altri schemi di codifica, è relativamente veloce, ha una struttura poco complessa e è considerato robusto rispetto ai vari parametri del canale.

I decodificatori di Viterbi vengono, inoltre, usati come elementi in schemi di codifica concatenati al fine di ottenere alte prestazioni, sono stati usati come decodificatori nei codici convoluzionali in presenza di interferenza intersimboli e, soprattutto, sono usati come decodificatori per la tecnologia CDMA [5].

Approcci alternativi al problema

Riporteremo ora alcune strutture che sono legate all'algoritmo di Viterbi, in particolare gli algoritmi di decodifica sequenziale e la minima probabilità di errore per bit. Quando il diagramma a traliccio diventa grande, è naturale abbandonare la ricerca esaustiva dell'algoritmo di Viterbi a favore di una ricerca sequenziale di tipo *trial-and-error*, la quale esamina solo quei cammini che è più probabile che siano i più corti. Questi algoritmi sono noti anche come *decodifica sequenziale*. Uno dei più semplici è senz'altro l'algoritmo "a pila", nel quale viene mantenuta una lista dei cammini minimi parziali trovati, il cammino in cima alla pila viene esteso e tutti i suoi successori riordinati nella lista fintanto che non si trova un cammino che raggiunge il nodo terminale, o che decresce illimitatamente. Molte proprietà della decodifica sequenziale sono le stesse della decodifica di Viterbi e condividono, inoltre, la stessa probabilità di errore. Ciò consente di decodificare codici più lunghi e, quindi, più potenti grazie ad una quantità variabile di memoria richiesta per memorizzare i dati in ingresso z.

Riguardo al fenomeno dell'interferenza intersimbolica, nei primi tentativi per trovare algoritmi non lineari ottimi fu usata la probabilità di errore per bit come criterio di ottimalità. Le proprietà markoviane del processo portano ad algoritmi che sono più trattabili ma meno attrattivi dell'algoritmo di Viterbi. Il principio generale usato è il seguente. Per prima cosa, si calcola la probabilità congiunta $P(x_k, z)$ per ogni stato x_k del *trellis*, o in alternativa $P(\xi_k, z)$ per ogni transizione. Questo deriva dalla seguente osservazione:

$$\begin{aligned} P(x_k, z) &= P(x_k, z_0^{k-1}) P(z_k^K | x_k, z_0^{k-1}) \\ &= P(x_k, z_0^{k-1}) P(z_k^K | x_k) \end{aligned}$$

visto che, dato x_k, gli output z_k^K dall'istante k all'istante K sono indipendenti dagli output z_0^{k-1} dall'istante 0 all'istante $k-1$. In modo simile,

$$\begin{aligned} P(\xi_k, z) &= P(x_k, x_{k+1}, z) \\ &= P(x_k, z_0^{k-1}) P(x_{k+1}, z_k | x_k, z_0^{k-1}) P(z_{k+1}^K | x_{k+1}, x_k, z_0^k) \\ &= P(x_k, z_0^{k-1}) P(x_{k+1}, z_k | x_k) P(z_{k+1}^K | x_{k+1}). \end{aligned}$$

Si giunge quindi alla formula ricorsiva

$$\begin{aligned} P(x_k, z_0^{k-1}) &= \sum_{x_{k-1}} P(x_k, x_{k-1}, z_0^{k-1}) \\ &= \sum_{x_{k-1}} P(x_{k-1}, z_0^{k-2}) P(x_k, z_{k-1} | x_{k-1}) \end{aligned}$$

che consente di calcolare le M quantità $P(x_k, z_0^{k-1})$ dalle M quantità $P(x_{k-1}, z_0^{k-2})$ con $|A|$ moltiplicazioni ed addizioni usando le lunghezze

$$e^{-\lambda(\xi_{k-1})} = P(x_k | x_{k-1}) P(z_{k-1} | \xi_{k-1}).$$

Analogamente, si ha la ricorsione all'indietro

$$\begin{aligned} P(z_k^K | x_k) &= \sum_{x_{k+1}} P(z_k^K, x_{k+1} | x_k) \\ &= \sum_{x_{k+1}} P(z_k, x_{k+1} | x_k) P(z_{k+1}^K | x_{k+1}) \end{aligned}$$

con simile complessità.

Si consideri ora un processo a registri a scorrimento e sia $S(u_k)$ l'insieme degli stati x_{k+1} la cui prima componente è u_k. Allora,

$$P(u_k, z) = \sum_{x_{k+1} \in S(u_k)} P(x_{k+1}, z).$$

Dato che $P(u_k, z) = P(u_k | z) P(z)$, la stima di u_k con massima probabilità a posteriori si riduce alla ricerca del massimo di tale quantità. Analogamente, se si vuole calcolare la stima MAP di un uscita y_k, allora si calcola

$$P(y_k, z) = \sum_{\xi_k \in S(y_k)} P(\xi_k, z),$$

dove $S(y_k)$ è l'insieme di tutti i ξ_k che portano a y_k. Questo algoritmo, però, è meno attraente di quello di Viterbi in quanto necessita di ricorsioni (dirette e all'indietro) e la conseguente memorizzazione di tutti i dati. Nonostante sia possibile eliminare le ricorsioni all'indietro attraverso ulteriori osservazioni, il risultato è un algoritmo sub-ottimale che presenta un significativo costo computazionale aggiuntivo [2].

In conclusione, possiamo considerare gli output aumentati dell'algoritmo di Viterbi. Un buon indicatore del comportamento dell'algoritmo è la profondità con cui i cammini vengono uniti; questo può essere usato per stabilire o meno se un canale di comunicazione è attivo, in sincronia, ecc.. Un indicatore utile per verificare l'affidabilità di un particolare segmento può essere ottenuto dalla differenza delle lunghezze del cammino migliore e di quello più prossimo ad esso nel momento in cui vengono uniti. Infine, l'algoritmo può essere modificato per memorizzare L cammini migliori per generare quindi una lista degli L cammini più probabili.

Bibliografia

[1] A. J. Viterbi, *Error bounds for convolutional codes and an asymptotically optimum decoding algorithm*, IEEE Trans. Inform. Theory, vol. IT-13, pp. 260-269, Apr. 1967.

[2] K. Abend and B. D. Fritchman, *Statistical detection for communication channels with intersymbol interference*, Proc. IEEE, vol. 58, pp. 779-785, May 1970.

[3] D. L. Neuhoff, *The Viterbi algorithm as an aid in text recognition*, Stanford Electronic Labs., Stanford, Calif., unpublished.

[4] J. Raviv, *Decision making in Markov chains applied to the problem of pattern recognition*, IEEE Trans. Inform. Theory, vol. IT-13, pp. 536-551, Oct. 1967.

[5] Kamil Sh. Zingangirov, *Theory of Code Division Multiple Access Communication*, IEEE Press, Wiley-Interscience.