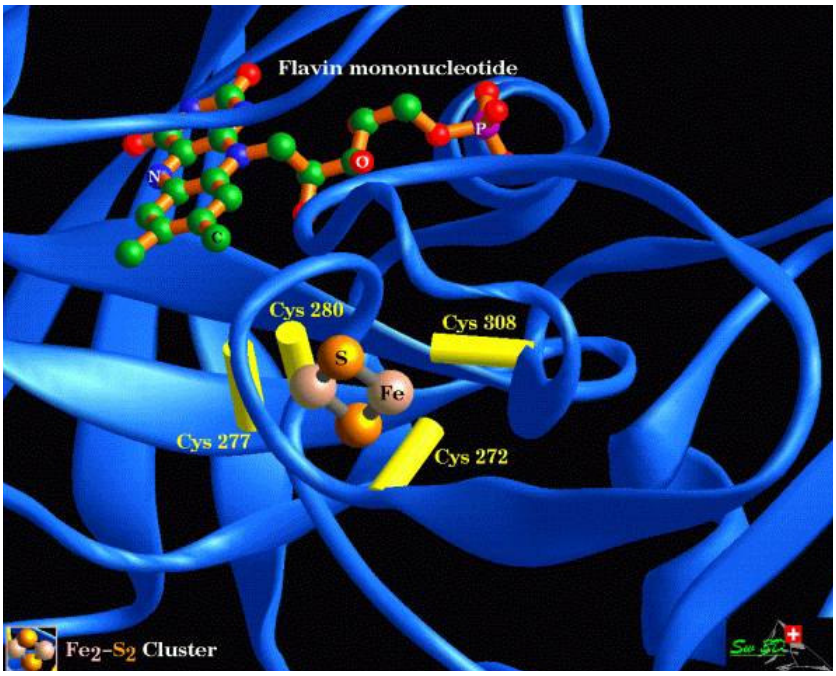
UNIVERSITA' DEGLI STUDI DI FIRENZE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

Elaborato per l'esame di
"Intelligenza Artificiale" A.A. 1998-99
Prof. G. Soda

Andrea Fedeli Sauro Menchetti

Allineamento tramite Modelli di Markov Nascosti



Sommario

Introduzione

L'analisi computazionale di sequenze biologiche (descrizioni di molecole di proteine, di DNA e di RNA) ha completamente cambiato le sue caratteristiche alla fine degli anni '80. La principale causa di questi cambiamenti è stata l'avvento di nuove, efficienti tecniche sperimentali, come il sequenziamento del DNA, che hanno portato ad una crescita esponenziale dei dati a disposizione. L'interesse si sta progressivamente spostando dall'accumulo dei dati alla loro interpretazione, dato che anche altri progetti di sequenziamento come quello del genoma stanno andando avanti velocemente. Strumenti computazionali per la classificazione di sequenze, per separare le regioni del DNA che codificano proteine da quelle che non le codificano, per predire la struttura molecolare, per ricostruire la storia dell'evoluzione e per individuare deboli similarità tra le sequenze, sono divenuti una componente essenziale del processo di ricerca. La bioinfirmatica sta emergendo come una disciplina alla frontiera tra la biologia e l'informatica ed ha ripercussioni in medicina, nelle biotecnologie e nella società in vari modi. Le grandi basi di dati di informazioni biologiche hanno tratto giovamento dai comuni algoritmi, ma tale approccio non è più in grado di indirizzare molti dei più interessanti problemi di analisi delle sequenze. Questo è dovuto all'inerente complessità dei sistemi biologici ed alle lacune delle teorie finora sviluppate sull'organizzazione delle vita a livello molecolare. L'approccio dell'apprendimento automatico (reti neurali, modelli di Markov nascosti, reti bayesiane) è adatto in tutti quei domini caratterizzati dalla presenza di grandi quantità di dati anche rumorosi e dall'assenza di teorie generali. L'idea fondamentale di questo approccio è di *imparare la teoria automaticamente dai dati*, attraverso processi di inferenza, di adattamento di modelli e di apprendimento da esempi; questo rappresenta un ap-

proccio percorribile complementare ai metodi convenzionali. I metodi di apprendimento automatico sfruttano pesantemente la potenza dei calcolatori e traggono grandi benefici dal progresso in velocità delle macchine. È stato osservato che sia la velocità dei computer, sia il volume delle sequenze biologiche si sono incrementate della stessa quantità dagli anni '80, raddoppiando circa ogni 15-18 mesi. Dal punto di vista teorico, l'ambiente di lavoro probabilistico bayesiano unifica tutti i metodi di apprendimento automatico e la confluenza di tre fattori - dati, calcolatori e teoria – servirà allo sviluppo dell'apprendimento automatico in bioinformatica e altrove. Una critica spesso sollevata ai metodi di apprendimento automatico è che sono degli approcci a scatola nera: non si può sempre capire come mai un dato modello dia una certa risposta. È importante capire comunque, che molte altre tecniche nella biologia molecolare contemporanea sono usate solamente su base empirica.

1.1 I dati biologici come sequenze di simboli

Una caratteristica fondamentale delle catene molecolari che sono le responsabili delle funzionalità e dell'evoluzione degli organismi viventi, è che possono essere descritte come *sequenze di simboli* scelti da un *alfabeto* di piccole dimensioni. Sperimentalmente le sequenze biologiche possono essere determinate con completa certezza, ed in una posizione di una determinata sequenza ci sarà una sola lettera e non un misto di diverse possibilità. La natura discreta dei dati genetici li rende abbastanza diversi da molti altri tipi di dati scientifici, dove le leggi fondamentali della fisica o la sofisticazione delle tecniche sperimentali necessitano di limiti inferiori di incertezza. Questa caratteristica ha anche un profondo impatto sui tipi di algoritmi che sono stati sviluppati e applicati per un'analisi di tipo computazionale. Mentre spesso lo scopo è quello di studiare una particolare sequenza con le sue funzionalità e la sua struttura molecolare, l'analisi tipicamente procede attraverso lo studio di un insieme di sequenze consistente di differenti versioni tra le varie specie oppure di differenti versioni della stessa specie. Un confronto tra le sequenze di varie specie deve tenere conto della natura "rumorosa" dei dati risultante in parte da eventi casuali amplificati dall'evoluzione: poi-

ché le sequenze di DNA o di amminoacidi aventi le stesse funzionalità saranno diverse, i modelli delle sequenze *devono essere probabilistici*.

1.1.1 Qualità delle basi di dati

Nonostante le sequenze di dati possono essere determinate sperimentalmente con alta precisione, non sono generalmente disponibili ai ricercatori se non con addizionale rumore proveniente da cattiva interpretazione dell'esperimento e incorretta gestione e memorizzazione nei database pubblici. Dato che le sequenze biologiche sono immagazzinate in formato elettronico e che i database pubblici sono curati da gruppi di persone altamente diversi, è forse comprensibile che in molti casi la percentuale di errori cresce come conseguenza della loro gestione. Un altro contributo non trascurabile a questa situazione è il modo in cui i dati sono memorizzati: le caratteristiche delle sequenze sono normalmente indicate per mezzo di una lista delle posizioni rilevanti in formato numerico e non tramite la loro funzionalità. Gli algoritmi di ricerca tentano di costruire approcci associativi per trovare specifiche sequenze che si accordano ad una rappresentazione spesso non troppo certa del loro contenuto: questo è molto diverso dal ricercare le sequenze in base alla loro funzionalità. In questa situazione è importante che la bioinfirmatica tenga conto di queste potenziali sorgenti di errore quando crea approcci di apprendimento automatico per la predizione e la classificazione. Tali tecniche danno un modo alternativo e potente di trovare informazioni e annotazioni erronee. In un insieme di dati, se qualcosa è difficile da imparare, è altamente probabile che esso rappresenti o un caso atipico o un errore. Una delle ragioni del successo delle tecniche di apprendimento automatico su domini di dati imperfetti, è che tali metodi sono capaci di gestire il rumore presente nei dati.

1.1.2 Ridondanza delle basi di dati

Un altro problema che ricorre nell'analisi delle sequenze biologiche, è le ridondanza dei dati. Diversi gruppi di persone possono inserire dati che si riferiscono a sequenze già esistenti, provocando delle duplicazioni delle stesse sequenze che posso essere o meno identiche a quelle presenti. L'uso di dati ridondanti può essere una sorgente per vari tipi di errore. Per lo più tutti gli approcci di apprendimen-

to automatico hanno dei problemi quando certe sequenze sono rappresentate molte volte: ad esempio, se l'insieme dei dati è usato per predire certe caratteristiche, le sequenze usate per addestrare il modello saranno molto correlate a quelle usate per il test del modello e si avrà una sovrastima della capacità predittiva. Benché siano stati proposti alcuni algoritmi per risolvere questo problema, è spesso meglio "pulire" l'insieme dei dati in modo da rendere tutte le sequenze ugualmente rappresentate; può essere cioè necessario evitare le sequenze "molto correlate". D'altra parte, una precisa definizione di "molto correlate" dipende strettamente dal problema in esame. Una strategia alternativa consiste nel pesare le varie sequenze in accordo alla loro novità. Un grande rischio di questo approccio è quello di pesare molto i dati errati. I metodi di apprendimento automatico sono capaci di estrarre le caratteristiche essenziali dai vari esempi e di scartare l'informazione non desiderata quando presente; inoltre sono capaci di trovare più complesse correlazioni e non linearità presenti nelle sequenze esaminando le proprietà statistiche dei segmenti che sono altamente informative.

1.2 L'allineamento delle sequenze

Per riuscire a trovare caratteristiche funzionali o strutturali comuni ad un insieme di dati, le nuove sequenze aggiunte ad un a base di dati sono normalmente allineate a tutte le altre presenti. Il problema fondamentale che sorge è di determinare quando le similarità sono sufficientemente grandi in modo che si possa inferire una somiglianza strutturale o funzionale dall'allineamento delle due sequenze. In altre parole, dato un metodo di allineamento che ha scoperto certe sovrapposizioni tra le sequenze, si può definire una soglia che mi dice se le sequenze sono omologhe? La risposta a questa domanda è non banale e dipende interamente dalla particolare struttura o funzionalità che si vuole esaminare; tale soglia sarà differente per ogni compito. La misura di quanto due sequenze sono allineate non è la stessa attraverso l'intero dominio delle sequenze e le corrispondenze prodotte dagli algoritmi di allineamento dipendono da vari fattori come ad esempio se l'algoritmo è stato progettato per ottimizzare una funzione localmente o globalmente. Alcuni problemi di rilevanza biologica hanno bisogno di effettuare un confronto globale tra due sequenze mentre altri analizzano solo una sottosequenza dei

segmenti: si parla allora di allineamento globale o locale. I classici algoritmi di allineamento sono basati sulla programmazione dinamica che ha però il difetto di essere esponenziale nel numero di sequenze: detto *K* il numero di sequenze da allineare ed *N* la lunghezza media delle sequenze, si ha che la complessità è dell'ordine di $O(N^K)$.

1.2.1 La matrice di sostituzione

Lo schema di allineamento è influenzato dalla scelta della matrice di sostituzione che stabilisce un costo per i vari eventi prodotti dall'evoluzione. Una matrice di sostituzione specifica un insieme di costi s_{ij} per la sostituzione di una lettera dell'alfabeto con un'altra. Alcune matrici sono generate da un modello semplificato dell'evoluzione delle proteine che involve le frequenze p_i degli amminoacidi e le frequenze di sostituzione q_{ij} di coppie di amminoacidi osservate negli allineamenti naturali delle sequenze. Una corrispondenza che interessa un amminoacido raro dovrebbe contare molto di più di una che interessa un comune amminoacido, mentre un errore tra due amminoacidi intercambiabili dovrebbe contribuire di più di uno tra due amminoacidi funzionalmente incorrelati. Un esempio di matrice di sostituzione che si riferisce a sequenze di DNA è la seguente:

	A	C	G	T	Gap (-)
A	0	1	1	1	2
C	1	0	1	1	2
G	1	1	0	1	2
T	1	1	1	0	2
Gap (-)	2	2	2	2	100

Si può dimostrare che ogni elemento della matrice di sostituzione può essere scritto nella forma:

$$s_{ij} = \frac{1}{\lambda}\left(\ln \frac{q_{ij}}{p_i p_j}\right)$$

dove λ è un fattore di scala. Modifiche del valore di λ cambieranno i valori assoluti dei costi ma non i costi relativi dei differenti allineamenti e questo non andrà ad influenzare l'allineamento. Le più semplici matrici di sostituzione sono quelle

in cui tutti gli elementi della diagonale hanno lo stesso valore positivo *s* mentre gli altri hanno lo stesso valore negativo $\underline{s}$. Date quindi due sequenze X_1 ,…, X_N e Y_1 ,…, Y_M che si sono evolute nel tempo, l'allineamento consiste nel minimizzare un certo costo indotto dalla matrice di sostituzione.

L'ambiente probabilistico

L'apprendimento automatico è un diretto discendente di una disciplina più vecchia, l'adattamento di modelli statistici ai dati osservati e come quest'ultima ha lo scopo di estrarre informazioni utili da un insieme di dati costruendo dei buoni modelli probabilistici. La principale novità che introduce è quella di automatizzare il processo finché possibile, spesso utilizzando modelli con un grande numero di parametri; questo approccio si adatta bene in quelle situazioni in cui sono disponibili grandi quantità di dati ma dove non esiste ancora una teoria consolidata. Inoltre, se i dati disponibili sono affetti da rumore e siamo quindi costretti a ragionare in un ambiente incerto, l'approccio bayesiano ci fornisce una teoria robusta che unifica differenti tecniche. Le principali caratteristiche dell'approccio bayesiano possono essere riassunte nei seguenti punti:

1. Utilizza ipotesi o modelli con tutta l'informazione di sottofondo e i dati disponibili.
2. Usa il linguaggio della teoria della probabilità per assegnare probabilità a priori a modelli o ipotesi.
3. Usa il calcolo delle probabilità per valutare le probabilità a posteriori delle ipotesi o dei modelli alla luce dei dati disponibili e per fornire una risposta *univoca* a certi quesiti.

Può essere provato in senso matematico stretto, che questo è il solo modo di ragionare consistente in presenza di incertezza.

2.1 Gli assiomi

In generale un modello può essere visto come un'ipotesi con la differenza che i modelli tendono ad essere ipotesi molto complesse che interessano un grande numero di parametri. In generale un modello verrà indicato con $M = M(w)$, dove w è il vettore di tutti i parametri. Data una certa conoscenza di fondo I, si può associare ad ogni ipotesi X un certo *grado di credenza* che indicheremo con $h(X \mid I)$ che deve soddisfare ai seguenti assiomi:

1. se $h(X \mid I) > h(Y \mid I)$ e $h(Y \mid I) > h(Z \mid I)$, allora $h(X \mid I) > h(Z \mid I)$
2. $h(\underline{X} \mid I) = F[h(X \mid I)]$ dove $\underline{X}$ rappresenta la negazione di X
3. $h(X, Y \mid I) = G[h(X \mid I), h(Y \mid X, I)]$

Si può provare che esiste sempre un fattore di scala k dei gradi di credenza per cui:

$$P(X \mid I) = k\, h(X \mid I) \in [0, 1]$$

dove P è unico e soddisfa tutte le regole delle probabilità. Da tali assiomi si può ricavare il fondamentale teorema di Bayes:

$$P(X \mid Y, I) = P(Y \mid X, I)\, P(X \mid I) / P(Y \mid I)$$

2.2 L'inferenza bayesiana

Il principale tipo di inferenza a cui siamo interessati è quella di derivare un modello $M = M(w)$ da un insieme di dati D. Per semplicità eliminiamo l'informazione di fondo dalle equazioni che seguiranno. Dal teorema di Bayes si ha che:

$$P(M \mid D) = P(D \mid M)\, P(M) / P(D)$$

La probabilità a priori $P(M)$ rappresenta la stima che il modello sia corretto prima di aver ottenuto qualunque dato. La probabilità a posteriori $P(M \mid D)$ rappresenta un aggiornamento della probabilità del modello M dopo che sono stati esaminati i dati D. Il termine $P(D \mid M)$ viene chiamato verosimiglianza. In molti casi le probabilità possono essere molto piccole, così che è più facile lavorare con i logaritmi:

$$\log P(M \mid D) = \log P(D \mid M) + \log P(M) - \log P(D).$$

Per quanto riguarda le probabilità a priori, quando le distribuzioni non sono uniformi, si utilizzano distribuzioni gaussiane, Gamma o di Dirichlet: la caratteristica fondamentale di tali distribuzioni è quella di non assegnare la probabilità nulla ad nessun evento, casa invece che può accedere con una distribuzione uniforme.

2.2.1 Stima dei parametri e selezione di un modello

Data una certa classe di modelli, due modelli M_1 e M_2 possono essere confrontati tramite le loro probabilità $P(M_1 \mid D)$ e $P(M_2 \mid D)$. Spesso si pone il problema di trovare il miglior modello di una classe, cioè di trovare un insieme di parametri w che massimizzi la probabilità a posteriori $P(M \mid D)$ o $\log P(M \mid D)$: questa è chiamata stima dei parametri massimizzando la probabilità a posteriori (MAP). Per lavorare con quantità positive, questo equivale a minimizzare:

$$E = - \log P(M \mid D) = - \log P(D \mid M) - \log P(M) + \log P(D).$$

Il termine $P(D)$ gioca il ruolo di costante di normalizzazione e non dipende dai parametri w, quindi il suo ruolo è ininfluente per l'ottimizzazione. Se la probabilità a priori $P(M)$ è uniforme su tutti i modelli considerati, allora il problema si riduce a trovare il massimo di $P(D \mid M)$ o $\log P(D \mid M)$: questa è chiamata stima dei parametri massimizzando la verosimiglianza (ML). Riassumendo, la stima MAP cerca di minimizzare la quantità

$$E = - \log P(D \mid M) - \log P(M)$$

mentre la stima ML cerca di minimizzare la quantità

$$E = - \log P(D \mid M).$$

Trovare un modello ottimo per una data classe di modelli ha senso solo se la distribuzione $P(M \mid D)$ è concentrata attorno ad un unico picco. In molti casi, in cui c'è un elevato grado di incertezza ed in cui la quantità di dati è relativamente piccola, siamo interessati alla funzione $P(M \mid D)$ sopra l'intero spazio dei modelli piuttosto che solo su un certo massimo e soprattutto siamo interessati nel calcolare le medie rispetto a $P(M \mid D)$. A parità di probabilità, è meglio scegliere il modello più semplice.

2.3 I modelli grafici

Una volta che l'ambiente probabilistico bayesiano è stato definito, l'idea successiva è quella di utilizzare i modelli grafici. Poiché nell'analisi bayesiana il punto di partenza è per lo più sempre una distribuzione di probabilità di grado elevato, tale espressione deve essere decomposta e semplificata. La più comune semplificazione è quella di assumere che alcune variabili siano indipendenti o più precisamente che alcuni insiemi di variabili siano indipendenti, data la dipendenza condizionale con altri insiemi di variabili. Queste relazioni di indipendenza possono essere spesso rappresentate da un grafo dove le variabili sono associate con i nodi e un collegamento mancante rappresenta una particolare relazione di indipendenza. Le relazioni di indipendenza permettono la fattorizzazione della distribuzione di probabilità di grado elevato in un prodotto di semplici locali distribuzioni associate a piccoli gruppi di variabili correlate fra loro. I modelli grafici possono essere suddivisi in due grandi categorie a seconda se gli archi associati sono orientati o meno. I modelli grafici non diretti sono impiegati in quelle situazioni in cui le interazioni sono considerate completamente simmetriche, mentre i modelli grafici diretti sono utili in quelle situazioni in cui le interazioni non sono simmetriche e riflettono relazioni casuali o irreversibilità temporale. Il linguaggio di rappresentazione dei modelli grafici è utile nella maggior parte delle applicazioni di apprendimento automatico; le reti bayesiane, le reti neurali, i modelli di Markov nascosti rappresentano un esempio di tali metodologie.

2.4 Due semplici modelli probabilistici

Il più semplice modello probabilistico per le sequenze biologiche consiste nell'usare un singolo dado. Supponiamo che i dati D siano delle sequenze di DNA su un alfabeto A = {A, C, G, T} di quattro lettere. Il più semplice modello è quello di assumere che le sequenze siano state ottenute lanciando un dado con quattro facce (una per lettera dell'alfabeto) e che i vari lanci siano indipendenti tra loro. Qualunque sia la classe di modelli utilizzata, il primo passo consiste nel calcolare esplicitamente la verosimiglianza e nell'assegnare una probabilità a priori. Supponiamo che i dati siano costituiti da una sola sequenza di lunghezza N: $D = \{S\}$ con $S = S_1, \ldots, S_N$ e $S_i \in A$. I parametri del nostro modello M sono le quattro probabilità p_A, p_C, p_G, p_T , la cui somma deve fare 1. La verosimiglianza è data da:

$$P(D \mid M) = \prod_{X \in A} p_X^{n_X}$$

dove n_X è il numero di volte che la lettera X appare nella sequenza S. Un altro modello semplice consiste nell'usare più dadi. Supponiamo che i dati siano costituiti da K sequenze ognuna di lunghezza N. Per esempio, si potrebbe pensare ad un allineamento multiplo delle K sequenze dove il simbolo "-" potrebbe essere considerato un simbolo dell'alfabeto. In questo modello assumiamo che ci siano N dadi indipendenti, uno per ciascuna posizione e che ogni sequenza è il risultato di aver lanciato gli N dadi in un ordine fissato. Sia p_X^i la probabilità di produrre la lettera X con il dado numero i e sia n_X^i il numero di volte che la lettera X appare nella posizione i; poiché si assume che il dado e le sequenze siano indipendenti, la verosimiglianza è data da:

$$P(D \mid M) = \prod_{i=1}^{N} \prod_{X \in A} p_X^{n_X^i}$$

I successivi modelli che costruiremo non sono altro che raffinamenti di questi semplici modelli che prendono in considerazione aspetti aggiuntivi di cui non si è tenuto conto: i modelli di Markov nascosti sono un'ottima classe di modelli per riuscire ad allineare gruppi di sequenze con una complessità lineare nel numero di sequenze.

Gli algoritmi di apprendimento

Dopo aver costruito un modello parametrizzato $M(w)$ per l'insieme dei dati, il passo successivo consiste nell'eseguire uno dei seguenti punti:

- la stima completa della distribuzione $P(w, D)$ e della probabilità a posteriori $P(w| D)$
- la stima dell'insieme ottimo dei parametri w massimizzando $P(w| D)$
- la stima dei valori medi rispetto alla probabilità a posteriori cioè, per esempio, dell'integrale della forma $E(f) = \int f(w)P(w \mid D)dw$.

Gli algoritmi di apprendimento possono quindi essere suddivisi in tre categorie, a seconda se l'obiettivo è quello di stimare una densità di probabilità, un insieme dei suoi parametri o le corrispondenti medie. Si può affermare anche che un qualunque problema può essere riformulato come un problema di ottimizzazione.

3.1 Programmazione dinamica

Il termine programmazione dinamica si riferisce ad una tecnica generale di ottimizzazione che può essere applicata ad un problema scomponibile in sottoproblemi simili ma di dimensione più piccola, così che la soluzione del problema originario può essere ottenuta mettendo insieme le soluzioni dei problemi più piccoli. Il tipico problema a cui può essere applicata la programmazione dinamica è quello

di trovare il percorso più breve tra due nodi di un grafo. Chiaramente il percorso più breve dal nodo *A* al nodo *B* che passa per il nodo *C* è la concatenazione del percorso più corto da *A* a *C* con il percorso più corto da *C* ad *A*. Questo è anche noto come il principio di Bellman. La soluzione del problema generale è costruita ricorsivamente unendo le soluzioni dei sottoproblemi più semplici. Gli algoritmi di allineamento possono essere visti in termini di trovare il percorso più breve in un appropriato grafo con un determinata metrica. Allineare due sequenze di lunghezza N richiede di trovare il percorso più breve in un grafo con N^2 vertici. Poiché la programmazione dinamica essenzialmente richiede di visitare tutti i vertici una volta, è facile vedere che la complessità in tempo è dell'ordine di $O(N^2)$. La programmazione dinamica e l'algoritmo di Viterbi saranno usati per calcolare la verosimiglianza e l'allineamento delle sequenze usando i modelli di Markov nascosti.

3.2 Discesa lungo il gradiente

Spesso siamo interessati alla stima dei parametri, cioè a trovare il miglior modello $M(w)$ che minimizza la probabilità a posteriori $f(w) = -\log P(w \mid D)$ o la verosimiglianza $-\log P(D \mid w)$. Tutte le volte che $f(w)$ è differenziabile, si può cercare di trovare i suoi minimi usando uno dei più vecchi algoritmi di ottimizzazione, la discesa lungo il gradiente. Come indica il nome, la discesa lungo il gradiente è una procedura iterativa che può essere espressa vettorialmente come:

$$w^{t+1} = w^t - \eta \frac{\partial f}{\partial w^t}$$

dove η è la lunghezza del passo (o learning rate) che può essere fissa o aggiustata durante il processo di apprendimento. Mentre l'idea generale della discesa lungo il gradiente è semplice, usando modelli parametrizzati molto complessi si possono avere varie implementazioni, dipendenti da come realmente viene calcolato il gradiente. Nei modelli grafici spesso si richiede la propagazione dell'informazione all'indietro: questo è il caso dei modelli di Markov nascosti in cui servono una procedura di propagazione in avanti e una all'indietro (la procedura forward ba-

ckward). Ovviamente il metodo di discesa lungo il gradiente dipende dalle condizioni iniziali e se la funzione che deve essere ottimizzata ha molti minimi o massimi locali, non è detto che si riesca ad individuare l'ottimo globale.

3.3 Algoritmo EM

Un'altra classe importante di algoritmi di ottimizzazione è la massimizzazione della media (EM) (che nel casi dei modelli di Markov nascosti è chiamata algoritmo di Baum-Welch). L'utilità di questi algoritmi è rivolta soprattutto ai modelli di Markov nascosti e fa uso del concetto di energia libera. L'algoritmo EM è utile nei modelli con variabili nascoste: tipici esempi di variabili nascoste sono i dati mancanti o non osservabili e i nodi nascosti in un modello grafico. Se D denota i dati, supponiamo che sia disponibile una versione parametrizzata della distribuzione di probabilità congiunta delle variabili osservate e nascoste $P(D, H \mid w)$. Nel caso di maggiore interesse, w indica i parametri di un modello. Supponiamo di voler massimizzare la verosimiglianza $\log P(D \mid w)$ (le stesse idee possono essere estese ad una stime di tipo MAP). Poiché in generale è difficile ottimizzare $\log P(D \mid w)$ direttamente, l'idea base è quella di cercare di ottimizzare la sua media:

$$E(\log P(D \mid w)) = E(\log P(D, H \mid w) - \log P(H \mid D, w)).$$

L'algoritmo EM è un algoritmo iterativo che procede in due passi che si alternano, il calcolo della media ed il passo di massimizzazione. Durante il calcolo della media, viene calcolata la distribuzione delle variabili nascoste, avendo a disposizione i dati osservati e la corrente stima di w. Durante il passo di massimizzazione, i parametri sono aggiornati al miglior valore possibile, essendo nota la presunta distribuzione delle variabili nascoste. I calcoli dei vari passi fanno uso dell'energia libera del modello: si tratta di minimizzare tale funzione che è strettamente collegata alla media della verosimiglianza.

3.4 Vari aspetti degli algoritmi di apprendimento

In un modo o nell'altro, la scelta di un modello deve tenere conto del numero dei parametri per cercare di evitare fenomeni di sovrapprendimento o sotto-apprendimento dei dati. Un approccio per questo problema è quello di pesare la funzione da apprendere con un termine che tiene conto della complessità del modello. Un modo per evitare overfitting quando il modello ha troppi parametri rispetto ai dati, è quello di suddividere l'insieme dei dati in due parti: una parte servirà per addestrare il modello e verrà chiamata training set, mentre l'altra servirà per valutare le prestazioni del modello e verrà chiamata test set. Ognuno dei due insiemi di dati darà origine ad un certo errore: l'errore di addestramento decrescerà monotonicamente al crescere del numero delle epoche, mentre l'errore sull'insieme di test raggiungerà un minimo e poi comincerà a crescere. Il fenomeno dell'overfitting è associato con la memorizzazione dei dati insieme al loro rumore fino al punto che è dannoso per la generalizzazione. L'approccio corretto in una tale situazione sarebbe quello di modificare il modello. Un'altra alternativa è quella di fermare l'addestramento quando l'errore sul test set incomincia a crescere o quando l'errore sul training set ha raggiunto una certa soglia. Comunque quest'ultima tecnica può lasciare un parziale overfitting dei dati. Inoltre, per ottenere un corretto addestramento, tutte le classi di dati dovrebbero essere egualmente rappresentate nel training set.

Gli algoritmi di apprendimento si possono suddividere in due classi, a seconda se l'aggiornamento avviene dopo ogni esempio oppure dopo l'intero insieme di esempi. L'addestramento è detto on-line se l'aggiustamento dei parametri del modello avviene dopo la presentazione di ogni esempio, mentre è detto batch se i parametri sono aggiornati dopo la presentazione di un grande numero di esempi, se non di tutti. L'apprendimento on-line non richiede di tenere in memoria molti esempi ed è più flessibile e più facile da implementare; può però introdurre un certo grado di casualità legato al fatto che l'aggiornamento avviene sulla base di un solo esempio. Può essere dimostrato che l'apprendimento on-line fatto con un tasso di apprendimento sufficientemente piccolo, approssima l'apprendi-mento batch.

Un'ultima osservazione riguarda i modelli che si ottengono dopo l'addestramento. Quando un modello complesso viene adattato ai dati per mezzo di una

ottimizzazione di tipo ML o MAP, si ottengono dei parametri differenti se si variano certi fattori durante la procedura di apprendimento come ad esempio la procedura di addestramento, l'ordine di presentazione degli esempi, il training set. Inoltre possono essere impiegate classi di modelli differenti. È naturale pensare che la migliore classificazione o predizione possa essere raggiunta mediando i parametri dei vari modelli ottenuti in modi diversi.

I modelli di Markov nascosti (HMM)

Il problema che si pone ogni volta che viene individuata una nuova sequenza biologica, è quello di vedere se presenta delle similarità con le altre sequenze già presenti nell'archivio: questo può essere fatto confrontando le sequenze due a due. Il problema diventa ancora più complesso nel caso sia abbia a che fare con sequenze incomplete o con frammenti di sequenze, cosa che è molto comune se si ha che fare con il genoma umano. È certamente di grande interesse riconoscere e classificare tali frammenti, anche perché si pensa che coprano una sostanziale frazione se non tutto il genoma umano. Gli HMM formano una classe utile di modelli grafici probabilistici che può servire a confrontare non coppie di sequenze, ma insiemi di sequenze, per scoprirne le caratteristiche comuni.

4.1 Definizione

Un HMM discreto del 1° ordine è un modello probabilistico per le serie temporali definito da:

- un insieme finito di stati S
- un alfabeto discreto di simboli A
- una matrice delle probabilità di transizione $T = (P(i \mid j))$
- una matrice delle probabilità di emissione $E = (P(X \mid i))$

dove $P(i \mid j)$ indica la probabilità di transizione dal nodo j verso il nodo i e $P(X \mid i)$ indica la probabilità che il nodo i emetta il simbolo X. Il sistema evolve casualmente da uno stato all'altro mentre emette simboli dall'alfabeto. Si parla di modello del 1° ordine perché le transizioni e le emissioni dipendono solo dallo stato corrente e non dal passato. In aggiunta agli stati precedentemente nominati, ci sono due stati speciali, lo stato *Start* e lo stato *End*. Al tempo zero il sistema si troverà sempre nello stato *Start*. Le probabilità di transizione e quelle di emissione sono i parametri del modello. Data una certa architettura di HMM, è di interesse calcolare la probabilità di una sequenza dato il modello (verosimiglianza), qual è il percorso più probabile associato con una data sequenza e infine supposti non noti i parametri, si vogliono valutare i loro valori alla luce dei dati osservati.

4.1.1 HMM per le sequenze biologiche

Per quanto riguarda l'alfabeto, si avrà un alfabeto costituito da 20 amminoacidi nel caso delle proteine, mentre si avrà un alfabeto costituito da 4 nucleotidi nel caso di DNA o RNA. La scelta di una architettura per l'HMM dipende fortemente dal problema. Nel caso di sequenze biologiche, l'aspetto lineare delle sequenze è ben rappresentato dalla cosiddetta architettura left-right.

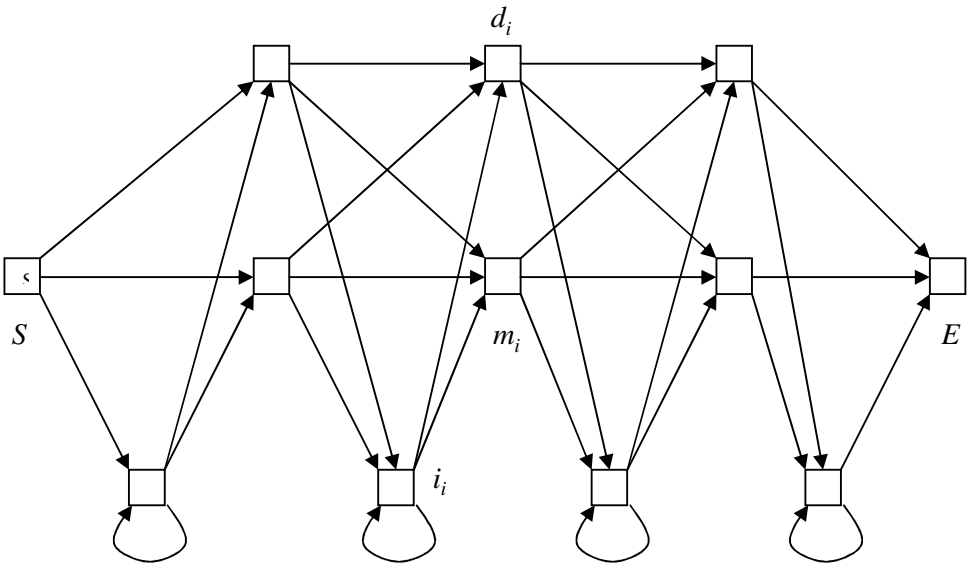


Figura 1: L'architettura lineare standard

Un'architettura left-right non permette il ritorno in uno stato se è avvenuta una transizione da quello stato in un altro stato. L'architettura left-right più usata per le sequenze biologiche è l'architettura standard lineare della figura 1.

Cominciamo cercando di modellare una famiglia di sequenze correlate fra loro. L'architettura standard rappresenta una semplice ma fondamentale variazione del modello con il dado. Il principale problema di tale modello è che la lunghezza delle sequenze non è costante ma sono presenti cancellazioni e inserimenti: l'architettura standard risolve tale problema in modo semplice. Oltre ai due stati *Start* ed *End*, ci sono altri tre tipi di stati chiamati rispettivamente *main*, *insert* e *delete*. L'insieme totale dei stati viene indicato con $S = \{Start, m_1, \ldots, m_N, i_1, \ldots, i_{N+1}, d_1, \ldots, d_N, End\}$. La lunghezza del modello N è di solito uguale alla lunghezza media dell'insieme di sequenze. Gli stati main ed insert emettono sempre un simbolo dell'alfabeto, mentre i delete sono muti. Come si può vedere, il modello contiene uno stato di cancellazione per ogni nodo di tipo main, mentre c'è una corrispondenza biunivoca tra gli stati di inserimento e le transizioni tra i nodi della spina dorsale (fila centrale); inoltre gli stati di inserimento hanno un loop su se stessi. L'architettura del grafo e la tipologia dei nodi risolvono in modo efficiente il problema delle cancellazioni e degli inserimenti; il loop sugli stati di inserimento consente anche degli inserimenti multipli.

Cerchiamo adesso di valutare il numero di parametri di tale modello. Detta $|A|$ la cardinalità dell'alfabeto, si hanno $(N + N + 1)\ |A| = (2N + 1)\ |A|$ parametri di emissione. Per quanto riguarda il numero dei parametri di transizione, valutiamo gli archi uscenti da ogni nodo: tutti i nodi hanno tre archi uscenti meno che lo stato *End* e gli ultimi stati di ogni tipo; quindi $(3N + 2 + 3\ (N - 1) + 2 + 3N + 2) = = 9N + 3$ parametri di transizione. Nel caso di DNA o RNA il numero totale di parametri è $17N + 4$, mentre se si ha che fare con le proteine tale numero sale a $49N + 4$. Inoltre si hanno $2N + 1$ vincoli di normalizzazione sulle probabilità di emissione e $3N + 1$ vincoli di normalizzazione sulle probabilità di transizione.

4.1.2 Informazione a priori e inizializzazione

A causa della natura multinomiale del modello associata alle emissioni ed alle transizioni, la probabilità a priori più naturale per i parametri è la distribuzione di Dirichlet. Per transizioni, si possono usare delle distribuzioni di Dirichlet con lo

stesso parametro oppure con parametri differenti; è da notare che il vettore su cui è centrata la distribuzione non è uniforme, in quanto dovrebbero essere predominanti le transizioni verso gli stati main. Per i parametri di emissione, si possono usare distribuzioni con lo stesso parametro o con parametri distinti, mentre il vettore su cui è centrata la distribuzione può essere uniforme od uguale alla composizione media frequenziale del training set. Per quanto riguarda l'inizializzazione dei parametri, le transizioni devono essere aggiustate in modo da favorire gli spostamenti verso gli stati main, mentre le emissioni possono essere inizializzate in modo uniforme, a caso o con le frequenze delle composizioni medie del training set. Una inizializzazione che devia molto da quella uniforme può produrre degli effetti indesiderati se viene usato l'algoritmo di Viterbi per l'addestramento.

4.2 Verosimiglianza ed algoritmi fondamentali

In questo paragrafo vedremo come calcolare la verosimiglianza e il percorso più probabile associato ad una data sequenza biologica. Tutti gli algoritmi usati sono di tipo ricorsivo e possono essere visti come un'applicazione di programmazione dinamica o come una propagazione dei dati nel grafo diretto associato con l'HMM. Questi algoritmi costituiscono la base per i successivi algoritmi di apprendimento.

Cerchiamo innanzitutto di calcolare la verosimiglianza $P(S \mid w)$ di una sequenza $S = S_1,\ldots, S_t ,\ldots, S_T$ rispetto ad un certo HMM $M = M(w)$ dove w è l'insieme dei parametri. Si definisce un percorso π in M, un insieme di stati consecutivi che partono dallo stato *Start* e che finiscono nello stato *End*, con associata l'emissione di una lettera per ogni stato di emissione (main ed insert). La probabilità di trovare un percorso le cui emissioni coincidano con quelle della sequenza osservata è data da:

$$P(S,\pi \mid w) = \prod_{Start}^{End} P(j \mid i) \prod_{t=1}^{T} P(S_t \mid i)$$

dove il primo prodotto riguarda tutte le transizioni lungo il percorso π, ed il secondo corrisponde a tutte le emissioni delle lettere della sequenza dagli stati *i* facenti parte del percorso π. Ovviamente se una sequenza di emissioni lungo il percorso non coincide con *S*, si ha che $P(S, \pi \mid w) = 0$. La verosimiglianza può essere calcolata come:

$$P(S \mid w) = \sum_{\pi} P(S, \pi \mid w) .$$

Questa espressione non costituisce un metodo di calcolo computazionalmente efficiente per calcolare la verosimiglianza, in quanto il numero di percorsi in un'architettura è tipicamente esponenziale. Si deve quindi trovare un metodo più efficiente per calcolare tale valore, e questo può essere fatto usando un meccanismo di propagazione iterativo lungo il grafo dell'HMM. Di seguito vengono riportate le due procedure fondamentali di propagazione in avanti ed indietro che serviranno come base per le altre procedure.

4.2.1 L'algoritmo Forward

Definiamo innanzitutto la probabilità congiunta $\alpha(i, t)$ di trovarsi nello stato *i* avendo osservato i primi $t - 1$ simboli della sequenza, dati i parametri del modello, nel seguente modo:

$$\alpha(i,t) = P\left(N_t = i, S_1,...,S_{t-1} \mid w\right)$$

dove $N_t = i$ indica l'evento di trovarsi nel nodo *i* al tempo *t*, mentre S_1 ,…, S_{t-1} indicano i simboli della sequenza al tempo 1,…, $t - 1$. Ovviamente tutte le probabilità utilizzate sono condizionate dal modello, quindi per semplicità di notazione, non sempre verrà riportato questo particolare. L'algoritmo iterativo per il calcolo di tali probabilità può essere inizializzato nel seguente modo:

$$\alpha(Start,0) = 1$$
$$\alpha(emis,0) = 0$$

dove *emis* si riferisce a tutti gli altri nodi di emissione. Si può osservare che:

$$P(S \mid w) = \alpha(End, T+1).$$

Il generico elemento $\alpha(i, t)$ può essere calcolato ricorsivamente nel seguente modo, distinguendo i nodi di emissione da quelli di cancellazione:

$$\alpha(i,t) = P\left(N_t = i, S_1, ..., S_{t-1}\right) =$$

$$= \sum_{j \in G(i)} P(N_{t-1} = j, S_1, ..., S_{t-2}) P(N_t = i \mid N_{t-1} = j) P(S_{t-1} \mid N_t = i) =$$

$$= \sum_{j \in G(i)} \alpha(j, t-1) P(i \mid j) P(S_{t-1} \mid i) \quad se \quad i \in E$$

$$= \sum_{j \in G(i)} P(N_t = j, S_1, ..., S_{t-1}) P(N_t = i \mid N_{t-1} = j) =$$

$$= \sum_{j \in G(i)} \alpha(j, t) P(i \mid j) \quad se \quad i \in D$$

dove $P(i \mid j)$ indica la probabilità di transizione dal nodo j verso il nodo i e $P(S_{t-1} \mid i)$ indica la probabilità che il nodo i emetta il $(t - 1)$-esimo simbolo della sequenza; E indica l'insieme di tutti i nodi di emissione, D quello dei nodi di cancellazione, $G(i)$ l'insieme dei genitori del nodo i. Ovviamente $P(N_t = i \mid N_{t-1} = j) = P(i \mid j)$ poiché le transizioni non dipendono dal tempo e così anche per le emissioni. Da qui in poi si sottenderà la dipendenza dal tempo quando non necessaria e dunque si scriverà, per esempio $P(N_t = i \mid N_{t-1} = j)$ come $P(i \mid j)$. Per quanto riguarda gli stati di emissione, il generico $\alpha(i, t)$ sarà dato dalla somma su tutti i genitori del nodo i degli $\alpha(G(i), t - 1)$ moltiplicati per la probabilità di transizione dell'arco che va da $G(i)$ ad i, il tutto moltiplicato per la probabilità di emissione del simbolo S_{t-1} da parte del nodo i. Notiamo che per gli stati di cancellazione è necessario uti-

lizzare una relazione differente: questo perché tali stati non emettono alcun simbolo dell'alfabeto e quindi la probabilità $\alpha(i, t)$ sarà data dalla somma su tutti i genitori del nodo i degli $\alpha(G(i), t)$ moltiplicati per la probabilità di transizione dell'arco che va da $G(i)$ ad i. Questo potrebbe creare qualche problema nella fase di aggiornamento degli stati di cancellazione, poiché in generale al tempo t non sono noti tutti gli $\alpha(i, t)$, in quanto la conoscenza di tali probabilità dipenderà dall'ordine in cui le calcoliamo. Se però ogni stato di cancellazione ha al più un genitore che sia esso stesso stato di cancellazione, sfruttando questa proprietà che induce un ordinamento tra gli stati di cancellazione, possiamo calcolare gli $\alpha(i, t)$ per tali stati nel seguente modo:

1. Si calcolano prima tutti gli $\alpha(i, t)$ per gli stati di emissione.
2. Si considera poi quello stato di cancellazione che non ha tra i suoi genitori un altro stato di cancellazione e si calcola il corrispondente valore di $\alpha(i, t)$. Quindi si procede al calcolo di $\alpha(i, t)$ per quel figlio del precedente nodo che è stato di cancellazione; si prosegue così ricorsivamente in modo da disporre, per ogni nodo di cancellazione, di tutti gli $\alpha(G(i), t)$ necessari.

Al tempo $T + 1$, come anche al tempo 0, non disponiamo di alcun simbolo della sequenza e dunque ci limitiamo ad aggiornare le precedenti formule usando la sola probabilità di transizione. Ecco quella parte di programma che si occupa di aggiornare gli $\alpha(i, t)$:

```
parnodi[0].setAlfa(0, 1);

for (t = 0; t <= l+1; t++)
{
   // aggiornamento dei nodi di emissione
   for (i = 0; i < nnodi; i++)
   if (nodi[i].getTipo() != 2)
   {
      if (t == 0 && i != 0)
         parnodi[i].setAlfa(0, 0);
      else if (t > 0 && t <= l)
         parnodi[i].calcAlfa(t, *archi, seq[t-1], parnodi);
      else if (t == l+1)
         parnodi[i].calcAlfa(t, *archi, 0, parnodi, 1);
   }

   // aggiornamento dei nodi di cancellazione in ordine
```

```
    for (i = 0; i < nnodi; i++)
    if (nodi[i].getTipo() == 2)
      parnodi[i].calcAlfa(t, *archi, seq[t-1], parnodi);

    // scalatura degli alfa
    scalcoef[t] = 0;
    for (i = 0; i < nnodi; i++)
      scalcoef[t] += parnodi[i].getAlfa(t);
    for (i = 0; i < nnodi; i++)
      parnodi[i].setAlfa(t, parnodi[i].getAlfa(t)/scalcoef[t]);
  }
```

Si fa uso di due cicli annidati: quello esterno scandisce tutti i tempi a partire da 0 fino a $T + 1$ incluso, mentre quello interno esamina tutti i nodi. Come prima cosa vengono aggiornati i nodi di emissione (tipo diverso da 2), poi quelli di cancellazione (tipo uguale a 2) secondo l'ordine imposto dall'architettura del grafo. Infine tutti i coefficienti calcolati vengono scalati con un opportuno coefficiente di scalatura, pari alla somma di tutti gli $\alpha(i, t)$ a quel tempo: si terrà conto di questi coefficienti quando ne sarà necessario, in modo che i risultati finali non ne siano influenzati. È tuttavia molto utile scalare gli $\alpha(i, t)$ per stabilizzare computazionalmente l'algoritmo. Valutiamo la complessità di tale algoritmo: detta T la lunghezza della sequenza corrente, si ha che il numero totale di nodi dell'architettura è $N + 2 + N + 1 + N = 3N + 3$. Tralasciando termini moltiplicati e costanti che non influiscono sulla complessità asintotica, si ha che la complessità dell'algoritmo per ogni sequenza è data da $O(NT)$. Supponendo che la lunghezza media delle sequenze sia uguale alla lunghezza del modello (ipotesi generalmente verificata), si ottiene una complessità $O(N^2)$ in tempo e spazio per ogni sequenza. La complessità sarebbe notevolmente maggiore qualora la connessione dei nodi di cancellazione fosse qualunque o comunque differente da quella lineare qui esaminata. Si otterrebbe una complessità superiore anche nel caso in cui la procedura per calcolare gli $\alpha(i, t)$ avesse la necessità di reperire i genitori del nodo i: per evitare questo ulteriore incremento, ogni nodo porta con sé anche l'informazione necessaria a trovare i suoi genitori senza effettuare iterazioni aggiuntive.

4.2.2 L'algoritmo Backward

Definiamo adesso la probabilità $\beta(i, t)$ di aver osservato i simboli della sequenza da S_t a S_T dato lo stato i e dati i parametri del modello, nel seguente modo:

$$\beta(i,t) = P(S_t,...,S_T \mid N_t = i, w).$$

L'algoritmo iterativo per il calcolo di tali probabilità può essere inizializzato nel seguente modo:

$$\beta(End, T+1) = 1$$
$$\beta(emis, T+1) = 0$$

in quanto si parte sempre dal nodo *End* (in quanto è una procedura di propagazione all'indietro): il tempo $T + 1$ sta ad indicare che tutta la sequenza è stata emessa. Tutti gli altri nodi di emissione indicati da *emis* vengono posti a 0 al tempo $T + 1$. Il generico elemento $\beta(i, t)$ può essere calcolato ricorsivamente nel seguente modo:

$$\beta(i,t) = P(S_t,...,S_T \mid N_t = i) =$$

$$= \sum_{j \in F(i) \cap E} P(S_{t+1},...,S_T \mid N_{t+1} = j) P(j \mid i) P(S_t \mid j) +$$

$$+ \sum_{j \in F(i) \cap D} P(S_t,...,S_T \mid N_t = j) P(j \mid i) =$$

$$= \sum_{j \in F(i) \cap E} \beta(j, t+1) P(j \mid i) P(S_t \mid j) + \sum_{j \in F(i) \cap D} \beta(j,t) P(j \mid i)$$

Il coefficiente $\beta(i, t)$ viene calcolato eseguendo una somma su tutti i figli $F(i)$ del nodo i e distinguendo i due seguenti casi:

1. Il figlio è un nodo di cancellazione: in questo caso si calcola la probabilità $\beta(F(i), t)$ moltiplicata per la probabilità di transizione $P(j \mid i)$.
2. Il figlio è un nodo di emissione: in questo caso si calcola la probabilità $\beta(F(i), t + 1)$ moltiplicata per la probabilità di emissione $P(S_t \mid j)$ (se esiste) e per la probabilità di transizione $P(i \mid j)$.

L'aggiornamento dei $\beta(i, t)$ può dare dei problemi se il nodo i ha come figlio uno stato di cancellazione, in quanto in generale al tempo t non abbiamo calcolato ancora alcun $\beta(i, t)$ che riguardi i nodi di cancellazione. Se però ogni stato di cancellazione ha al più un figlio che sia esso stesso stato di cancellazione, sfruttando questa proprietà che induce un ordinamento tra gli stati di cancellazione, possiamo calcolare i $\beta(i, t)$ nel seguente modo:

1. Per ogni nodo i, si calcola la parte di $\beta(i, t)$ definita dal punto 2. precedente, cioè la parte che riguarda i figli di emissione: non si ha alcun problema in quanto si utilizzano i $\beta(i, t)$ già calcolati in precedenza relativi a tempi successivi.
2. Si considera quindi il nodo di cancellazione che non ha tra i suoi figli un altro nodo di cancellazione: questo può essere calcolato correttamente in quanto si dispone di tutti i dati necessari. A partire da questo nodo, si propaga il calcolo dei $\beta(i, t)$ a tutti i suoi figli di cancellazione in modo ricorsivo.
3. Adesso che si dispone di tutti i $\beta(i, t)$ dei nodi di cancellazione, possiamo completare il calcolo anche per gli altri nodi di emissione.

Ordinando i nodi del grafo in modo opportuno, se ad ogni nodo di emissione è collegato al più un nodo di cancellazione, è possibile fondere i passi 2. e 3. in un solo passo. Ecco quella parte di programma che si occupa di aggiornare i $\beta(i, t)$:

```
parnodi[nodoend].setBeta(l+1, 1);

for (t = l+1; t >= 0; t--)
{
  // aggiornamento dei soli nodi di emissione
  for (i = nnodi-1; i >= 0; i--)
  {
    if (t == l+1 && i != nodoend)
```

```
        parnodi[i].setBeta(l+1, 0);
      else if (t == l)
        parnodi[i].calcBeta(t, *archi, 0, parnodi, 1);
      else if (t < l)
        parnodi[i].calcBeta(t, *archi, seq[t], parnodi);
    }

    // aggiornamento dei rimanenti nodi di cancellazione
    for (i = nnodi-1; i >= 0; i--)
      parnodi[i].calcBeta(t, *archi, 0, parnodi, 2);

    // scalatura dei beta
    for (i = 0; i < nnodi; i++)
      parnodi[i].setBeta(t, parnodi[i].getBeta(t)/scalcoef[t]);
  }
```

Questo algoritmo è analogo al precedente con la sola differenza che i coefficienti di scalatura sono gli stessi dell'algoritmo precedente: tali coefficienti non devono essere quindi ricalcolati. La complessità asintotica di tale algoritmo è $O(NT)$, dove T e' la lunghezza della sequenza corrente ed N la lunghezza del modello: supponendo che la lunghezza media delle sequenze sia uguale alla lunghezza del modello, si ottiene una complessità $O(N^2)$ in tempo e spazio per ogni sequenza, la stessa del precedente algoritmo.

4.2.3 Utilizzo dei coefficienti alfa e beta

Noti tutti gli $\alpha(i, t)$ e tutti i $\beta(i, t)$, possiamo calcolare la probabilità $P(N_t = i \mid S)$ di trovarsi in un certo nodo i al tempo t, data una sequenza S: tale probabilità verrà indicata con $\gamma(i, t)$. Per far questo, calcoliamo prima la probabilità congiunta $P(N_t = i, S)$ dove S è la sequenza:

$$P(N_t = i, S) = \alpha(i,t)\beta(i,t) = P(N_t = i, S_1,...,S_{t-1})P(S_t,...,S_T \mid N_t = i).$$

Se poi marginaliziamo su i, otteniamo la probabilità della sequenza $P(S)$. A questo punto è sufficiente dividere la probabilità congiunta $P(N_t = i, S)$ per $P(S)$ per ottenere $P(N_t = i \mid S)$. Quindi il coefficiente $\gamma(i, t)$ e' dato da:

$$\gamma(i,t) = P(N_t = i, | S, w) = \frac{\alpha(i,t)\beta(i,t)}{\sum_{j \in I} \alpha(j,t)\beta(j,t)}$$

dove I indica l'insieme di tutti i nodi del grafo. È utile anche calcolare la probabilità $\gamma(j, i, t)$ della transizione dal nodo i al nodo j al tempo t che è data da:

$$\gamma(j, i, t) = P(N_t = j, N_{t-1} = i \mid S, w) \text{ se } j \in E$$
$$\gamma(j, i, t) = P(N_t = j, N_t = i \mid S, w) \text{ se } j \in D.$$

Si può procede in modo analogo, calcolando prima le due probabilità congiunte $P(N_t = j, N_{t-1} = i, S)$ e $P(N_t = j, N_t = i, S)$ e quindi dividendo per $P(S)$. Il coefficiente $\gamma(j, i, t)$ è dunque dato da:

$$\gamma(j, i, t) = \alpha(i, t)\, P(j \mid i)\, P(S_{t+1} \mid j)\, \beta(i, t) / P(S) \text{ se } j \in E$$
$$\gamma(j, i, t) = \alpha(i, t)\, P(j \mid i)\, \beta(i, t) / P(S) \text{ se } j \in D.$$

Inoltre se marginaliziamo $\gamma(j, i, t)$ rispetto ad i, otteniamo $\gamma(j, t) = P(N_t = j \mid S)$:

$$\gamma(j,t) = P(N_t = j \mid S, w) = \sum_{i \in I} \gamma(j,i,t) .$$

Utilizzando questi coefficienti, possiamo calcolare alcuni parametri utili per gli algoritmi di apprendimento:

1. La probabilità $f(i) = P(i \mid S)$ che la sequenza passi attraverso il nodo i: tale probabilità si ottiene marginalizzando $\gamma(i, t)$ su t:

$$f(i) = P(i \mid S) = \sum_{t=0}^{T} \gamma(i,t) .$$

2. La probabilità $f(i, X) = P(i, X \mid S)$ che un simbolo X sia emesso dal nodo i (se i è un nodo di emissione) data la sequenza S: tale probabilità si ottiene marginalizzando $\gamma(i, t)\ P(S_t = X)$ su t:

$$f(i, X) = P(i, X \mid S) = \sum_{t=0}^{T} \gamma(i,t) P(S_t = X) .$$

Poiché $P(S_t = X)$ vale o 0 o 1 essendo nota la sequenza, questa marginalizzazione equivale a sommare solo quei $\gamma(i, t)$ per cui $S_t = X$ al tempo t. Ovviamente la marginalizzazione di $f(i, X)$ su X dà $f(i)$:

$$f(i) = \sum_{X \in S} f(i, X) .$$

3. La probabilità $f(j, i) = P(N_t = j, N_{t-1} = i \mid S)$ che la sequenza attraversi un arco: tale probabilità si ottiene marginalizzando $\gamma(j, i, t)$ su t:

$$f(j, i) = P(N_t = j, N_{t-1} = i \mid S) = \sum_{t=0}^{T} \gamma(j, i, t) .$$

Inoltre marginalizzando $f(j, i)$ su i si ottiene $f(j)$:

$$f(j) = \sum_{i \in G(j)} f(j, i) .$$

Il calcolo di tali coefficienti avviene nel seguente modo:

```
for (t = 0; t <= lungseq+1; t++)
{
   d = 0;

   // calcolo del denominatore della formula 7.13 pag. 153
   for(i = 0; i < nnodi; i++)
      d += parnodi[i].getAlfa(t)*parnodi[i].getBeta(t);

   // calcolo della formula 7.13 pag. 153
   for(i = 0; i < nnodi; i++)
   {
      n = parnodi[i].getAlfa(t)*parnodi[i].getBeta(t);
      if (n != 0)
         n /= d;
      parnodi[i].setGamma(t, n);
   …
   }
   …
}
```

La complessità di questo algoritmo è $O(NT)$: poiché T è direttamente proporzionale ad N, si ottiene $O(N^2)$. Per quanto riguarda il calcolo dei coefficienti $f(i)$ e $f(j, i)$, che non è altro che una semplice sommatoria sui $\gamma(i, t)$ e sui $\gamma(j, i, t)$, la complessità è dell'ordine di $O(T)$ e quindi non influisce sulla complessità globale dell'algoritmo.

4.2.3 Il percorso più probabile

Il percorso più probabile di una sequenza S lungo il grafo viene individuato in modo semplice: si cerca qual è il nodo più probabile ad ogni istante di tempo t. Tale nodo sarà sempre un nodo di emissione e mai un nodo di cancellazione. Infatti, se la probabilità di trovarsi al tempo t in un nodo di emissione MAX è p_{MAX} e questa è la massima probabilità considerando i soli nodi di emissione, si ha che la probabilità di trovarsi in un nodo di cancellazione al tempo t sarà sempre minore o uguale a p_{MAX}. Infatti nel caso migliore ci dovrà essere almeno una transizione tra il nodo MAX e il nodo di cancellazione $CANC$, il che porta la probabilità di trovarsi nel nodo di cancellazione $CANC$ a $p_{MAX} P(CANC \mid MAX)$ ove $P(CANC \mid MAX)$, che è una probabilità di transizione, è sempre minore o uguale a 1. Questo è vero per ogni istante di tempo t. Un modo per calcolare quale sia il nodo più probabile

al tempo t è quello di scegliere quello che massimizza $\gamma(i, t)$, fissato t. Infatti $\gamma(i, t)$ è proprio $P(N_t = i \mid S)$, cioè la probabilità di trovarsi nel nodo i al tempo t, data S. Un altro metodo è quello di utilizzare l'algoritmo di Viterbi.

4.2.4 L'algoritmo di Viterbi

Iniziamo col definire le seguenti variabili:

$$\delta(i,t) = \max_{\pi(i,t)} P(\pi(i,t) \mid w)$$

dove $\pi(i, t)$ rappresenta la parte iniziale di una sequenza $S_1, \ldots, S_t$ che termina nello stato i. Si ha che $\delta(i, t)$ rappresenta la probabilità associata al percorso più probabile dei primi t simboli della sequenza S che termina nello stato i. Queste variabili possono essere aggiornate usando un meccanismo di propagazione simile all'algoritmo forward, dove le sommatorie sono sostituite con delle massimizzazioni:

$$\delta(i,t) = \left(\max_{j \in G(i)} \delta(j, t-1) P(i \mid j) \right) P(S_t \mid i) \ \ se \ \ i \in E$$

$$\delta(i,t) = \max_{j \in G(i)} \delta(j,t) P(i \mid j) \ \ se \ \ i \in D .$$

Per ricostruire il percorso più probabile, si deve tenere traccia ad ogni tempo t del precedente stato ottimale.

4.2.5 Calcolo dell'allineamento e del percorso più probabile

L'allineamento delle sequenze è basato sul percorso più probabile delle medesime nel modello di Markov nascosto. Leggendo una sequenza, si evidenziano in un qualche modo tutti i simboli che, secondo il percorso più probabile, vengono emessi da uno stato main. Questo perché gli stati main tendono a contenere la parte non rumorosa della sequenza dove è quindi possibile rintracciare delle caratteristi-

che comuni. Si mostrerà più avanti negli esempi come questo accada effettivamente. Tuttavia non è possibile procedere all'utilizzo del percorso più probabile così come è stato definito in quanto non è necessariamente un percorso, cioè non è necessario che esista un cammino tra un nodo al tempo t - 1 ed un nodo al tempo t. In effetti è piuttosto l'elenco dei nodi più probabili nei vari istanti di tempo 1,…,T in quanto è semplice costruire un esempio per cui quello che viene chiamato "percorso più probabile" non sia affatto un cammino sul grafo. Per questo non possiamo scegliere il nodo più probabile al tempo t, come nodo di emissione del simbolo al tempo t, ma dobbiamo curarci del fatto che esista un cammino tra il nodo al tempo t - 1 e tale nodo. Si può procedere quindi nel modo seguente: si scandiscono tutti i nodi tenendo traccia del più probabile per il quale esista un cammino con il nodo al tempo t - 1. Si segnerà in un modo opportuno il simbolo della sequenza se il nodo così calcolato sia un nodo main o un nodo insert. Ecco la parte di programma che si occupa di individuare il percorso più probabile:

```
void TGrafo::PercorsoProbabile(int path[], int seq[], int T,
                                                        int quale)
{
   for (int t = 1; t <= T; ++t)
   {
      int nodoPiuProbabile = nnodi-1;
      double max = parnodi[nodoPiuProbabile].getGamma(t);

      for (int n = nnodi-2; n > 0; n--)
      if ((nodi[n].getTipo() != 2) && (n != nodoend))
      {
         int nodoSuccessivo = 1;

         if (t > 1)
         {
            if (nodi[path[t - 2]].getTipo() == 0)
            {
               if (nodi[n].getId() <= nodi[path[t - 2]].getId())
                  nodoSuccessivo = 0;
            }
            else if (nodi[path[t - 2]].getTipo() == 1)
            if (nodi[n].getId() < nodi[path[t-2]].getId()
               nodoSuccessivo = 0;
         }
```

```
            if (nodoSuccessivo && max < parnodi[n].getGamma(t))
            {
               max = parnodi[n].getGamma(t);
               nodoPiuProbabile = n;
            }
         }

         path[t-1] = nodoPiuProbabile;
      }
      …
   }
```

Come si vede il flag `nodoSuccessivo` controlla se il nodo abbia o meno un cammino con il nodo al tempo t - 1. Con le assunzioni fatte nel nostro modello, questa ricerca è lineare nel numero dei nodi in quanto l'esistenza del cammino per la particolare struttura può essere verificata attraverso due semplici controlli sul numero che identifica il nodo. In genere il controllo dell'esistenza di un cammino richiederà un tempo lineare nel numero dei nodi e dunque l'algoritmo generale avrà una complessità quadratica. Dunque la complessità totale dell'algoritmo in tempo è $O(NT)$ analoga a quella in spazio.

Dobbiamo notare inoltre che il percorso più probabile *dipende* dal quali coefficienti vengono massimizzati. Infatti se il punto di massimo dei coefficienti definiti dall'algoritmo di Viterbi corrisponde al punto di massimo dei coefficienti definiti dall'algoritmo generale, questo in generale non vale per l'intero vettore dei coefficienti: se da entrambi eliminiamo il punto di massimo comune, otteniamo due vettori con punti di massimo differenti. Dunque il percorso più probabile sarà differente a seconda dell'algoritmo con il quale viene calcolato: in particolare con Viterbi sarà meno accurato in quanto quei coefficienti contengono molta meno informazione, proprio per come sono stati definiti. D'ora in poi chiameremo il percorso più probabile o l'allineamento calcolato con l'algoritmo di Viterbi, "Viterbi" e quello calcolato con l'algoritmo generale "gradiente". Vedremo negli esempi come l'allineamento così come l'addestramento effettuato con Viterbi sia peggiore di quello con il gradiente.

A questo punto, disponendo di un vero e proprio percorso più probabile, possiamo calcolare l'allineamento che consiste soltanto nell'evidenziare i simboli emessi da nodi di tipo main. Sono necessarie anche altre procedure per aggiungere

spazi e rendere gradevole la visione dell'allineamento, ma che non influiscono in modo rilevante sulla complessità generale.

4.3 Gli algoritmi di apprendimento

Gli algoritmi di addestramento implementati per l'addestramento dell'HMM sono:

- l'algoritmo EM
- l'algoritmo di discesa lungo il gradiente
- l'algoritmo di Viterbi (variazione dell'algoritmo di discesa lungo il gradiente).

Viene trattata la stima ML, supponendo che le probabilità a priori siano uniformi. Nel caso in cui le sequenze di allineamento siano più di una, queste possono essere considerate indipendenti e la verosimiglianza totale è data dal prodotto delle verosimiglianze di tutte le sequenze.

4.3.1 L'algoritmo EM

Questo algoritmo si propone di adattare il modello all'insieme di sequenze considerato, massimizzando una certa funzione di energia. È un algoritmo di tipo batch in quanto esegue l'aggiornamento ad ogni epoca, dopo aver osservato tutte le sequenze. Il passo di aggiornamento dell'algoritmo comporta la sostituzione di certe probabilità con la loro stima ottenuta massimizzando l'energia libera del modello:

$$P(X \mid i) = f(X, i) / f(i)$$
$$P(j \mid i) = f(j, i) / f(i).$$

dove X è un simbolo della sequenza. Tali formule riescono ad adattare molto bene il modello alle sequenze ma purtroppo portano ad una cattiva generalizzazione in quanto provocano l'assorbimento del rumore presente nell'insieme di addestra-

mento. Quindi è bene utilizzare questo algoritmo solo in modalità batch e se non è necessaria un'elevata capacità di generalizzazione. In questi casi è meglio utilizzare l'algoritmo di discesa lungo il gradiente. L'aggiornamento delle variabili avviene nel seguente modo:

```
for (int n = 0; n < nnodi; ++n)
{
   for (int c = 0; c < cardalf; ++c)
      nodi[n].setProbEmis(c, eta *sommanix[n][c] / sommani[n]);

   for (c = 0; c < nodi[n].getNumFig(); c++)
   {
      int i = nodi[n].getArcFig(c);
      archi->aggiornaP(i, eta * sommanji[i] / sommani[n]);
   }
}
```

Vediamo che le probabilità di emissione e di transizione vengo semplicemente sostituite con le appropriate stime. La variabile `eta` che rappresenta il learning rate, viene sempre posta sempre ad 1 per questo algoritmo. La complessità dell'algoritmo EM è la stessa di quello di discesa lungo il gradiente, con le stesse costanti additive e moltiplicative.

4.3.2 L'algoritmo di discesa lungo il gradiente

Una possibile realizzazione dell'algoritmo di discesa lungo il gradiente può essere quella di massimizzare la verosimiglianza, o equivalentemente di minimizzare l'inverso del logaritmo della stessa. Si ha che la verosimiglianza di un percorso è data da:

$$P(S,\pi \mid w) = \prod_{Start}^{End} P(j \mid i) \prod_{t=1}^{T} P(S_t \mid i)$$

ove con π si è indicato un percorso qualsiasi. Ovviamente $P(S \mid w)$ si otterrà marginalizzando la $P(S, \pi \mid w)$:

$$P(S \mid w) = \sum_{\pi} P(S, \pi \mid w) .$$

Si ha che:

$$\frac{\partial P(S,\pi)}{\partial P(K|i)} = \frac{n(i,X,\pi,S)}{P(K|i)} P(S,\pi) \Rightarrow$$

$$\Rightarrow \frac{\partial \log P(S)}{\partial w_{iX}} = \sum_{K \in A} \frac{\partial \log \sum_{\pi} P(S,\pi)}{\partial P(K|i)} \frac{\partial P(K|i)}{\partial w_{iX}} =$$

$$= \sum_{K \in A} \frac{\sum_{\pi} n(i,X,\pi,S) P(\pi|S)}{P(K|i)} \frac{\partial P(K|i)}{\partial w_{iX}} =$$

$$= \sum_{K \in A} \frac{1}{\sum_{\pi} P(S,\pi)} \frac{\sum_{\pi} n(i,K,\pi,S) P(S,\pi)}{P(K|i)} \frac{\partial P(K|i)}{\partial w_{iX}} =$$

$$= \sum_{\pi} n(i,X,\pi,S) P(\pi \mid S) + \sum_{K \in A} \sum_{\pi} n(i,K,\pi,S) P(\pi \mid S) P(K \mid i)$$

dove A indica l'insieme dei simboli dell'alfabeto. La funzione $n(i, K, \pi, S)$ compare nel calcolo della derivata di $P(S,\pi)$.Infatti:

$$\frac{\partial P(S,\pi)}{\partial P(K|j)} P(K \mid j) = \frac{\partial \prod_{Start}^{End} P(j|i) \prod_{t=1}^{T} P(S_t|i)}{\partial P(K|j)} P(K \mid j) =$$

$$= n(i,K,\pi,S) \prod_{Start}^{End} P(j \mid i) \prod_{t=1}^{T} P(S_t \mid i)$$

in quanto non possiamo sapere in generale quanto vale $n(i, K, \pi, S)$, cioè quante volte si ripete $P(K \mid j)$ nella produttoria, poiché tale numero dipende per esempio dalla sequenza S. Possiamo interpretare $n(i, K, \pi, S)$ quindi come il numero di volte che il simbolo K viene emesso dal nodo i in un certo percorso π legato ad una sequenza S. Sommando su K possiamo calcolare $n(i, \pi, S)$. Moltiplicando gli $n(i, K, \pi, S)$ e gli $n(i, \pi, S)$ per la probabilità $P(\pi \mid S)$ di ciascun percorso π data la sequenza S e quindi marginalizzando sui percorsi π, otteniamo:

1. per gli $n(i, K, \pi, S)$ la probabilità $P(i, K \mid S)$ che è già stata calcolata nel coefficiente $f(i, K)$:

$$\sum_{\pi} n(i, K, \pi, S) P(\pi \mid S) = f(i, K)$$

2. per gli $n(i, \pi, S)$ la probabilità $P(i \mid S)$ che è già stata calcolata nel coefficiente $f(i)$:

$$\sum_{\pi} n(i, \pi, S) P(\pi \mid S) = f(i) .$$

A questo punto è utile introdurre una riparametrizzazione del modello, usando i seguenti esponenti normalizzati:

$$P(X \mid i) = \frac{e^{w_{iX}}}{\sum_{Y} e^{w_{iY}}} \qquad P(j \mid i) = \frac{e^{w_{ji}}}{\sum_{k} e^{w_{ki}}}$$

dove w_{iX} e w_{ji} sono le nuove variabili. Questa riparametrizzazione ha tre vantaggi:

- si mantengono automaticamente i vincoli di normalizzazione sulle emissioni e sulle transizioni;
- le probabilità di emissione e di transizione non possono mai raggiungere il valore 0;
- è invariante rispetto alle traslazioni.

Le formule di aggiornamento dell'algoritmo di discesa lungo il gradiente sono quindi le seguenti:

$$\Delta w_{iX} = \eta\big(f(i, X) - f(i) P(X \mid i)\big)$$

dove η indica il tasso di apprendimento o learning rate. Per le probabilità di transizioni vale la seguente formula che si ricava in modo del tutto simile:

$$\Delta w_{ji} = \eta\big(f(j,i) - f(i)P(j \mid i)\big).$$

Questo algoritmo può essere utilizzato sia in modalità on-line, sia in modalità batch. Nel nostro programma viene utilizzato sempre nella forma on-line. L'implementazione di tale algoritmo è la seguente:

```
for (i = 0; i < nnodi; i++)
{
   n1 = parnodi[i].getN();
   for (j = 0; j < cardalf; j++)
   {
      n2 = parnodi[i].getNX(j);
      prob = nodi[i].getProbEmis(j);
      delta = eta*(n2 - n1*prob);
      delta += nodi[i].getEmis(j);
      nodi[i].setEmis(j, delta);
   }
   nodi[i].calcProb();

   for (j = 0; j < nodi[i].getNumFig(); j++)
   {
      nar = nodi[i].getArcFig(j);
      n2 = pararchi[nar].getN();
      ar = archi->getArco(nar);
      prob = ar->ptrans;
      delta = eta*(n2 - n1*prob);
      archi->aggiorna(nar, delta);
   }
}
archi->calcPTrans();
```

La complessità di questo algoritmo è dell'ordine di $O(N)$ in quanto il numero di figli di ogni nodo, dato da `getNumFig()`, è costante e quindi viene assorbita dalle complessità per il calcolo degli $\alpha(i, t)$, $\beta(i, t)$ e $\gamma(i, t)$. Concludendo, l'addestramento tramite l'algoritmo di discesa lungo il gradiente di M sequenze richiede un tempo dell'ordine di $O(MN^2)$, lineare quindi nel numero delle sequenze come si voleva. Lo spazio richiesto è dell'ordine di $O(N^2)$, indipendentemente dal numero delle sequenze. Pagando una penalizzazione nel tempo non si ottiene co-

munque un miglioramento della complessità spaziale, se non sulle costanti additive e moltiplicative.

4.3.3 L'algoritmo di addestramento di Viterbi

Il precedente algoritmo di discesa lungo il gradiente utilizza tutti i percorsi relativi ad una sequenza per l'aggiornamento. L'idea generale dell'algoritmo di Viterbi è quella di sostituire i calcoli relativi a tutti i percorsi con dei calcoli che riguardano solo un piccolo numero di percorsi, tipicamente uno solo. In questa versione dell'algoritmo, si ignorano tutti i percorsi tranne il più probabile: questo porta ad una modifica abbastanza semplice dei coefficienti $f(i)$ ed $f(i, K)$ che non sono più delle probabilità, pur mantenendo il loro significato iniziale:

- $f(i)$ che indicava la probabilità di essere nel nodo i data la sequenza S, viene posto ad 1 nella versione proposta. Un'altra possibilità potrebbe essere quella di porlo ad 1 se il nodo fa parte in un qualche senso del percorso più probabile. È comunque vero che quasi tutti i nodi fanno parte del percorso più probabile e la distinzione viene fatta negli $f(i, K)$.
- $f(i, K)$ che indicava la probabilità di emettere il simbolo K dal nodo i, data la sequenza S, viene modificato nel seguente modo: vale 1 se il simbolo K viene emesso almeno una volta dal nodo i nel percorso più probabile; vale 0 altrimenti. Nella versione proposta non vale più che la marginalizzazione di $f(i, K)$ su K sia $f(i)$ in quanto può accadere che per ogni K tutti gli $f(i, K)$ valgano 0.
- $f(i, j)$ che indicava la probabilità di andare dal nodo i al j, data la sequenza S, viene sostituita in modo analogo alla precedente: vale 1 se il nodo fa parte del percorso più probabile, 0 altrimenti. Anche queste non sono più probabilità perché può accadere che dal nodo j gli archi uscenti che fanno parte del percorso più probabile siano 2 (per esempio se j è uno stato di inserimento, l'arco che lo riporta in se stesso e un altro arco uscente verso un main od un insert).

Apportando queste modifiche a tali coefficienti, l'algoritmo di aggiornamento è lo stesso del gradiente.

Risultati e Test

Per poter misurare quanto è allineato un insieme di sequenze, definiamo innanzitutto delle metriche che quantifichino il concetto di allineamento. Confronteremo poi i vari algoritmi di apprendimento su un insieme di sequenze di DNA e di proteine, con particolare riguardo al problema della generalizzazione.

5.1 Misure degli allineamenti

Prima di entrare nei dettagli delle prove è necessario definire alcuni strumenti con cui effettuare delle misure. In effetti cosa significa "allineamento"? Di questo è già stata data una definizione che tuttavia non può essere applicata in quanto il numero delle sequenze compare all'esponente nel calcolo della complessità, mentre nel nostro algoritmo compare come fattore moltiplicativo. Ovviamente ciò che si perde è proprio una misura rigorosa dell'allineamento: questo può in effetti essere calcolata anche a posteriori, verificando così che non sarà stata del tutto minimizzata, ma poiché anche la misura dipende molto dalla natura di ciò che vogliamo allineare, è interessante chiedersi se non si possa utilizzare una misura o un insieme di misure che prescindano dalla natura delle sequenze trattate: d'altra parte un essere umano ha un chiaro concetto qualitativo di allineamento e quindi ha senso chiedersi come tradurre in numero questo concetto.

Una prima possibilità è quella di contare in ogni colonna come si ripartiscono in percentuale i vari simboli dell'alfabeto. In questo modo possiamo dividere in due classi le colonne prodotte dell'allineamento: quelle allineate e quelle non

allineate dove si considerano allineate tutte quelle colonne in cui esiste un simbolo *X* che appare con una percentuale molto elevata, per esempio il 95%. Questo perché alla comune esperienza appare ragionevole chiamare 'allineata' una colonna in cui compare sempre lo stesso simbolo. In genere si dirà che una colonna è allineata al tot% se esiste un simbolo che compare almeno con quella frequenza.

Questa divisione è tuttavia abbastanza drastica e si potrebbe anche pensare che esistano dei casi in cui questo numero è sempre 0 perché le colonne sono allineate solo al 94% dunque appare naturale definire una seconda misura che tenga conto di questa sfumatura. Ogni colonna contribuisce alla misura con:

- 0,25 se la colonna è allineata allo 80%;
- 0,5 se la colonna è allineata al 90%;
- 1 se la colonna è allineata al 95%.

Sotto lo 80% possiamo ragionevolmente affermare che la colonna non è affatto allineata in quanto su 5 simboli ce ne è uno differente.

Un'altra possibilità, del tutto differente, per misurare l'allineamento è quello di fare riferimento ad una ipotetica (o reale) sequenza più probabile (o più comune). Definiamo la sequenza più probabile in un insieme di sequenze allineate in tante colonne nel seguente modo: il simbolo *i*-esimo della sequenza più probabile è il simbolo più probabile della colonna *i*-esima. Non va confuso questo con il percorso più probabile. A questo punto, avendo un parametro di confronto, definiamo la seguente misura: ogni simbolo della colonna *i*-esima contribuisce con +1 alla misura se è differente dal simbolo omologo della sequenza più probabile, con 0 altrimenti. Quindi se tutte le sequenze sono uguali, questa misura vale 0, se le sequenze sono casuali la misura sarà massima. In particolare la media di questo valore, se i simboli sono *M*, tutti equiprobabili, su *K* sequenze di lunghezza *N* sarà con probabilità 1, quando *K* tende a infinito, $(1-1/M)N$, ovvero dividendo per il numero di colonne in modo da avere un valore normalizzato $(1-1/M)$.

Questa misura tuttavia non fa differenza tra colonne allineate e colonne non allineate: infatti non dovrebbe avere molta influenza il fatto che sia diverso un simbolo in una colonna allineata solo al 20%. Quindi si potrebbe procedere ad una misura simile alla precedente che tuttavia pesi l'allineamento delle colonne. In particolare ogni simbolo contribuisce alla misura:

- +8 se la colonna è allineata al 97%;
- +4 se la colonna è allineata al 95%;
- +2 se la colonna è allineata al 90%;
- +1 se la colonna è allineata allo 80%;
- +0,5 se la colonna è allineata al 60%.

Quindi un simbolo diverso dal più probabile in una colonna allineata quasi perfettamente viene pesato 16 volte di più che non in una colonna allineata solo al 60% (quindi con poco allineamento). Questa misura vale 0 sia per le colonne perfettamente allineate sia per quelle che non lo sono affatto: questo può fornire una utile indicazione. Infatti questa misura man mano che l'algoritmo produce allineamenti migliori tenderà prima a salire, in quanto cominciano ad esserci alcune colonne allineate, quindi a diminuire. Si dovrà fare attenzione al fatto che questa diminuzione non indichi un overfitting.

Per ognuna di queste misure è inoltre interessante considerare, oltre al valore medio, anche il valore massimo ed il numero di sequenze comprese tra la (media + massimo) / 2 ed il massimo che rappresentano il numero di sequenze che non sono state allineate, qualora la differenza tra massimo e media sia rilevante; altrimenti è un numero abbastanza casuale. È importante osservare come nessuna di queste misure possa funzionare da sola:

- il numero di colonne allineate, anche pesato, non ci dice nulla sull'allineamento delle singole sequenze;
- il numero di simboli differenti dalla sequenza più probabile non ci dice nulla su quanto sia lo scostamento dalle colonne allineate;
- la misura dello scostamento dalle colonne allineate, in quanto vale 0 sia per sequenze perfettamente allineate sia per sequenze poco allineate non ci dice se quel numero è piccolo perché ci sono poche colonne allineate o perché l'allineamento è buono.

Un buon metodo è quello di considerare l'andamento di tali misure: è atteso crescente per la prima, decrescente per la seconda e prima crescente, poi decrescente per la terza. In particolare ad una diminuzione della seconda misura, è un fatto po-

sitivo l'aumento della terza in quanto segno che le colonne si stanno allineando e dovrebbe corrispondere anche ad un aumento della prima.

5.2 Test

Sono stati effettuati dei test per misurare la bontà degli allineamenti, secondo le misure appena date. Abbiamo utilizzato due insiemi di addestramento differenti: uno inventato ad hoc che utilizza i simboli del DNA come alfabeto ed un altro che utilizza un insieme di immunoglobuline. Per ognuno di questi insiemi sono stati effettuati vari addestramenti utilizzando solo una parte, da 1/3 a 2/3, dell'insieme di sequenze in modo da poter verificare la capacità di generalizzare in quanto l'allineamento poi è stato sempre calcolato su tutte le sequenze. Abbiamo in questo modo operato una sorta di validazione incrociata.

5.2.1 Esempi di allineamento

Questi test vogliono, tra l'altro, chiarire il significato delle misure proposte e dell'allineamento. L'insieme delle sequenze considerato è il seguente:

```
AAACTGTTGGGCCCC
AAACTTTGTTGGCCCC
AAACTTTGGGCCACC
AACTTTGGGCCCC
AAACTTTGTTGGGCCCC
AAACTTTGGAAGCCACC
AATGACTTTGGGCCCC
AAACTTTGGGCCCC
AAACTTTGACGGCCCC
AACACTTTGGGCCCC
AACTTTGGCCC
AAACTTTGGGCGGCCC
AGAACTTTGGGCCCC
AAACTTTGGGCCATGCC
AAACTATTGGGCCCC
AAACTTTGGGCCCC
AAAGCTTTGGGCCCC
AAACTTTGGGCCCC
AAACTTTGGGCCCC
AAATCTTGGGCCCC
AAACTTTGGGCCCC
AAACTTTGGGCCCCGGGTTTTAAACTTTGGGCCCCGGGTTTT
```

Come si vede sono tutte “variazioni sul tema” che è AAACTTTGGGCCCC che tolgono, spostano o aggiungono uno o più simboli. Da questo insieme si riesce ad ottenere il seguente allineamento effettuato con Viterbi utilizzando per l’addestramento lo stesso insieme di sequenze:

```
A A  A CTgTTG  G  G C    CC   C
A A  A CT TTGttG  G C    CC   C
A A  A CT TTG  G  G C    CA   Cc
A Ac    T TTG  G  G C    CC   C
A A  A CT TTGttG  GGC    CC   C
A A  A CT TTG  GaaG C    CA   Cc
A AtgA CT TTG  G  G C    CC   C
A A  A CT TTG  G  G C    CC   C
A A  A CT TTGacG  G C    CC   C
A Ac A CT TTG  G  G C    CC   C
A Ac    T TTG  G    C    CC
A A  A CT TTG  G  G Cggcc     C
AgA  A CT TTG  G  G C    CC   C
A A  A CT TTG  G  G C    CAtgcC
A A  A CTaTTG  G  G C    CC   C
A A  A CT TTG  G  G C    CC   C
A A  AgCT TTG  G  G C    CC   C
A A  A CT TTG  G  G C    CC   C
A A  A CT TTG  G  G C    CC   C
A A  AtCT T G  G  G C    CC   C
A A  A CT TTG  G  G C    CC   C
A A  A CT TTG  G  G C    CC   Cgggttttaaactttgggccccgggtttt
```

Questo è l’allineamento vero e proprio: notiamo che in maiuscolo sono stati scritti i simboli che nel percorso più probabile vengono emessi da nodi di tipo main, mentre in minuscolo quelli emessi da nodi di tipo insert che rappresentano il rumore. Come si può vedere, l’algoritmo è riuscito ad eliminare quasi completamente il rumore introdotto. Oltre all’allineamento vengono anche stampate le informazioni sulle misure sopra introdotte:

```
Sequenza più probabile : AAAC-TTTGGG-CCCC
```

Questa è la sequenza più probabile secondo la definizione data: notiamo che è proprio il “tema” comune. Le lineette ‘-‘ indicano che quello stato ‘main’ generalmente non emette alcun simbolo.

```
Misura 1 0.590909 / lungmod = 0.03693
```

```
Misura 1 max 4.00000 #seq comprese tra 2.29545 e 4.00000 = 3
```

Questa è la misura del numero di simboli differenti medio, normalizzato e massimo. Notiamo che il valore normalizzato è bassissimo. Il numero di colonne considerate non allineate 3 dovrebbe essere casuale, ma non lo è troppo: per esempio le sequenze 4 e 6 appaiono non allineate.

```
Misura 3 1.454545 / lungmod = 0.09091
Misura 3 max 12.00000 #seq comprese tra 6.72727 e 12.00000 = 3
```

Questa è la misura dello scostamento dalle colonne allineate. Le 3 colonne considerate non allineate sono le stesse individuate dall'altra misura.

```
Misura 4a 13 / lungmod = 0.81250
Misura 4b 14 / lungmod = 0.89062
```

Questo sono le misure sul numero di colonne allineate al 95% con di seguito la normalizzazione. La prima misura è quella non pesata, la seconda quella pesata.

```
Sequenza prob. >95% : AA**-TTTGGG-CC*C
```

Infine questo riporta la stessa sequenza più probabile eliminando con degli * quelle colonne che non sono allineate al 95%. L'allineamento con il gradiente è il seguente:

```
A A  A CTgTTG   G  G C    CC                              C
A A  A CT TTGtt G  G C    CC                              C
A A  A CT TTG   G  GCC    AC                              C
A A    CT TTG   G  G C    CC                              C
A A  A CT TTGttgG  G C    CC                              C
A A  A CT TTG   GaaGCC    AC                              C
A AtgA CT TTG   G  G C    CC                              C
A A  A CT TTG   G  G C    CC                              C
A A  A CT TTGac G  G C    CC                              C
A Ac A CT TTG   G  G C    CC                              C
A A    CT TTG      G C     C                              C
A A  A CT TTG   G  G Cgg  CC                              C
AgA  A CT TTG   G  G C    CC                              C
A A  A CT TTG   G  G Ccatg C                              C
A A  A CTaTTG   G  G C    CC                              C
A A  A CT TTG   G  G C    CC                              C
```

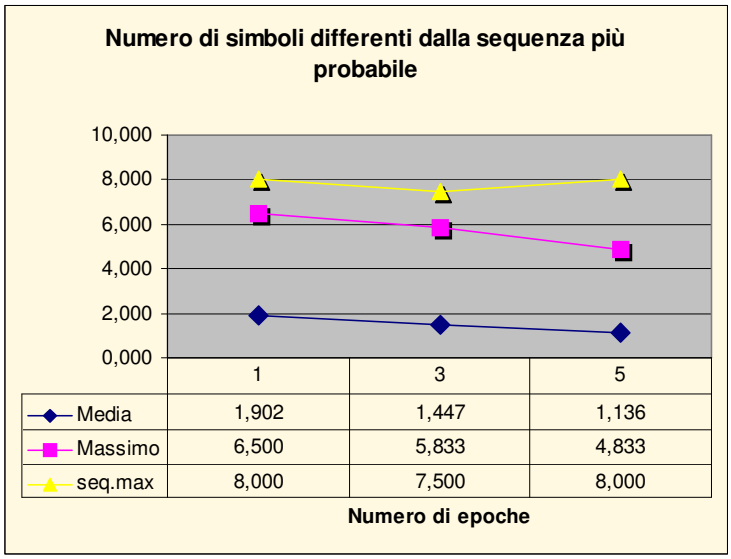
```
A A  AgCT TTG   G  G C    CC                                C
A A  A CT TTG   G  G C    CC                                C
A A  A CT TTG   G  G C    CC                                C
A A  AtC  TTG   G  G C    CC                                C
A A  A CT TTG   G  G C    CC                                C
A A  A CT TTG   G  G C    CCcgggttttaaactttgggccccgggtttt

Sequenza più probabile : AAAC-TTTGGG-CCCC

Misura 1 0.500000 / lungmod = 0.03125
                  / lungmod / cardalf  0.00781
Misura 1 max 3.00000 #seq comprese tra 1.75000 e 3.00000 = 9
Misura 3 1.090909 / lungmod = 0.06818
                  / lungmod / cardalf  0.01705
Misura 3 max 7.00000 #seq comprese tra 4.04545 e 7.00000 = 3
Misura 4a 13 / lungmod = 0.81250
Misura 4b 14 / lungmod = 0.89062

Sequenza prob. >95% : AA*C-TTTGGG*C*CC
```

Come si vede i risultati ottenuti sono leggermente migliori, ma tutto sommato simili, com'era giusto visto che il modello di partenza era lo stesso per entrambi gli allineamenti.

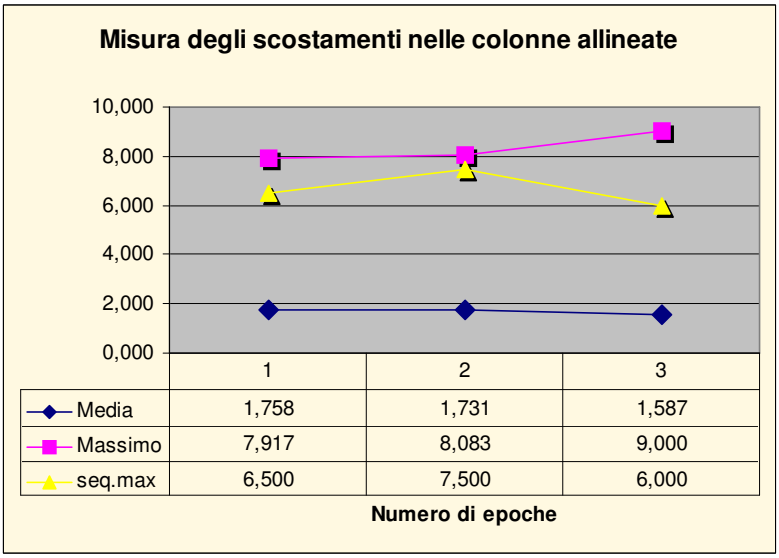
5.2.2 Test sul DNA

Innanzitutto vediamo i test effettuati su un campione inventato con l'alfabeto del DNA. Il modello è stato addestrato con un insieme di sequenze più piccolo di quello su cui poi è stato effettuato l'allineamento in modo, da effettuare una specie di validazione incrociata, osservando se vengono allineate correttamente anche le sequenze su cui il modello non è stato direttamente addestrato. Sono stati effettuati addestramenti con tutti gli algoritmi e su un numero differente di epoche.
Nei grafici si indicherà con:

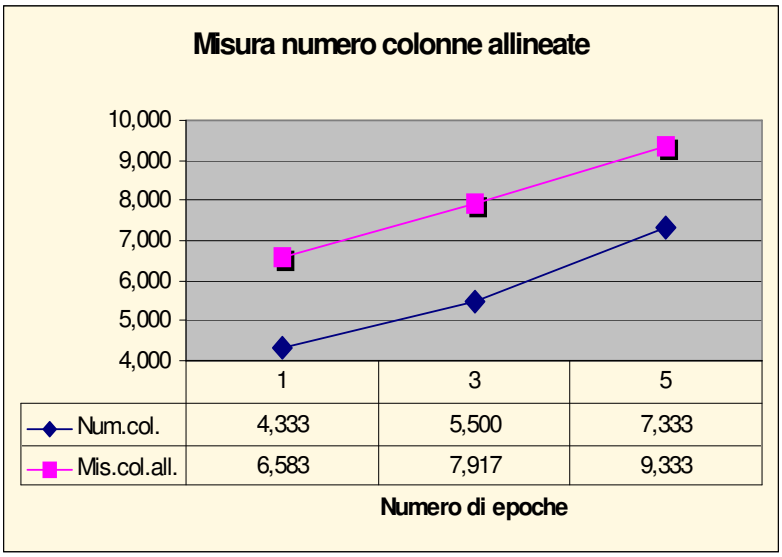
- Media: la media su tutti gli algoritmi di addestramento, di allineamento e su tutte le sequenze.
- Massimo: media dei valori massimi registrati su tutti gli algoritmi di addestramento, di allineamento e su tutte le sequenze.
- Seq.Max. : il numero di sequenze tra (media + massimo) / 2 e il massimo.



Questo grafico è relativo all'andamento, durante l'addestramento, della misura indicata nella sua media, nel suo valore massimo e nel numero di sequenze che hanno una misura alta. I valori non sono normalizzati e sono relativi ad una media effettuata su tutti gli addestramenti dopo 1, 3 e 5 epoche. Notiamo che la media ed il valore massimo diminuiscono, mentre sul numero di sequenze a valore alto c'è un andamento non uniforme: questo è dovuto soprattutto al fatto che questo allineamento è particolarmente semplice per cui si avrebbe già un ottimo allineamento anche solo dopo 3 epoche. Nell'esempio precedente erano state usate solo 3 epoche.

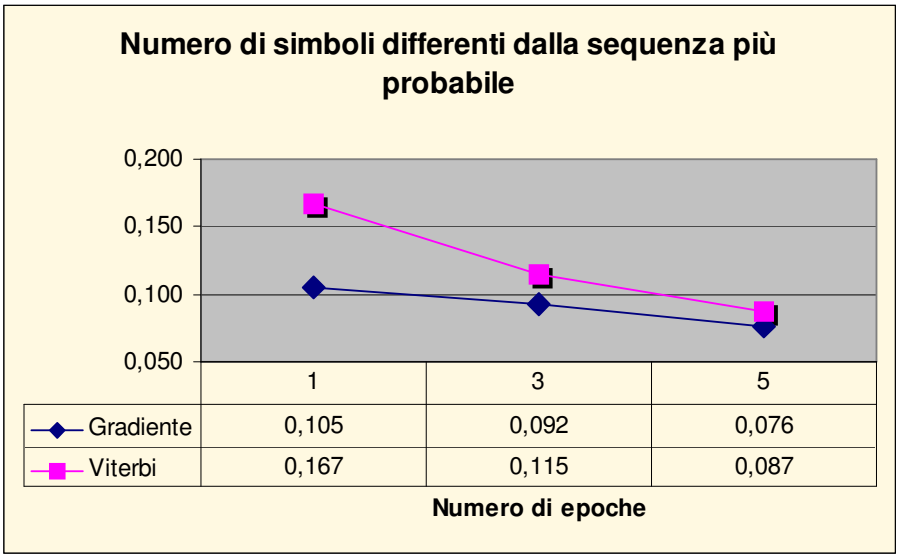


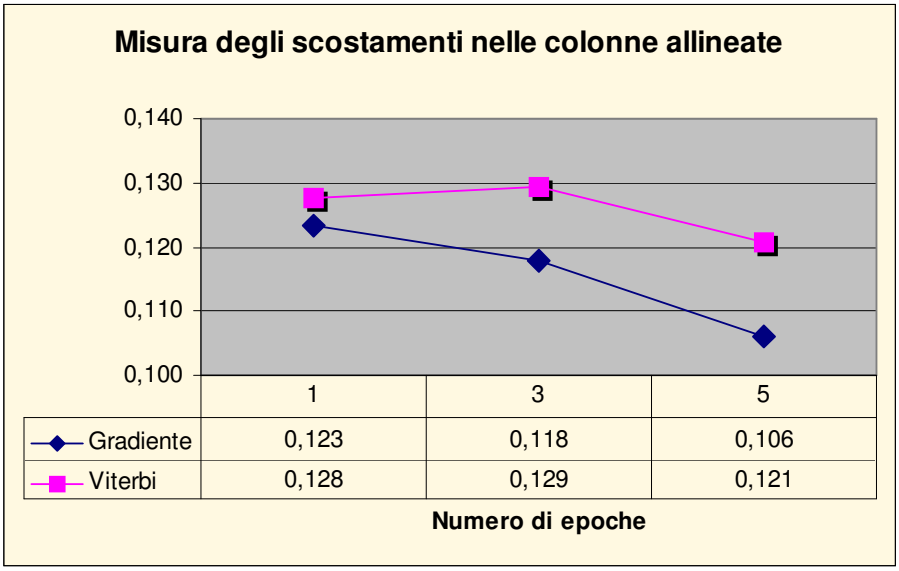
Qui l'andamento sembra abbastanza casuale: la media diminuisce leggermente, il valore massimo aumenta. Questo in realtà è perfettamente accettabile in quanto, per come è definita la misura, è ragionevole attendersi che la media diminuisca in quanto l'allineamento migliora, mentre il valore massimo può anche salire perché mentre l'allineamento migliora e con questo la misura media, la penalizzazione per una sequenza che viene mal allineata aumenta.



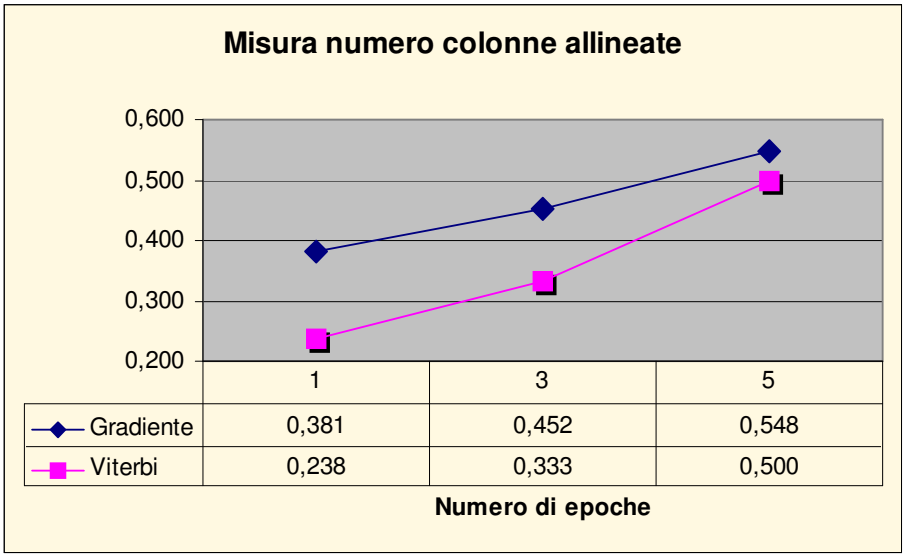
Vediamo che le due misure sul numero di colonne allineate aumentano concordemente. Questa caratteristica si ripete quasi sempre.

Questi grafici si limitano ad indicare il funzionamento delle misure e degli algoritmi nella media, ma non permettono di cogliere le differenze tra i vari algoritmi di addestramento e/o allineamento. Difatti un allineamento fatto con il gradiente richiede un tempo circa 4 volte maggiore di uno fatto con Viterbi: infatti solo il primo utilizza tutte l'informazione contenuta nel grafo. Da questo e dalle considerazioni fatte prima ci attendiamo un funzionamento migliore dell'algoritmo a gradiente. Vediamo che il gradiente riesce ad ottenere risultati migliori anche su un numero basso di epoche, mentre Viterbi necessita di un numero di epoche maggiore.





Anche da qui possiamo trarre conclusioni analoghe a quelle del grafico precedente: infatti l'andamento di Viterbi è a parabola, mentre l'altro è scende costantemente. L'andamento a parabola è indicativo del fatto che c'è stato bisogno di un numero di epoche maggiore per pervenire ad un allineamento buono, in quanto dopo una sola epoca Viterbi, in media, non è stato in grado di riuscire in questo. Il gradiente invece non ha avuto questo problema come dimostra l'andamento sempre decrescente della misura: segno che già dopo un'epoca si è oltrepassato il punto in cui questa misura ha il suo massimo.

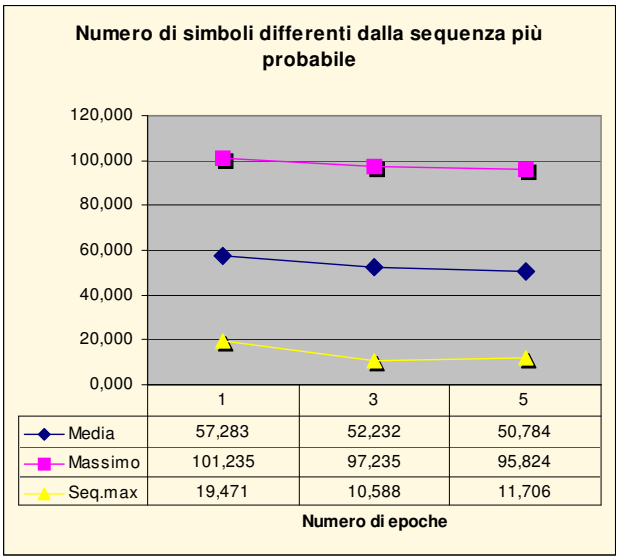


Anche questo conferma quanto detto prima: si può vedere che su un numero alto di epoche i due algoritmi di allineamento si comportano allo stesso modo com'è giusto, prima che intervengano fenomeni di sovraddestramento che qui non sono apparsi. Il numero di colonne allineate è un buon indicatore del sovraddestramento qualora ricominci a scendere: significa che le sequenze fuori dall'insieme di addestramento non vengono più allineate: poiché sono in questo esempio ed in tutti gli altri tra 1/3 e 2/3 dell'insieme totale delle sequenze il loro non allineamento porta ad un immediato peggioramento di questa misura. Si può notare un sovraddestramento soprattutto utilizzando un addestramento EM.

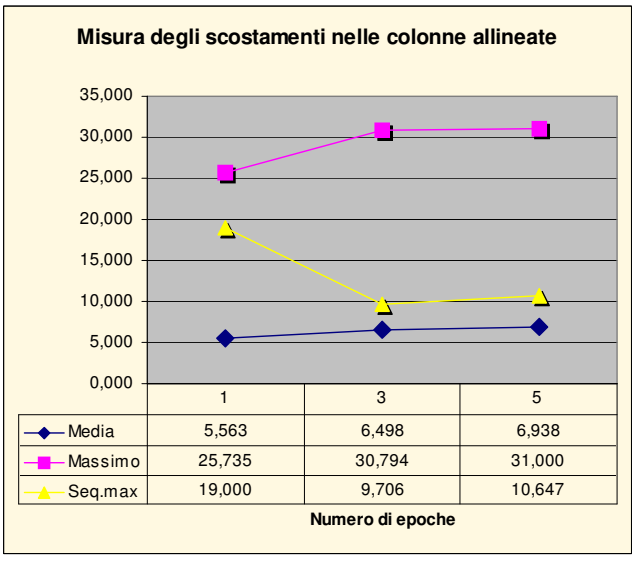
5.2.3 Test sulle proteine

Un test analogo a quello precedente è stato fatto su un insieme di sequenze che hanno un significato biologico. Qui infatti abbiamo utilizzato delle immunoglobuline di diversa provenienza biologica: queste possiedono, per motivi chimici e fisici, almeno due colonne che devono allinearsi. L'insieme di 150 immunoglobuline è stato diviso in più file ognuno con un numero di proteine tra 70 e 110 su cui sono stati fatti gli addestramenti con tutti gli algoritmi. Quindi sono stati fatti gli allineamenti sull'intero insieme. Notiamo che perché una colonna sia allineata al 95% possono esserci solo 5 simboli differenti lungo quella colonna e gli altri 145 devono esseri corretti: quindi se una colonna viene allineata il modello è riuscito a generalizzare, in quanto è pressoché impossibile che più di trenta simboli, scelti in

una rosa di 20, siano casualmente gli stessi. Come prima innanzitutto vediamo i grafici che riassumo l'intero addestramento.

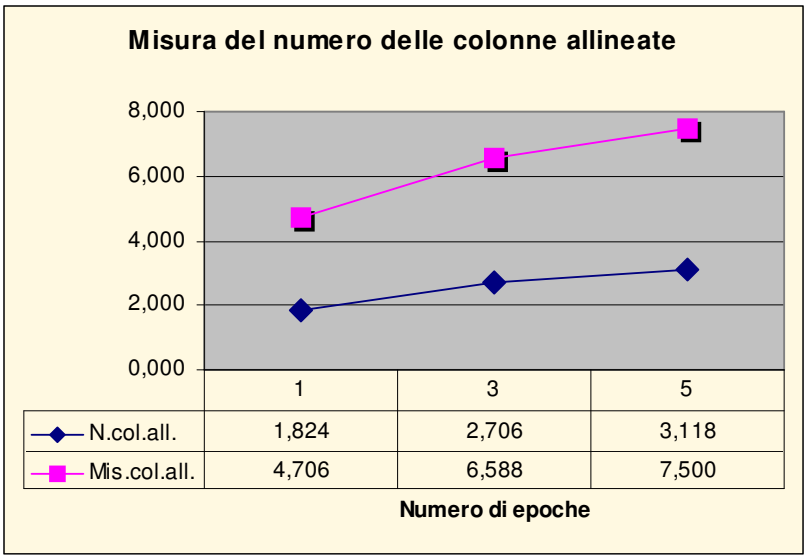


Notiamo che in media c'è una leggera diminuzione di questa misura sia in media sia nel suo valore massimo.

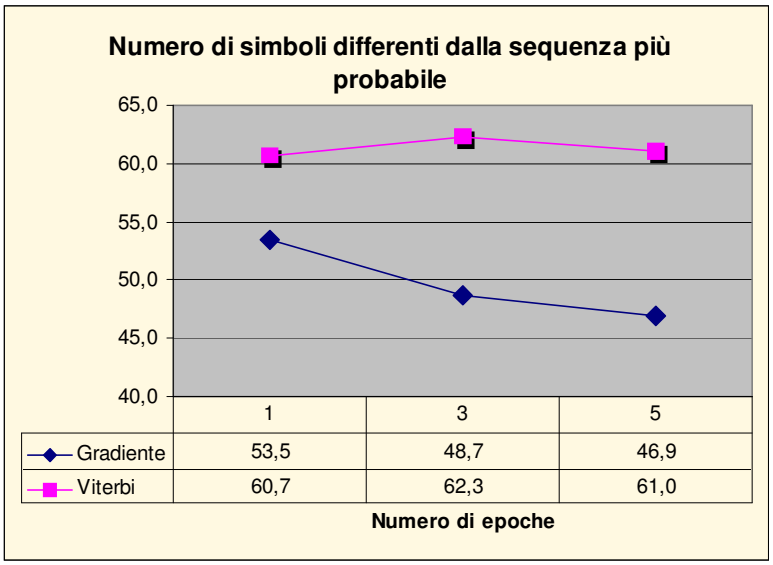


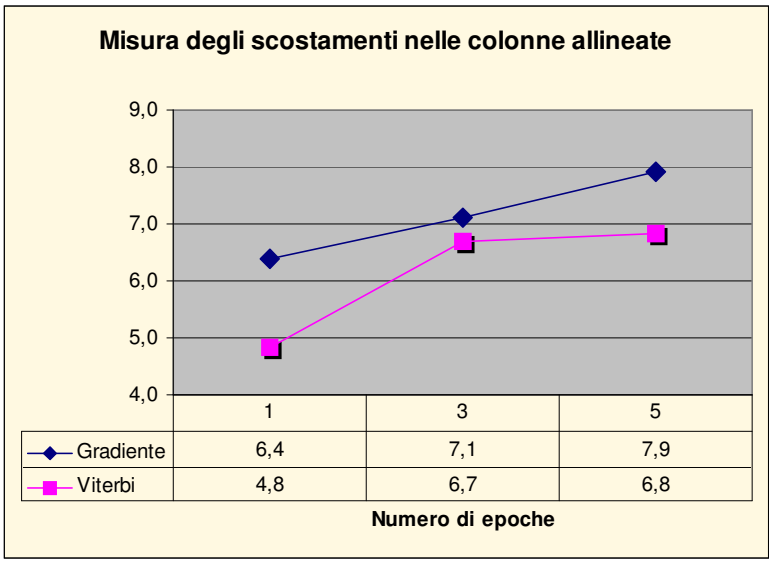
Qui notiamo che questa misura cresce anche se tende a stabilizzarsi dopo 5 epoche almeno nella media: questo può indicare che dopo 5 epoche la misura abbia raggiunto il valore massimo e possa d'ora in poi solo migliorare o andare in sovraddestramento. In realtà sappiamo che sarà comunque un sovraddestramento in quanto la proprietà biologica prima discussa viene evidenziata dopo 3 epoche con

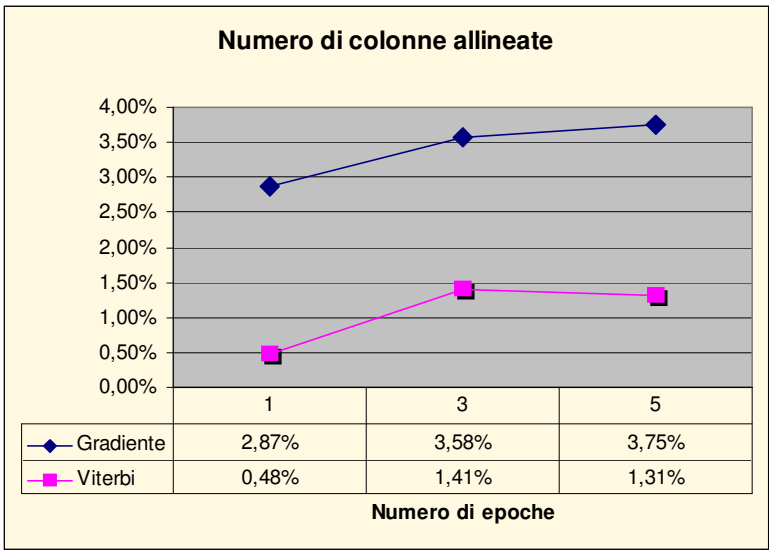
Viterbi e con il gradiente e dopo 5 con EM. Ulteriori allineamenti, interessanti come studio degli allineamenti, non hanno una rilevanza biologica.



Anche questo grafico è concorde con gli altri: questo sottolinea anche la coerenza delle misure adottate e la loro forte correlazione. Notiamo che questi risultati, con eccezione della misura degli scostamenti, abbiamo ottenuto risultati il cui andamento è analogo a quello dell'esempio inventato precedentemente: un altro segno dell'efficacia delle misure considerate. A questo punto è utile considerare i risultati differenziati per algoritmo di allineamento.


Numero di simboli differenti dalla sequenza più probabile
65,0
60,0
55,0
50,0
45,0
40,0
1
3
5
Gradiente
53,5
48,7
46,9
Viterbi
60,7
62,3
61,0
Numero di epoche

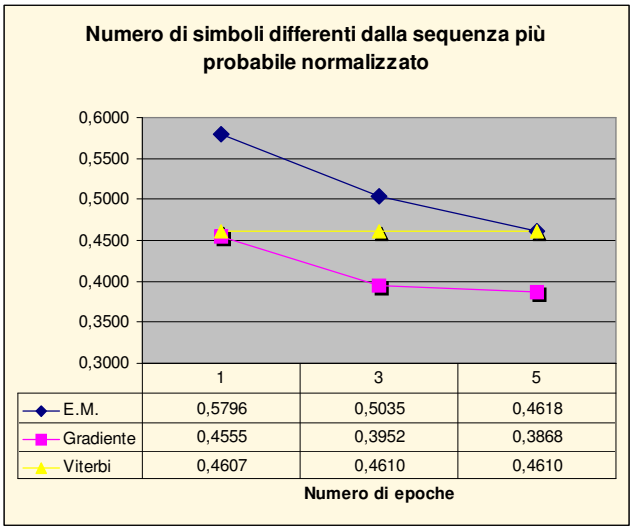

Misura degli scostamenti nelle colonne allineate
9,0
8,0
7,0
6,0
5,0
4,0
1
3
5
Gradiente
6,4
7,1
7,9
Viterbi
4,8
6,7
6,8
Numero di epoche

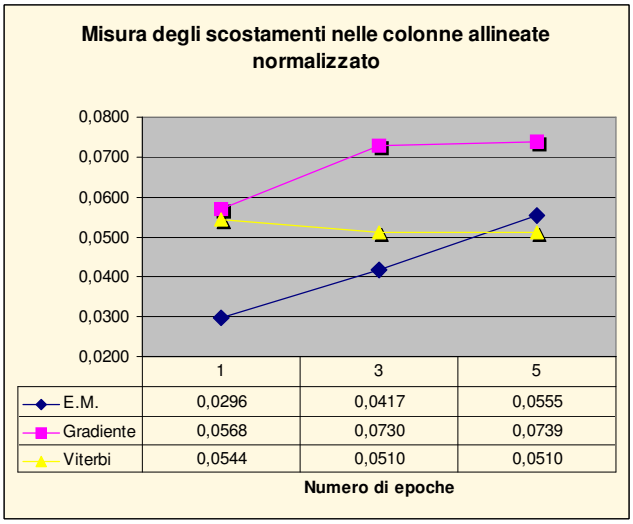

Numero di colonne allineate
4,00%
3,50%
3,00%
2,50%
2,00%
1,50%
1,00%
0,50%
0,00%
1
3
5
Gradiente
2,87%
3,58%
3,75%
Viterbi
0,48%
1,41%
1,31%
Numero di epoche

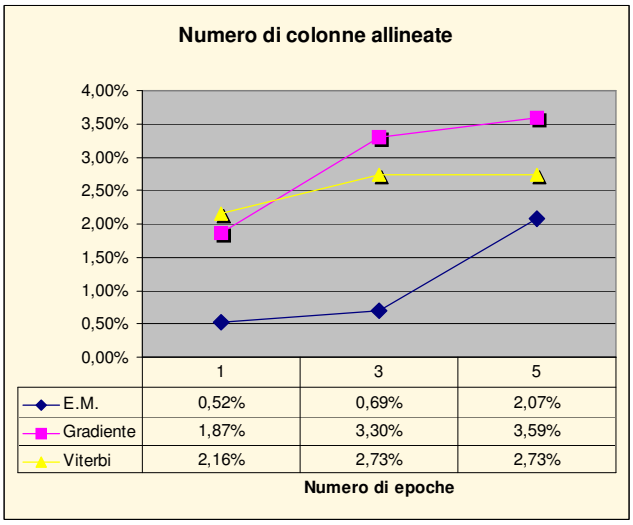
Notiamo che l'algoritmo di Viterbi ha un comportamento anomalo, mentre quello del gradiente è molto più netto nei suoi andamenti: questo è indicativo di come il primo riesca a estrarre in modo peggiore informazioni dal modello poco addestrato, mentre ci riesce molto meglio il secondo.

Un ultimo confronto interessante è tra i vari algoritmi di addestramento.
I grafici della pagina successiva rendono l'idea del funzionamento dei tre algoritmi:

- L'algoritmo che sfrutta tutte le informazioni presenti per l'aggiornamento, chiamato "gradiente" è indubbiamente il migliore, come si evidenzia in tutte e tre le misure.
- L'algoritmo di Viterbi per l'aggiornamento ha un comportamento abbastanza singolare in quanto sembra abbastanza indipendente dal numero di epoche seppure stabilizzato su dei valori del tutto inaccettabili. In realtà questo è dovuto ad un andamento del tutto irregolare nei vari esempi per cui si sono registrati in certi casi aumenti ed in certi altri diminuzioni. Come si era già detto non è un buon algoritmo per l'aggiornamento in quanto considera solo una parte del modello. Peraltro si presta molto male a generalizzare in quanto è stato sempre aggiornato lungo i percorsi più probabili del solo insieme di addestramento.
- L'algoritmo EM ha una non buona velocità di convergenza su poche epoche, mentre su un numero alto di epoche tende ad avviarsi verso gli stessi valori del gradiente. In realtà su un numero alto di epoche è ancora più veloce, ma tende ad adattarsi troppo all'insieme di addestramento e ad andare in sovraddestramento facilmente. Ad esempio, sull'insieme di sequenze di DNA, l'algoritmo EM va in sovraddestramento dopo 25 epoche imparando in modo pressoché perfetto l'insieme di addestramento, mentre il gradiente anche dopo 150 epoche, pur in presenza di risultati ottimi di allineamento già dopo 10 epoche, continua a funzionare bene.


Numero di simboli differenti dalla sequenza più probabile normalizzato
0,6000
0,5500
0,5000
0,4500
0,4000
0,3500
0,3000
1
3
5
E.M. 0,5796 0,5035 0,4618
Gradiente 0,4555 0,3952 0,3868
Viterbi 0,4607 0,4610 0,4610
Numero di epoche


Misura degli scostamenti nelle colonne allineate normalizzato
0,0800
0,0700
0,0600
0,0500
0,0400
0,0300
0,0200
1
3
5
E.M. 0,0296 0,0417 0,0555
Gradiente 0,0568 0,0730 0,0739
Viterbi 0,0544 0,0510 0,0510
Numero di epoche


Numero di colonne allineate
4,00%
3,50%
3,00%
2,50%
2,00%
1,50%
1,00%
0,50%
0,00%
1
3
5
E.M. 0,52% 0,69% 2,07%
Gradiente 1,87% 3,30% 3,59%
Viterbi 2,16% 2,73% 2,73%
Numero di epoche

UNIVERSITA' DEGLI STUDI DI FIRENZE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

Esercitazioni per l'esame di
"Intelligenza Artificiale"
Prof. G. Soda

*WIDA**

di Sauro Menchetti

A.A.1998-99

Introduzione

L'algoritmo WIDA* (Weighted Iterative Deepening A*) è una modifica dell'algoritmo di ricerca IDA* in cui il peso relativo di g e h viene controllato ponendo:

$$f(n) = g(n) + w\ h(n)$$

dove $w \geq 0$ è un peso associato alla funzione h. Grandi valori di w enfatizzano il potere euristico, mentre piccoli valori di w enfatizzano il carattere Breadth First dell'algoritmo. L'idea che porta all'introduzione del peso w, è quella di riuscire a trovare la soluzione in modo più veloce: questo porta purtroppo a dover rinunciare all'ottimalità dell'algoritmo nel caso in cui l'euristica h sia ammissibile.

1.1 Caratteristiche generali dell'algoritmo WIDA*

L'algoritmo di ricerca WIDA* esegue gli stessi passi dell'algoritmo IDA*; l'unica differenza riguarda la funzione $f(n)$:

$$f(n) = g(n) + h(n) \text{ in IDA*}$$
$$f(n) = g(n) + w\ h(n) \text{ in WIDA*}$$

dove $f(n)$ rappresenta una stima del costo del percorso vincolato a passare dal nodo n, $g(n)$ rappresenta un valore esatto (distanza dalla radice al nodo n) e $h(n)$ rappresenta una stima della distanza dal goal (è la funzione euristica). In generale, il fattore di correzione w fa perdere l'ammissibilità all'algoritmo di ricerca che quindi non garantisce più di trovare la soluzione ottimale del problema.

1.1.1 L'algoritmo WIDA*

L'algoritmo può essere riassunto nei seguenti passi, in cui la funzione euristica indica i nodi da scegliere all'interno di una ricerca Depth First:

1) *Inizializza il valore di taglio C;*
2) *Poni nello stack S i nodi iniziali;*
3) *Se lo stack S è vuoto, incrementa il valore di taglio C e vai al passo 2), altrimenti sia n il primo nodo di S;*
4) *Se n è un nodo goal, stop e restituisci il percorso dal nodo iniziale a n;*
5) *Altrimenti rimuovi n da S, poni sul top dello stack S ogni figlio m di n per cui $f(m) \leq C$ e ritorna al passo 3).*

1.2 Implementazione dell'algoritmo WIDA*

L'algoritmo illustrato lascia alcuni margini di scelta quando si passa all'implementazione: ad esempio, in che modo deve essere incrementato il valore di taglio *C* e in che ordine i figli di un nodo *n* devono essere inseriti sul top dello stack *S*. Un'altra questione aperta riguarda come evitare di generare degli stati già generati in precedenza per evitare eventuali cicli infiniti.

1.2.1 Evitare la ripetizione degli stati

La tecnica usata per evitare la ripetizione degli stati implementa un metodo per non creare cammini che abbiano cicli. La funzione di espansione (o l'insieme degli operatori) deve rifiutare di generare qualsiasi successore del nodo che coincida con un suo qualsiasi antenato (non si fa un controllo su tutti i nodi generati in precedenza ma solo sugli antenati per avere un utilizzo lineare e non esponenziale della memoria). Questo viene realizzato memorizzando i nodi che costituiscono il cammino dalla radice al nodo corrente in uno stack *T* ed usando un insieme di riferimenti tra lo stack *S* e lo stack *T*. L'occupazione di spazio rimane così lineare.

1.2.2 Incremento del valore di taglio

Un modo semplice ma non molto efficiente per incrementare il valore di taglio è quello di aumentarlo ad ogni ripartenza di un valore costante fissato a priori, ad esempio uno. Esiste un altro modo di più difficile implementazione ma di maggiore efficacia: si incrementa il valore di taglio di una quantità dipendente dalla ripartenza a cui siamo giunti. Per determinare questa quantità, si fa uso di una variabile ausiliaria `min_cut_off` che contiene, per ogni ripartenza, il valore della funzione *f(n)* più piccolo fra tutti quelli generati superiore al valore di taglio corrente. Osservando poi che un goal avrà sempre un valore di *f(n)* intero, si incrementa il valore di taglio della quantità `min_cut_off` arrotondata all'intero immediatamente superiore. Questo accorgimento permette di evitare inutili ripartenze e quindi di diminuire il numero di nodi espansi a vantaggio di una maggiore velocità dell'algoritmo.

1.2.3 Inserimento dei figli sul top dello stack

Ci sono vari modi per inserire i figli in cima allo stack: si può stabilire un ordine prefissato in cui applicare gli operatori ed inserire in quest'ordine i figli oppure si possono inserire casualmente i figli sul top dello stack. Nell'algoritmo realizzato, i figli vengono inseriti in modo che quello con il più piccolo valore di *f(n)* si trovi in cima allo stack. L'idea si basa sul fatto che se l'euristica fosse perfetta, il valore di *f(n)* calcolato sulla radice sarebbe uguale a quello calcolato sul goal e su tutti i nodi che si trovano sul cammino che porta dalla radice al goal: quindi, nell'espandere i nodi, dovrei scegliere quello con il valore di *f(n)* uguale al padre, che non è altro che quello con il valore più piccolo di *f(n)*. Anche questo piccolo accorgimento apporta qualche lieve miglioramento all'algoritmo.

Test sull'algoritmo WIDA*

I test sull'algoritmo hanno come obiettivo la riduzione del numero dei nodi espansi[1] e quindi del tempo di ricerca, pur cercando di trovare soluzioni vicine a quelle ottimali. Sono testate varie funzioni per *w*, evidenziandone pregi e difetti.

2.1 Test

Il test viene eseguito su un campione rappresentativo di 1000 configurazioni del gioco dell'otto generate a caso e distinte l'una dall'altra. Tali configurazioni sono salvate su di un file e vengono caricate in memoria centrale quando è necessario. Per ogni configurazione iniziale vengono valutati alcuni parametri ed i dati riportati sulle tabelle sono la somma di questi parametri per tutte e 1000 le configurazioni; quindi, se si è interessati ai valori medi, basta dividere per 1000 i valori riportati nelle tabelle.

2.1.1 La distanza Manhattan

L'euristica usata è la distanza Manhattan: è un'euristica ammissibile, in quanto non sopravvaluta mai la distanza effettiva dal nodo *n* al goal più vicino. Poiché la distanza Manhattan è sempre minore od uguale alla distanza effettiva, l'idea di base è quella di moltiplicarla per un fattore maggiore od uguale ad uno, in modo da avvicinarsi al numero reale di passi che mancano per arrivare al goal. Il rischio è quello di sopravvalutare la distanza effettiva dal goal e quindi di non trovare una soluzione ottimale: infatti, come *w* cresce al di sopra di uno, cresce anche la lunghezza della soluzione, mentre diminuisce il numero di nodi generati. Inoltre, quando $w \geq 1$, la funzione *f(n)* perde la sua monotonicità. Un teorema ci fornisce un limite superiore all'incremento della lunghezza rispetto alla soluzione ottimale:

Teorema: Se $w \geq 1$, la soluzione trovata da WIDA* non può eccedere quella ottima di più di un fattore *w*.

La sovrastima della distanza effettiva implica l'utilizzo di maggiori informazioni provenienti dal problema: anche se questo va a scapito dell'ammissibilità, si possono risolvere gli stessi problemi in meno tempo.

[1] Un nodo si dice espanso se è un nodo goal o se sono stati generati tutti i suoi successori (può anche non avere successori e in tal caso è una foglia).

2.1.2 Obiettivo del test

Data la funzione euristica *h(n)* e data la distanza effettiva dal goal *d(n)*, si vuole determinare una funzione *w* tale che:

$$d(n) = w\, h(n) \text{ con } w \geq 1.$$

Se si riuscisse a trovare una tale funzione *w*, essa garantirebbe l'espansione del numero minimo di nodi, mantenendo anche l'ammissibilità: il problema è che la funzione *d(n)* non è nota a priori.

2.2 Misure di prestazioni

Per verificare la bontà di una funzione per *w* rispetto ad un'altra, vengono misurati vari parametri che tengono conto sia della riduzione del tempo di ricerca, sia dell'aumento del costo della soluzione.

La prima riga di ogni tabella contiene la somma dei costi dei cammini trovati dall'algoritmo di ricerca WIDA*. Questo valore permette di verificare di quanto le soluzioni trovate si discostino da quelle ottime: in altre parole, è una misura di ammissibilità dell'algoritmo. La seconda riga contiene la somma dei nodi espansi in totale dall'algoritmo ed è quindi un valore direttamente proporzionale al tempo impiegato per risolvere il problema. La terza riga contiene la somma dei nodi espansi durante l'ultima ripartenza e serve a caratterizzare quanto tempo viene impiegato in quest'ultima fase rispetto al tempo totale. La quarta riga contiene la somma del numero di ripartenze ed indica quante volte l'algoritmo ritorna sui propri passi. La quinta e la sesta riga contengono la somma delle penetranze. La penetranza è definita come il rapporto tra la lunghezza della soluzione trovata ed il numero di nodi generati. Nelle tabelle viene riportata la somma delle penetranze calcolata utilizzando i nodi espansi all'ultima ripartenza ed i nodi espansi in totale. La prima misura ha anche un'interpretazione grafica e sta ad indicare quanto velocemente l'algoritmo va verso il goal: chiamerò tale misura penetranza reale, mentre l'altra verrà chiamata penetranza totale. La penetranza sia reale sia totale, è un numero strettamente maggiore di zero e minore od uguale ad uno: se la funzione di successione è così precisa da generare un nodo alla volta, allora la penetranza è uguale ad uno. Valori prossimi allo zero indicano un numero molto elevato di nodi espansi rispetto a quelli che costituiscono la soluzione trovata.

2.3 Prestazioni dell'algoritmo IDA*

Vengono di seguito riportate le prestazioni dell'algoritmo IDA* sulle 1000 configurazioni distinte e generate a caso del gioco dell'otto, illustrando i miglioramenti apportati usando un incremento del valore di taglio dipendente dall'iterazione (ripartenza) a cui siamo giunti e inserendo il figlio con il valore di *f(n)* più piccolo sul top dello stack. L'algoritmo IDA* è un caso particolare dell'algoritmo WIDA* in cui $w = 1$.

La tabella seguente riporta i valori ottenuti eseguendo un incremento costante ed uguale ad uno del valore di taglio ed inserendo i figli del nodo espanso in cima allo stack secondo un ordine fissato a priori:

IDA*	w = 1.0
Somma dei cammini	22027
Somma dei nodi espansi in totale	3508711
Somma dei nodi espansi all'ultima ripartenza	864841
Somma delle ripartenze	7924
Somma delle penetranze reali (ultima iterazione)	126
Somma delle penetranze totali (nodi espansi in totale)	40

Tabella 1 Incremento costante del valore di taglio ed ordine prefissato di inserimento dei figli

Si può subito notare l'elevato numero di nodi espansi in totale rispetto a quelli espansi durante l'ultima iterazione: questo è dovuto in gran parte all'elevato numero di ripartenze senza successo che l'algoritmo esegue. Si notano anche dei piccoli valori per le due penetranze.

La seguente tabella riporta invece i valori ottenuti usando un incremento del valore di taglio dipendente dall'iterazione ed inserendo i figli nel top dello stack secondo un ordine determinato in precedenza:

IDA*	w = 1.0
Somma dei cammini	22027
Somma dei nodi espansi in totale	2186776
Somma dei nodi espansi all'ultima ripartenza	864841
Somma delle ripartenze	3962
Somma delle penetranze reali (ultima iterazione)	126
Somma delle penetranze totali (nodi espansi in totale)	54

Tabella 2 Incremento variabile del valore di taglio ed ordine prefissato di inserimento dei figli

Come si può vedere, il numero di ripartenze è esattamente dimezzato, provocando una diminuzione notevole del numero di nodi espansi in totale ed un incremento del valore della penetranza totale; gli altri valori rimangono invariati. Una giustificazione di questo dimezzamento può essere questa: ho notato che la differenza tra la lunghezza del cammino ottimo e la distanza Manhattan è sempre un numero pari. Quindi l'incremento variabile del valore di taglio sarà dato dal più piccolo numero pari diverso da zero, che è ovviamente due, in quanto la distanza minima tra due configurazioni è proprio due.

La seguente tabella riporta infine i valori ottenuti applicando entrambi i miglioramenti all'algoritmo di ricerca:

IDA*	w = 1.0
Somma dei cammini	22027
Somma dei nodi espansi in totale	2155960
Somma dei nodi espansi all'ultima ripartenza	834025
Somma delle ripartenze	3962
Somma delle penetranze reali (ultima iterazione)	135
Somma delle penetranze totali (nodi espansi in totale)	54

Tabella 3 Incremento variabile del valore di taglio ed inserimento ordinato dei figli nello stack

L'inserimento ordinato dei figli provoca un'ulteriore diminuzione dei nodi espansi, incrementando la penetranza reale; gli altri valori rimangono invariati.

Notazione

In tutte le tabelle che seguiranno, verranno utilizzate le seguenti abbreviazioni:
scw sta per somma del costo dei cammini;

snt sta per somma dei nodi espansi in totale;
snu sta per somma dei nodi espansi nell'ultima ripartenza;
sr sta per somma del numero delle ripartenze;
spu sta per somma delle penetranze reali (considerando l'ultima iterazione);
spt sta per somma delle penetranze totali (considerando i nodi espansi in totale).

2.4 Utilizzo di varie funzioni per *w* in WIDA*

Per cercare di migliorare le prestazioni dell'algoritmo IDA*, ho testato varie funzioni per *w*: i risultati ottenuti indicano che si può ottenere una soluzione in modo più veloce solo a scapito dell'ottimalità. Ci sono comunque funzioni che si comportano meglio di altre, nel senso che a parità di nodi espansi garantiscono di trovare una soluzione più vicina a quella ottimale oppure che a parità di incremento della lunghezza della soluzione, espandono meno nodi: lo scopo è quindi di trovare tali funzioni.

2.4.1 Funzione costante

La prima funzione testata per *w* è la funzione $w(h) = a$ dove $a \geq 1$ è un valore costante (se $a = 1$ si ottiene l'IDA*). Tale funzione è indipendente dall'altezza *h* a cui è giunto l'algoritmo: è quindi di facile implementazione. Il grafico di *w(h)* è il seguente:

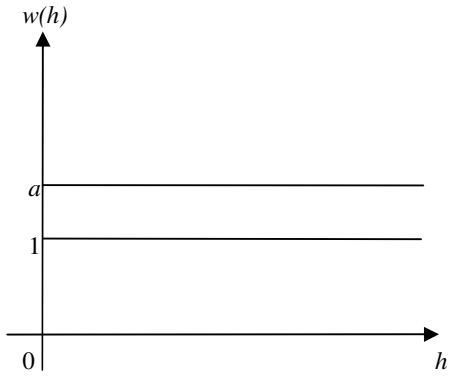


Figura 1 Funzione costante per *w(h)*

dove la variabile indipendente *h* indica l'altezza dell'albero di ricerca a cui siamo giunti e $y = 1$ è il valore sotto il quale la funzione *w(h)* non deve mai scendere. Tale funzione fa perdere l'ammissibilità all'algoritmo, in quanto moltiplica i valori di *h(n)* per un fattore costante indipendente dall'altezza: alcuni dei valori di *wh(n)* saranno certamente superiori a quelli di *d(n)* che rappresenta la vera distanza dal goal. Ecco i dati ottenuti al variare di *a* nell'intervallo [1.1, 2.5]:

a = *cost*	*a* = 1.1	*a* = 1.2	*a* = 1.3	*a* = 1.4	*a* = 1.5	*a* = 1.6
scw	22027	22105	22245	22513	22759	23099
snt	2842260	2510346	2233035	1955538	1862938	1642561
snu	664815	635779	609016	540400	524287	478205
sr	6952	6876	6597	6264	5942	5707
spu	157	164	172	182	182	189
spt	47	52	57	65	68	74

a = *cost*	*a* = 1.7	*a* = 1.8	*a* = 1.9	*a* = 2.0	*a* = 2.1	*a* = 2.5
scw	23319	23795	23947	24571	25179	28971
snt	1570199	1424117	1347613	1225261	1121676	938312
snu	488167	442678	451711	413923	409832	408783
sr	5383	5125	4921	4587	4249	3615
spu	184	188	188	200	200	221
spt	75	80	85	100	109	143

Tabella 4 Funzione *w(h) = a = cost*

I risultati evidenziano una forte diminuzione del numero di nodi espansi all'aumentare di *a*: purtroppo si perde l'ottimalità della soluzione. Anche le due penetranze subiscono un elevato aumento, mentre il numero di ripartenze è superiore (tranne per *a* = 2.5) a quelle di IDA*. Un buon compromesso tra velocità è ammissibilità è dato da *a* = 1.8 oppure *a* = 1.9.
Nota: la funzione *w(h) = a = cost* ottiene dei valori molto bassi per quanto riguarda la somma dei nodi espansi all'ultima iterazione, e dei valori molto alti per la somma delle penetranze (soprattutto per quella reale), difficilmente raggiungibili con altre funzioni.

2.4.2 Funzioni concave del tipo *w(h) = 1 + a / h^(n / m)*

Il passo successivo è stato quello di testare delle funzioni inversamente proporzionali ad una qualche potenza di *h*: quando *h* cresce, tali funzioni tendono ad uno ed per *h* = 1 assumono il valore 1 + *a*. L'andamento del grafico è il seguente:

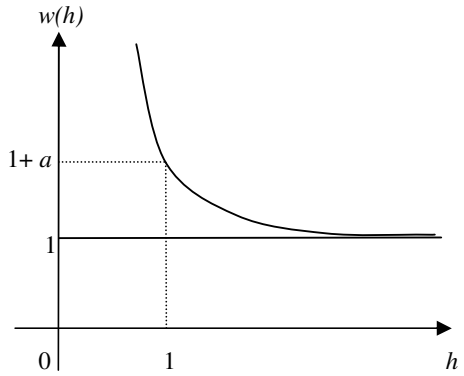


Figura 2 Funzione concava inversamente proporzionale ad *h^(n / m)*

Funzione *w(h) = 1 + a / h*
Ponendo *n* = *m* = 1, si ottiene una funzione per *w(h)* inversamente proporzionale all'altezza *h*: ecco i dati ottenuti al variare di *a*:

	a = 0.5	*a* = 0.7	*a* = 0.85	*a* = 1.0
scw	22569	23841	25289	26605
snt	2649400	2043299	1744680	1519284
snu	907132	1028816	1063353	1046026
sr	3306	2124	1644	1454
spu	122	127	134	140
spt	72	95	114	125

Tabella 5 Funzione *w(h) = 1 + a / h*

I risultati ottenuti sono peggiori di quelli ottenuti con un valore di *w(h)* costante: questo è dovuto al fatto che tale funzione è piuttosto elevata inizialmente (se *a* = 1

e $h = 1$, allora $w(h) = 2$), mentre poi vale pressoché 1. La somma dei nodi espansi all'ultima iterazione è persino peggiore di quella di IDA*.

Funzione *w(h) = 1 + a / h^2*

Il test seguente serve ad evidenziare che una potenza maggiore di uno per *h* porta a dei risultati praticamente uguali a quelli della tabella 5.

	$a = 0.5$	$a = 0.7$	$a = 0.85$	$a = 1.0$
scw	22569	23841	25295	26629
snt	1932378	1681313	1524483	1384784
snu	922306	1054551	1083889	1079510
sr	2427	1694	1394	1259
spu	119	126	131	138
spt	78	98	115	126

Tabella 6 Funzione *w(h) = 1 + a / h^2*

Alcuni parametri migliorano (somma dei nodi totali, somma ripartenze, somma penetranze totali), altri peggiorano (somma dei nodi espansi all'ultima iterazione che è maggiore che in IDA*, somma penetranze reali), rimanendo stabile la situazione complessiva.

Funzione *w(h) = 1 + a / h^(1 / 2)*

Si testano adesso degli esponenti minori di uno per *h*, visto che un incremento non ha portato dei buoni risultati. Ecco i dati per la radice quadrata di *h*:

	$a = 0.5$	$a = 0.7$	$a = 0.85$	$a = 1.0$
scw	22553	23641	24807	25729
snt	2169911	1653315	1402181	1305413
snu	736777	773895	790096	803916
sr	3874	2625	2065	1866
spu	146	147	150	151
spt	75	100	114	122

Tabella 7 Funzione *w(h) = 1 + a / h^(1 / 2)*

Si ottengono dei risultati migliori rispetto alla tabelle 5 e 6 a parità di incremento di lunghezza della soluzione. A differenza delle funzioni che usano un esponente maggiore di 1 per *h*, si ottengono dei miglioramenti rispetto ad IDA*: la somma dei nodi espansi in totale e quella espansi all'ultima iterazione sono peggiori di quelle ottenuti con i valori costanti.

Funzione *w(h) = 1 + a / h^(1 / 3)*

Diminuendo l'esponente di *h*, si misurano i seguenti valori:

	$a = 0.5$	$a = 0.7$	$a = 0.85$	$a = 1.0$
scw	22523	23413	24481	25291
snt	2018800	1664751	1335533	1198130
snu	659817	702680	669020	672910
sr	4363	3134	2504	2318
spu	154	156	161	164
spt	74	98	115	124

Tabella 8 Funzione *w(h) = 1 + a / h^(1 / 3)*

I dati migliorano sensibilmente rispetto alla precedente tabella, pur rimanendo la somma dei nodi espansi in totale e quella dei nodi espansi all'ultima iterazione in-

feriore a quella ottenuta con i valori costanti; i valori sono comunque migliori di quelli ottenuti con IDA*.

Funzione *w(h) = 1 + a / h^(1 / 4)*
L'ultima potenza testata per *h* è 1 / 4 che ci fornisce la seguente tabella:

	a = 0.5	*a* = 0.7	*a* = 0.85	*a* = 1.0
scw	22481	23431	24239	25009
snt	2078834	1544775	1270581	1190663
snu	648714	605391	584502	599847
sr	4693	3512	2916	2682
spu	158	166	172	173
spt	70	100	113	123

Tabella 9 Funzione *w(h) = 1 + a / h^(1 / 4)*

Si ha di nuovo un sensibile incremento delle prestazioni rispetto alla precedente tabella, pur rimando la somma dei nodi espansi all'ultima iterazione superiore a quella ottenuta con i valori costanti: questo conferma il fatto che piccoli esponenti per *h* funzionano meglio rispetto a valori tipo intero.

Nota: le funzioni del tipo *w(h) = 1 + a / h^(n / m)* forniscono dei valori molto bassi per quanto riguarda la somma delle ripartenze, mentre non sono da utilizzare per tutti gli altri parametri.

2.4.2 Funzione lineare *w(h) = a + (1 / a – 1) h / m*

Tale funzione nasce dal fatto che l'euristica *h(n)* = 0 se *n* è uno stato goal; se il goal si trova a profondità *d*, allora *w(d)* = 1 in quanto *w(h)* ≥ 1 qualunque sia *h*. Supponendo che *d* = *am* dove *m* è la distanza Manhattan del nodo iniziale (questa è un'approssimazione che è necessario fare, anche se è piuttosto grossolana) e calcolando l'equazione della retta che passa per i punti (0, *a*) e (*d*, 1), si ottiene tale funzione per *w(h)* di si riporta il grafico:

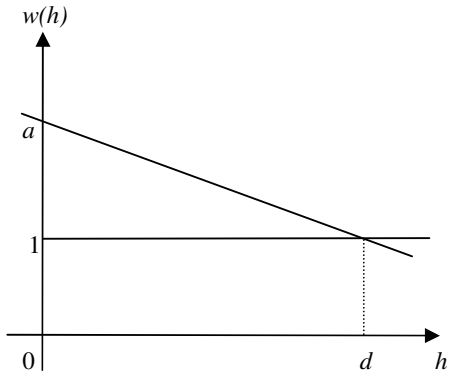


Figura 3 Funzione lineare per *w(h)*

L'idea è quindi di cercare di prevedere la distanza reale dal goal tramite la distanza Manhattan moltiplicata per il fattore *a*. Poiché inizialmente l'altezza *h* è nulla, il fattore *a* rappresenta una stima di quanto in media la distanza Manhattan sottovaluta la distanza effettiva. È molto difficile fornire una stima accurata di quanto in media la distanza Manhattan sottovaluti la distanza effettiva, quindi ho fatto variare il valore *a*, scegliendo poi come migliore stima il valore che fornisce i dati migliori. Ecco la tabella ottenuta al variare di *a*:

	$a = 1.1$	$a = 1.2$	$a = 1.3$	$a = 1.4$	$a = 1.5$
scw	22027	22129	22303	22611	23045
snt	2783385	2432053	2084949	1733426	1484955
snu	723755	689651	654000	600336	579842
sr	6782	6111	5436	4765	4164
spu	148	157	163	165	170
spt	47	53	63	74	81

	$a = 1.6$	$a = 1.7$	$a = 1.8$	$a = 1.9$	$a = 2.0$
scw	23483	24141	24843	25553	26415
snt	1349733	1194500	1058005	974269	950512
snu	571970	564212	529312	551851	575623
sr	3698	3322	3048	2750	2578
spu	176	178	185	180	182
spt	93	103	117	121	133

Tabella 10 Funzione $w(h) = a + (1 / a - 1)\ h / m$

A parità di incremento della soluzione ottimale, si ottengono dei valori migliori di quelli ottenuti con una funzione costante tranne per quanto riguarda la somma dei nodi espansi all'ultima iterazione. La somma delle ripartenze è solo leggermente superiore a quella ottenuta con $w(h) = 1 + a / h^{(n / m)}$.

Nota: la funzione lineare fornisce della cattive prestazioni per $a = 1.1$ e $a = 1.2$; le prestazioni sono comunque simili a quelle della funzione costante.

2.4.3 Funzione $w(h) = ((1 - b\ (1 - a) / am)\ h - ab) / (h - b)$ con $b = am + i$

L'idea che porta alla individuazione di tale equazione è la stessa di quella che porta all'equazione della funzione lineare. Presa la generica equazione dell'iperbole con quattro parametri liberi:

$$y = (ph + q) / (rh + s)$$

si impone il passaggio per i punti $(0, a)$ e $(d, 1)$.

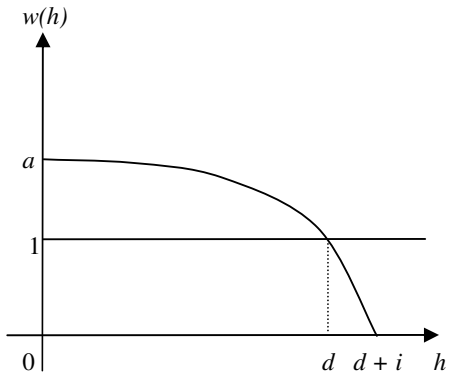


Figura 4 Funzione convessa per $w(h)$

Eseguendo i calcoli, si ottiene:

$$y = ((r + s\ (1 - a) / d)\ h + as) / (rh + s).$$

Dato che l'equazione ha ancora due gradi di libertà, per fissarne la curvatura, si impone l'ulteriore condizione di passaggio per il punto ($d + i$, 0). Dopo aver eseguito i calcoli ed essendosi semplificata l'unica variabile rimasta, si giunge all'equazione non ancora definitiva:

$$y = ((1 - (d + i)(1 - a) / d)\, h - a\,(d + i)) / (h - (d + i)).$$

Fatta l'approssimazione che $d = am$ dove m è la distanza Manhattan della configurazione iniziale, si giunge finalmente alla nostra equazione. Il parametro a assume lo stesso significato dell'equazione lineare, mentre il parametro i individua la curvatura dell'iperbole. Tramite l'approssimazione $d = am$, tale funzione cerca di assumere un valore unitario in corrispondenza dell'altezza della soluzione ottimale. Ecco i valori che si ottengono al variare di a ed i:

$a = 1.3$	$i = 5$	$i = 10$	$i = 15$	$i = 20$	$i = 25$
scw	22403	22509	22459	22445	22413
snt	2859367	2031479	1830115	1845973	1864253
snu	829846	607667	591768	578706	588213
sr	5948	5626	5550	5562	5564
spu	172	176	173	173	173
spt	63	65	65	64	64

Tabella 11 Funzione $w(h) = \{[1 - (am + i)(1 - a) / (am)]\, h - a(am + i)\} / [h - (am + i)]$ con $a = 1.3$

Per $i = 5$ ed $i = 10$ si ottengono delle prestazioni inferiori a quelle ottenute con il valore costante. Per gli altri valori di i, tale funzione fornisce dei valori migliori della funzione costante; per quanto riguarda la somma dei nodi espansi all'ultima iterazione, si riesce a raggiungere la funzione costante. Gli altri valori sono simili a quelli della funzione lineare. Vediamo che cosa succede incrementando a:

$a = 1.5$	$i = 3$	$i = 5$	$i = 8$	$i = 9$	$i = 10$	$i = 11$
scw	23103	23273	23375	23393	23393	23355
snt	2152472	1833276	1495686	1395989	1362903	1350158
snu	687756	622587	503412	494383	484189	495673
sr	5344	5046	4785	4726	4695	4640
spu	180	184	187	185	186	185
spt	73	77	80	80	80	80

$i = 12$	$i = 13$	$i = 14$	$i = 15$	$i = 17$	$i = 20$	$i = 25$
23339	23319	23323	23329	23331	23315	23265
1355169	1370418	1366776	1367543	1361553	1365478	1410895
505746	519273	517507	513539	516662	514404	523751
4588	4590	4572	4577	4568	4523	4464
186	184	184	184	185	185	182
82	82	82	83	83	84	83

Tabella 12 Funzione $w(h) = \{[1 - (am + i)(1 - a) / (am)]\, h - a(am + i)\} / [h - (am + i)]$ con $a = 1.5$

Per $i = 5$ ed $i = 10$ si ottengono sempre delle prestazioni piuttosto scarse; per $i = 10$ si ottengono le migliori prestazioni a parità di incremento della soluzione ottimale. Tutti gli altri valori sono leggermente migliori delle altre funzioni, tranne per quanto riguarda i valori della somma dei nodi espansi all'ultima iterazione e della somma delle penetranze reali ottenuti con la funzione costante. Incrementando ulteriormente a, si ottiene:

$a = 1.7$	$i = 5$	$i = 10$	$i = 15$	$i = 20$	$i = 25$
scw	24075	24275	24281	24233	24245
snt	1490321	1075659	1109007	1116060	1088497
snu	548947	451199	472004	475459	477959
sr	4444	4086	3918	3814	3705
spu	190	192	193	192	192
spt	86	92	97	98	100

Tabella 13 Funzione $w(h) = \{[1 - (am + i)(1 - a) / (am)]\ h - a(am + i)\} / [h - (am + i)]$ con $a = 1.7$

Le prestazioni sono migliori di quelle ottenute con la funzione lineare, ma peggiori anche se non di molto della funzione costante per quanto riguarda la somma dei nodi espansi all'ultima iterazione e la somma delle penetranze reali.

$a = 2.0$	$i = 5$	$i = 10$	$i = 15$	$i = 20$	$i = 25$
scw	25709	26069	26155	26227	26289
snt	1022868	878507	868567	863534	849699
snu	434997	407216	432334	430832	455639
sr	3923	3531	3310	3165	3028
spu	205	206	206	202	198
spt	109	118	122	123	127

Tabella 14 Funzione $w(h) = \{[1 - (am + i)(1 - a) / (am)]\ h - a(am + i)\} / [h - (am + i)]$ con $a = 2.0$

Un valore di $a = 2$ non cambia la precedente situazione.

$a = 2.5$	$i = 5$	$i = 10$	$i = 15$	$i = 20$	$i = 25$
scw	30239	30647	31205	31303	31567
snt	847922	744727	747141	716560	721013
snu	456587	457991	482898	483852	488319
sr	2923	2539	2395	2347	2324
spu	220	216	219	218	221
spt	151	161	165	167	172

Tabella 15 Funzione $w(h) = \{[1 - (am + i)(1 - a) / (am)]\ h - a(am + i)\} / [h - (am + i)]$ con $a = 2.5$

Infine per $a = 2.5$, si ottiene un incremento troppo elevato di mosse rispetto alla soluzione ottimale e dei valori peggiori di quelli raggiunti dalla funzione costante.

2.4.4 Funzione $w(m) = a(m - 10) / 12 + 1$ se $10 \leq m \leq 22$; 1 se $0 \leq m \leq 10$

Dopo aver testato la maggior parte di funzioni in cui w è funzione dell'altezza ed avendo visto quali sono le prestazioni raggiungibili, si testano adesso due funzioni in cui w è funzione della distanza Manhattan m del nodo n. Si parte dall'osservazione che la distanza Manhattan fornisce delle ottime approssimazioni della distanza reale se il valore dell'euristica non è superiore a circa dieci: cioè si può assumere che i valori della distanza Manhattan minori di dieci siano dei valori esatti e che quindi non vadano modificati. Quindi le due funzioni che seguiranno, saranno caratterizzate dal fatto che w è funzione della distanza Manhattan del nodo corrente e non dell'altezza; inoltre se $0 \leq m \leq 10$ circa, allora $w = 1$ indipendentemente da m. La prima funzione esaminata è una funzione lineare a tratti: vi è un primo tratto costante ed uguale ad uno che si raccorda con una retta nel punto (10, 1). Tale retta passa quindi per i punti (10, 1) e (22, 1 + a), dove delle variazioni di a cambiano il valore del coefficiente angolare. Il valore 22 è il valore massimo trovato per la distanza Manhattan. Si riporta il grafico di tale funzione:

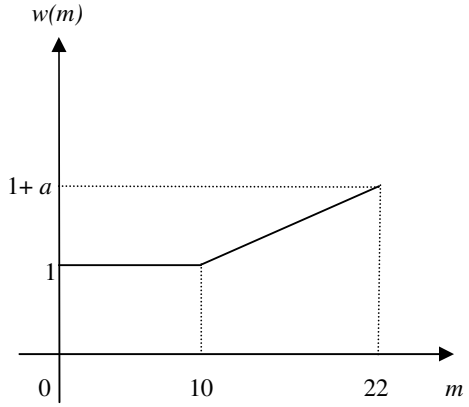


Figura 5 Funzione spezzata per *w(m)*

Ecco i valori che si ottengo per due valori di *a*:

	a = 0.3	*a* = 0.5
scw	22203	22615
snt	2432070	2156233
snu	634313	625190
sr	6544	5800
spu	157	162
spt	52	62

Tabella 16 Funzione $w(m) = a\,(\,m - 10)\,/\,12 + 1$ se $m \geq 10$, se $0 \leq m \leq 10$

Si nota che la somma dei nodi espansi in totale e delle ripartenze è persino peggiore di IDA*: tale funzione non è quindi utilizzabile.

2.4.5 Funzione *w(m) = 1 se 0 ≤ m ≤ l, a se m ≥ l con a ≥ 1*

Si testa adesso una funzione costante a tratti con un gradino a circa metà del dominio, come illustra il seguente grafico, al fine di vedere se è possibile ottenere dei buoni risultati considerando *w* come funzione della distanza Manhattan:

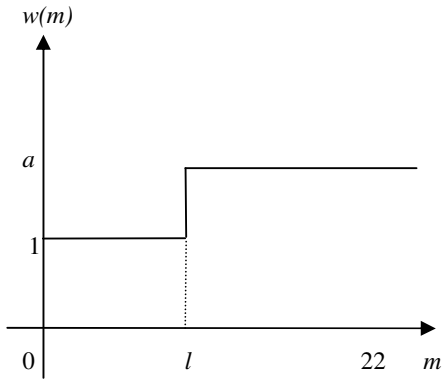


Figura 6 Funzione a gradino per *w(m)*

Il primo tratto è caratterizzato da un valore costante ed uguale ad uno, mentre il secondo tratto da un valore uguale ad *a*: c'è una discontinuità per $m = l$. L'effetto di tale funzione è quello di lasciare invariati i valori per cui $0 \leq m \leq l$, mentre gli altri vengono amplificati del valore costante *a* (è un effetto passa-alto). Si ottengono i seguenti valori al variare dei parametri *a* ed *l*:

$l = 10$	$a = 1.1$	$a = 1.2$	$a = 1.3$	$a = 1.4$	$a = 1.5$
scw	22331	22503	23163	23633	24253
snt	1569353	1968887	1624952	1459873	1306914
snu	665257	520034	508488	469840	493015
sr	4103	5736	4643	4494	3751
spu	155	178	179	185	185
spt	68	66	83	93	105

Tabella 17 Funzione con gradino in $l = 10$ e valore a variabile

A parità di incremento della lunghezza della soluzione trovata, per quanto riguarda la somma dei nodi espansi all'ultima ripartenza, si ottengono dei valori pressappoco uguali a quelli ottenuti con la funzioni costante, mentre gli altri parametri si avvicinano molto a quelli ottenuti con la funzione lineare, migliorando i valori ottenuti con la funzione costante. Per a = 1.1 si ottengono le migliori prestazioni a parità di incremento della lunghezza della soluzione trovata. Quindi tale funzione è una tra quelle che garantiscono delle alte prestazioni. Fissando $a = 1.1$ e facendo variare l, si ottengono i seguenti valori:

$a = 1.1$	$l = 9$	$l = 11$	$l = 12$
scw	22289	22287	22235
snt	2366256	1645761	1737650
snu	637497	709392	726636
sr	6385	4050	3966
spu	162	149	144
spt	55	65	62

Tabella 18 Funzione con $a = 1.1$ e gradino l variabile

I risultati sono peggiori rispetto a quelli riportati nella precedente tabella e indicano di fissare il limite l al valore dieci: piccole variazioni provocano un brusco calo delle prestazioni.

Nota: si verifica una certa instabilità al variare dei parametri in tale tipo di funzione.

2.5 Conclusioni

Facciamo adesso un riepilogo di tutte le funzioni usate, indicando quali parametri riescono ad ottimizzare.

Funzione *w(h) = a = cost*: è la più semplice funzione utilizzabile ma garantisce delle ottime prestazioni: fornisce dei valori ottimi per quanto riguarda le due penetranze e la somma dei nodi espansi all'ultima ripartenza.

Funzioni del tipo *w(h) = 1 + a / h^(n / m)*: è una famiglia di funzioni che non aumenta le prestazioni dell'algoritmo. L'unico parametro che riesce ad ottimizzare è la somma delle ripartente ma non la somma dei nodi espansi in totale.

Funzione lineare *w(h) = a + (1 / a – 1) h / m*: fornisce delle prestazioni leggermente superiori a quelle della funzione costante, eccetto che per la somma dei nodi espansi all'ultima iterazione. Il valore ottenuto per la somma delle ripartenze è simile a quello della precedente famiglia di funzioni.

Funzione convessa ***w(h) = ((1 – b (1 – a) / am) h – ab) / (h – b) con b = am + i*****:** questa classe di funzioni garantisce delle ottime prestazioni per tutti i parametri. Solo la funzione costante fornisce dei valori migliori per quanto riguarda la somma dei nodi espansi all'ultima iterazione.

Funzione ***w(m) = a (m – 10) / 12 + 1 se m ≥ 10, 1 se 0 ≤ m ≤ 10*****:** è un cattivo insieme di funzioni che non fornisce alcun incremento delle prestazioni.

Funzione ***w(m) = 1 se 0 ≤ m ≤ l, a se m ≥ l con a ≥ 1*****:** se fissiamo $l = 10$, otteniamo un'ottima funzione con prestazioni simili a quelle ottenute con le altre migliori funzioni: è forse la migliore funzione trovata.

2.5.1 Quale funzione scegliere?

Non esiste una funzione migliore delle altre in assoluto: ogni funzione ha un proprio campo d'azione in cui garantisce delle prestazioni superiori a quelle ottenute con tutte le altre funzioni. Ad esempio, se siamo disposti a pagare un incremento della lunghezza della soluzione medio di una mossa ogni tre configurazioni risolte, la funzione $w(m) = 1 \text{ se } 0 \leq m \leq l, a \text{ se } m \geq l \text{ con } a = 1.1 \text{ e } l = 10$ garantisce delle prestazioni superiori a tutte le altre; se invece siamo disposti a pagare un incremento medio di 1.3 mosse per configurazione risolta, la funzione $w(h) = ((1 – b (1 – a) / am) h – ab) / (h – b) \text{ con } b = am + i, a = 1.5 \text{ e } i = 10$ fornisce i migliori valori dei parametri.

UNIVERSITA' DEGLI STUDI DI FIRENZE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

Esercitazioni per l'esame di
"Intelligenza Artificiale"
Prof. G. Soda

Classic

di Sauro Menchetti

A.A.1998-99

Il CLASSIC

Il CLASSIC è un linguaggio di modellizzazione che si basa sulla descrizione strutturata dei concetti. Un punto di forza di questo approccio è che lo stesso linguaggio serve a definire lo schema, ad aggiornare il database, a formulare ed a rispondere alle query. Tale approccio permette inoltre di descrivere in modo parziale gli elementi del sistema (non è detto si conoscano subito tutte le informazioni di interesse), di rispondere alle query in modo descrittivo e di estendere in modo semplice lo schema del DB. Il CLASSIC esegue in modo automatico inferenze su concetti ed individui in modo tale che tutto il processo inferenza rimane trattabile. I database tradizionali hanno raggiunto una grande efficienza restringendo la potenza del modello, ma sono per lo più inadeguati in quelle situazioni in cui oggetti complessi sono il modo naturale di descrivere il dominio, quando l'informazione non è completa e diviene disponibile in modo incrementale e quando è utile un ruolo più attivo nel dedurre le relazioni tra i dati. Il campo dei database logici o deduttivi soddisfa la maggior parte delle precedenti richieste, ma il suo principale inconveniente è l'intrattabilità computazionale. Il CLASSIC si sforza quindi di fornire un insieme limitato di costrutti in grado di generare un linguaggio espressivo ed allo stesso tempo computazionalmente trattabile.

1.1 Dominio da modellare

Il dominio da modellare è una piccola parte di un archivio utilizzato dalla segreteria studenti di una facoltà. È di interesse distinguere le varie classi di studenti iscritti alla facoltà in base al sesso, al tipo di iscrizione ed al fatto se sono studenti a tempo pieno o meno (ad esempio se svolgono anche un'attività lavorativa). Combinando le precedenti alternative, gli studenti possono essere classificati in base a diversi criteri e si possono definire dei nuovi concetti: ad esempio, si può definire il concetto di "studente bravo" come quello di uno studente che è iscritto in corso ed è in pari con gli esami, pur svolgendo anche un'attività lavorativa. Per differenziare ed accomunare certe caratteristiche, si crea una gerarchia di concetti in cui i concetti figli ereditano tutte le caratteristiche dei genitori ed in più ne aggiungono di nuove: le classi foglia sono quindi le più complesse e quelle che semanticamente hanno più significato.

1.2 Analisi del dominio

Il primo concetto inserito nel DB è quello di `PERSONA`: tutti gli studenti sono persone ed in quanto tali possiedono un nome, un cognome, un'età ed un recapito. Le persone sono distinte in due classi disgiunte in base al sesso: si parla quindi di uomini e donne. Un altro concetto utile è quello di `OCCUPAZIONE` che viene specializzato in `LAVORATORE` e `STUDENTE`. Il concetto generico di `OCCUPAZIONE` raggruppa le caratteristiche comuni dei due sottoconcetti come ad esempio l'orario, il luogo e la durata (part-time o a tempo pieno). I due sottoconcetti si specializzano aggiungendo attributi e relazioni proprie come la data di assunzione, lo stipendio, il numero di matricola e l'anno di iscrizione. A partire dai due concetti di studente e di lavoratore, è naturale definire il concetto di `STUDENTE-LAVORATORE` che ingloba tutte le caratteristiche degli studenti e dei lavoratori. Infine è opportuno definire il concetto di `ISCRIZIONE` distinguibile nei due sottinsiemi disgiunti in corso e fuori corso e caratterizzato da una facoltà, da un anno di immatricolazione, da un anno accademico e da una tassa di iscrizione.

1.3 Definizione dello schema

La definizione dello schema consiste nel dare dei nomi ai concetti ed ai ruoli che sono di interesse per i vari utenti, in modo da costruire un vocabolario per interagire con il database. Un DB costruito con il CLASSIC è per lo più un archivio che contiene informazioni su *oggetti individuali* o *individui*. Gli oggetti hanno un'identità intrinseca e sono correlati gli uni agli altri per mezzo di relazioni binarie dette *ruoli* (noti anche come attributi o proprietà). Gli individui sono raggruppati in insiemi detti *concetti* che ne mettono in risalto le caratteristiche comuni. Una caratteristica del CLASSIC è quella di definire i concetti in modo composizionale: alcuni concetti sono ottenuti raggruppando insieme degli individui, altri più complessi raggruppano alcuni concetti già definiti. La definizione dello schema si distingue in una dichiarazione dei ruoli ed in una definizione dei concetti. Il DDL (data definition language) permette la definizione dello schema del DB.

Notazione: le parti riguardanti l'utilizzo del CLASSIC sono scritte in `Courier New`. Più precisamente, i simboli scritti in maiuscolo sono `CONCETTI`, quelli in minuscolo sono `ruoli` o `indici` e quelli misti sono `Individui`. I costruttori dei concetti come **`PRIMITIVE`** sono scritti con **`QUESTI CARATTERI`**. Gli operatori sul database sono scritti in **`questo stile`**, mentre le procedure del linguaggio ospite che sono argomenti del costrutto **`TEST`** in *`questi caratteri`*.

1.3.1 Dichiarazione dei ruoli

Serve per comunicare al database che un certo identificatore sarà usato come nome di un ruolo, in modo da permettere la rilevazione di alcuni tipi di errore come ad esempio quelli di tipo.

Ruoli per il concetto PERSONA:

```
define-role[nome]
define-role[cognome]
define-role[età]
define-role[recapito]
```

Ruoli per il concetto OCCUPAZIONE:

```
define-role[impiego]
define-role[orario]
define-role[durata]
define-role[luogo]
```

Ruoli per il concetto LAVORATORE:

```
define-role[data-assunzione]
define-role[mestiere]
define-role[qualifica]
define-role[stipendio]
```

Ruoli per il concetto STUDENTE:

```
define-role[matricola]
define-role[anno-iscrizione]
define-role[libretto]
define-role[piano-di-studi]
define-role[lista-esami-sostenuti]
```

Ruoli per il concetto ISCRIZIONE:

```
define-role[facoltà]
define-role[anno-immatricolazione]
define-role[anno-accademico]
define-role[tassa-iscrizione]
```

Ruolo per i concetti ISCRIZIONE-IN-CORSO e ISCRIZIONE-FUORI-CORSO:

```
define-role[numero-di-iscrizioni]
```

Ruoli per il concetto STUDENTE-BRAVO:

```
define-role[esami-previsti].
```

1.3.2 Definizione dei concetti

Serve a definire i concetti del dominio in modo composizionale, partendo dai più semplici fino a quelli più complessi. Si distinguono in primitivi e definiti: la descrizione di un concetto primitivo è intesa a rappresentare solo *condizioni necessarie* per l'estensione del concetto e l'appartenenza di un individuo all'estensione *deve* essere stabilita dall'utente; la descrizione di un concetto definito è intesa a rappresentare *condizioni necessarie e sufficienti* per l'estensione del concetto e l'appartenenza di un individuo all'estensione *viene calcolata* dal sistema. Per la definizione dei concetti vengono usati i concetti di `STRING` ed `INTEGER` facenti parte dell'host language in cui è implementato il CLASSIC.

Definizione del concetto `PERSONA`:

```
define-concept[PERSONA,
               (PRIMITIVE (AND THING
                               (AT-LEAST 1 nome)
                               (ALL nome STRING)
                               (AT-LEAST 1 cognome)
                               (ALL cognome STRING)
                               (AT-LEAST 1 età)
                               (AT-MOST 1 età)
                               (ALL età (AND INTEGER
                                             (TEST >=0)
                                             (TEST <=200)))
                               (AT-LEAST 1 recapito)
                               (ALL recapito STRING))
                persona)]
```

Definizione dei concetti `UOMO` e `DONNA`:

```
define-concept[UOMO,
               (DISJOINT-PRIMITIVE PERSONA genere maschile)]
define-concept[DONNA,
               (DISJOINT-PRIMITIVE PERSONA genere femminile)]
```

Definizione del concetto `OCCUPAZIONE`:

```
define-concept[OCCUPAZIONE,
               (PRIMITIVE (AND THING
                               (AT-LEAST 1 impiego)
                               (ALL impiego STRING)
                               (AT-LEAST 1 orario)
                               (ALL durata (ONE-OF
                                            part-time
                                            tempo-pieno))
                               (ALL luogo (ONE-OF
                                           ufficio
                                           scuola
                                           fabbrica)))
                occupazione)]
```

Definizione del concetto LAVORATORE:

```
define-concept[LAVORATORE,
               (AND PERSONA
                    OCCUPAZIONE
                    (AT-LEAST 1 data-assunzione)
                    (AT-LEAST 1 mestiere)
                    (AT-LEAST 1 qualifica)
                    (AT-LEAST 1 stipendio)
                    (ALL stipendio (AND INTEGER
                                        (TEST >=0)))
                    (ALL luogo
                    (ONE-OF ufficio fabbrica)))]
```

Definizione del concetto STUDENTE:

```
define-concept[STUDENTE,
               (AND PERSONA
                    OCCUPAZIONE
                    (AT-LEAST 1 matricola)
                    (AT-MOST  1 matricola)
                    (ALL matricola (AND INTEGER
                                        (TEST >=0)))
                    (AT-LEAST 1 anno-iscrizione)
                    (AT-MOST  1 anno-iscrizione)
                    (ALL anno-iscrizione
                         (AND INTEGER (TEST >=0)))
                    (AT-LEAST 1 libretto)
                    (AT-MOST  1 libretto)
                    (ALL libretto STRING)
                    (AT-LEAST 1 piano-di-studi)
                    (AT-MOST  1 piano-di-studi)
                    (ALL piano-di-studi STRING)
                    (AT-LEAST 1 lista-esami-sostenuti)
                    (ALL lista-esami-sostenuti STRING)
                    (SAME-AS (luogo)(ONE-OF scuola)))]
```

Definizione del concetto STUDENTE-LAVORATORE:

```
define-concept[STUDENTE-LAVORATORE, (AND LAVORATORE STUDENTE)]
```

Definizione del concetto ISCRIZIONE:

```
define-concept[ISCRIZIONE,
               (PRIMITIVE (AND THING
                               (AT-LEAST 1 facoltà)
                               (AT-MOST  1 facoltà)
                               (ALL facoltà STRING)
                               (AT-LEAST 1 anno-immatricolazione)
                               (AT-MOST  1 anno-immatricolazione)
                               (ALL anno-immatricolazione
                               (AND INTEGER (TEST >=0)))
                               (AT-LEAST 1 anno-accademico)
                               (AT-MOST  1 anno-accademico)
                               (ALL anno-accademico
                                    (AND INTEGER (TEST >=0)))
                               (SAME-AS (anno-iscrizione)
                                        (anno-accademico))
```

```
                              (AT-LEAST 1 tassa-iscrizione)
                              (ALL tassa-iscrizione
                                   (AND INTEGER (TEST >=0))))
                iscrizione)]
```

Definizione del concetto ISCRIZIONE-IN-CORSO:

```
define-concept[ISCRIZIONE-IN-CORSO,
               (DISJOINT-PRIMITIVE
                   (AND ISCRIZIONE (SAME-AS
                                        (anno-iscrizione)
                                        (numero-di-iscrizioni)))
                tipo-iscrizione in-corso)]
```

Definizione del concetto ISCRIZIONE-FUORI-CORSO:

```
define-concept[ISCRIZIONE-FUORI-CORSO,
               (DISJOINT-PRIMITIVE
                   (AND ISCRIZIONE
                        (TEST anno-iscrizione < numero-iscrizioni))
                tipo-iscrizione fuori-corso)]
```

Definizione del concetto STUDENTE-BRAVO:

```
define-concept[STUDENTE-BRAVO,
               (AND STUDENTE-LAVORATORE
                    ISCRIZIONE-IN-CORSO
                    (SAME-AS (lista-esami-sostenuti)
                             (esami-previsti)))]
```

Definizione del concetto STUDENTESSA-LAVORATRICE-FUORI-CORSO:

```
define-concept[STUDENTESSA-LAVORATRICE-FUORI-CORSO,
               (AND DONNA
                    STUDENTE-LAVORATORE
                    ISCRIZIONE-FUORI-CORSO)].
```

1.3.3 Gerarchia tra concetti

La precedente definizione dei concetti induce la seguente gerarchia tra i vari concetti:

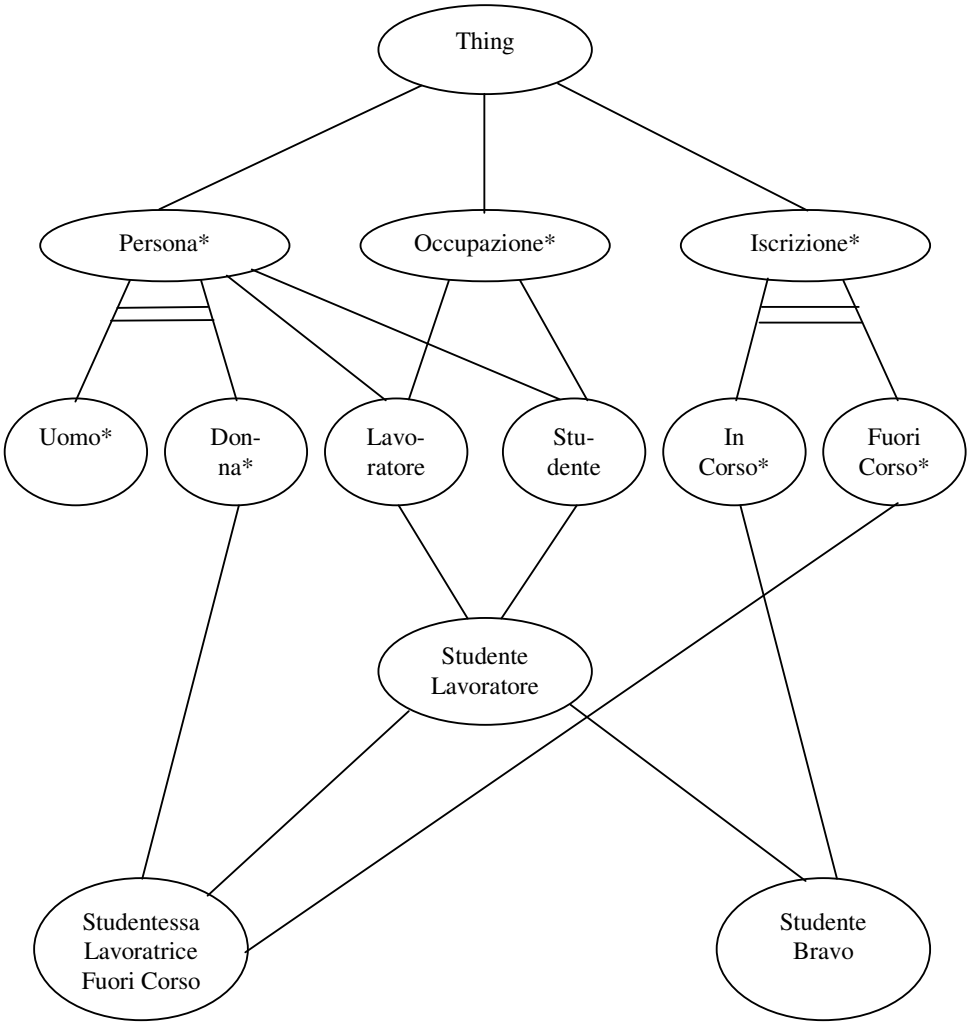


Figura 1: Gerarchia dei concetti

Il simbolo * indica i concetti primitivi, mentre la doppia linea = lega le coppie dei concetti primitivi disgiunti. Preoccupiamoci adesso di inserire delle informazioni nel DB.

1.4 Aggiornamento del Database

L'aggiornamento del DB avviene in modo incrementale e non è quindi necessario conoscere subito tutte le informazioni riguardanti i vari individui. Il CLASSIC permette di memorizzare una visione incompleta dei fatti e di aggiungere di volta in volta le informazioni che divengono disponibili: questo approccio è particolarmente utile in quei domini in cui vi è un'alta incertezza. Il DML (data manipulation language) permette l'aggiornamento del contenuto del DB.

Inserimento dell'individuo `Studente1`:

```
create-ind[Studente1]
assert-ind[Studente1, STUDENTE]
assert-ind[Studente1, (FILLS nome Luca)]
assert-ind[Studente1, (CLOSE nome)]
assert-ind[Studente1, (FILLS cognome Rossi)]
assert-ind[Studente1, (CLOSE cognome)]
assert-ind[Studente1, (FILLS età 23)]
assert-ind[Studente1, (FILLS recapito Via Garibaldi 3 Firenze)]
assert-ind[Studente1, (FILLS impiego Studente)]
assert-ind[Studente1, (FILLS orario 8-13)]
assert-ind[Studente1, (FILLS durata tempo-pieno)]
assert-ind[Studente1, (FILLS luogo scuola)]
assert-ind[Studente1, (FILLS matricola 2124345)]
assert-ind[Studente1, (FILLS anno-iscrizione 1°)]
assert-ind[Studente1, (FILLS lista-esami-sostenuti
                             Chimica Informatica Geometria)]
define-role[mezzo-di-trasporto]
assert-ind[Studente1, (ALL mezzo-di-trasporto
                      (ONE-OF autobus auto motorino bicicletta))]
```

Inserimento dell'individuo `StudenteBravo1`:

```
create-ind[StudenteBravo1, STUDENTE-BRAVO]
assert-ind[StudenteBravo1,
          (AND (FILLS nome Marco)
               (CLOSE nome)
               (FILLS cognome Bianchi)
               (CLOSE cognome)
               (FILLS età 22)
               (FILLS recapito Via Tevere 6 Roma))]
assert-ind[StudenteBravo1,
          (AND (FILLS impiego Studente)
               (FILLS durata tempo-pieno)
               (FILLS orario 8,30-13,30)
               (FILLS luogo scuola)
               (FILLS impiego Lavoratore)
               (FILLS durata part-time)
               (FILLS orario non-precisato)
               (FILLS luogo ufficio)
               (CLOSE impiego))]
define-role[laureando]
assert-ind[StudenteBravo1, (AND (ALL laureando (ONE-OF Si No))
                                (FILLS laureando Si))]
assert-ind[StudenteBravo1,
          (AND (FILLS data-assunzione 5-1-1995)
               (CLOSE data-assunzione)
```

```
                (FILLS mestiere Impiegato)
                (CLOSE mestiere)
                (FILLS qualifica 1° Livello)
                (CLOSE qualifica)
                (FILLS stipendio 500000)
                (CLOSE stipendio)
                (FILLS matricola 1805698)
                (FILLS anno-iscrizione 5°)
                (FILLS lista-esami-sostenuti tutti))]
assert-ind[StudenteBravo1,
           (AND (FILLS facoltà Ingegneria)
                (FILLS anno-immatricolazione 1994)
                (FILLS anno-accademico 1999)
                (FILLS tassa-iscrizione 750000))]
define-role[media]
assert-ind[StudenteBravo1,
           (AND (ALL media INTEGER) (FILLS media 107))]
```

Inserimento dell'individuo `StudententessaLavoratriceFuoriCorso1`:

```
create-ind[StudentessaLavoratriceFuoriCorso1,
                              STUDENTESSA-LAVORATRICE-FUORI-CORSO]
assert-ind[StudentessaLavoratriceFuoriCorso1,
           (AND (FILLS nome Daniela)
                (CLOSE nome)
                (FILLS cognome Lunghi)
                (CLOSE cognome)
                (FILLS età 26)
                (FILLS recapito Via Montale 9 Pisa))]
assert-ind[StudentessaLavoratriceFuoriCorso1,
           (AND (FILLS impiego Studente)
                (FILLS impiego Lavoratore)
                (CLOSE impiego))]
assert-ind[StudentessaLavoratriceFuoriCorso1,
           (AND (FILLS data-assunzione 5-1-1995)
                (CLOSE data-assunzione)
                (FILLS stipendio 1500000)
                (CLOSE stipendio)
                (FILLS matricola 1805698)
                (FILLS anno-iscrizione 4°))]
assert-ind[StudentessaLavoratriceFuoriCorso1,
           (AND (FILLS facoltà Farmacia)
                (FILLS anno-immatricolazione 1991)
                (FILLS anno-accademico 1999)
                (FILLS tassa-iscrizione 1450000))].
```

1.4.1 Regole

Una regola consiste di due operandi, un antecedente e un conseguente, entrambi concetti. Sono della forma "se un individuo è un <*concetto1*>, allora è anche un <*concetto2*>" e aggiungono al database l'informazione che ogni istanza del <*concetto1*> è anche un'istanza del <*concetto2*>. Questo è diverso dal definire il <*concetto2*> come parte del concetto <*concetto1*>, in quanto in quel caso il sistema non riconosce un individuo facente parte del <*concetto1*> se non soddisfa anche il <*concetto2*>. La regola permette invece al DB di dedurre che qualsiasi istanza del <*concetto1*> è anche un'istanza del <*concetto2*>. Ecco un esempio di regola:

```
define-role[impegno]
```

```
assert-rule[STUDENTE, (AND (ALL impegno (ONE-OF poco molto))
                           (FILLS impegno molto))]
```

significa che ogni individuo che è uno `STUDENTE` ha come riempitore del ruolo `impegno` la stringa `molto` (il ruolo `impegno` può assumere solo i due valori `poco` e `molto`).

1.5 Interrogazione del Database

Il CLASSIC mette a disposizione una serie di operatori per l'interrogazione del DB: tali operatori fanno parte del Query Language. Si possono formulare interrogazioni sia sui concetti, sia sui ruoli. Ecco alcuni esempi di query sui concetti:

```
concept-aspect[STUDENTE, AT-LEAST]
```

restituisce tutti gli **AT-LEAST** che fanno parte della definizione del concetto di `STUDENTE`;

```
concept-aspect[STUDENTE-BRAVO, ALL, lista-esami-sostenuti]
```

restituisce il tipo del vincolo sul riempitore del ruolo `lista-esami-sostenuti`;

```
concept-aspect[ISCRIZIONE-IN-CORSO, AT-LEAST, facoltà]
```

restituisce il limite inferiore sul vincolo del ruolo `facoltà`;

```
concept-subsumes[STUDENTE, STUDENTE-LAVORATORE]
```

restituisce vero se ogni individuo che soddisfa `STUDENTE-LAVORATORE` è anche un'istanza di `STUDENTE` (relazione di sussunzione). Ecco delle query sui riempitori dei ruoli:

```
ind-aspect[Studente1, FILLS, nome]
```

restituisce il nome dell'individuo `Studente1`.

```
ind-aspect[StudenteBravo1, CLOSE]
```

restituisce i ruoli dell'individuo `StudenteBravo1` chiusi da **CLOSE**.

```
ind-aspect[StudentessaLavoratriceFuoriCorso1, FILLS, stipendio]
```

restituisce l'ammontare dello stipendio dell'individuo `StudentessaLavoratriceFuoriCorso1`.

UNIVERSITA' DEGLI STUDI DI FIRENZE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

Esercitazioni per l'esame di
"Intelligenza Artificiale"
Prof. G. Soda

Neural Networks

di Sauro Menchetti

A.A.1998-99

Introduzione

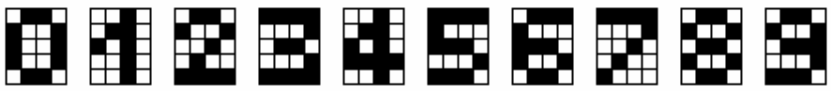
Il software bp è un simulatore di reti neurali che implementa l'algoritmo di propagazione all'indietro (backpropagation) per l'addestramento della rete. Una rete neurale rappresenta una funzione parametrizzata da un insieme di pesi ed è una rappresentazione particolarmente utile in quelle situazioni in cui si ha a che fare con funzioni che devono gestire ingressi affetti da rumore, situazioni in cui le tecniche basate sulla logica formale si trovano talvolta in difficoltà. Lo scopo dell'addestramento è quello di aggiustare i parametri della rete in base ad un insieme di esempi in modo che la funzione rappresentata si avvicini sempre più a quella che risolve correttamente il problema.

1.1 Descrizione del problema

Il problema trattato riguarda il riconoscimento delle dieci cifre numeriche: la rete prende come input una qualsiasi cifra disturbata o meno da rumore e cerca di fornire come output la cifra corrispondente. Un'apposita funzione si occupa di prendere in ingresso una cifra priva di disturbo e di fornire in uscita la stessa cifra alterata dal rumore. Si tratta quindi di un problema di classificazione dell'input in 10 classi distinte. L'obiettivo dell'addestramento della rete è quello di riuscire a classificare correttamente anche le cifre rumorose raggiungendo una certa capacità di generalizzazione, in modo da ricondurre un qualsiasi ingresso in una delle 10 classi previste che più gli assomiglia.

1.1.1 Codifica delle cifre

Ogni cifra viene codificata in una matrice di zero e di uno di dimensioni cinque per quattro. Ecco le dieci cifre non alterate dal rumore:



L'input è quindi costituito da un vettore di 20 elementi binari che linearizza la matrice cinque per quattro della cifra. Gli input riguardanti una singola cifra vengono forniti contemporaneamente agli ingressi della rete neurale. La funzione generatrice di rumore può aggiungere, togliere o spostare un pixel; non vengono prese in considerazione rotazioni o traslazioni dell'immagine.

1.2 Scelta dell'architettura della rete neurale

La scelta della corretta architettura di rete adatta a risolvere un dato compito è un problema di ricerca nello spazio delle possibili architetture (il problema dell'apprendimento è invece una ricerca nello spazio dei pesi): una rete troppo piccola può essere incapace di risolvere un certo problema mentre una rete con troppi parametri può creare fenomeni di overfitting, con un forte apprendimento dell'insieme degli esempi ed incapacità di generalizzare. Poiché si tratta di un problema di riconoscimento di caratteri, l'architettura della rete ricadrà tra gli autoassociatori. Un autoassociatore è una rete feedforward con uguale numero di ingressi e di uscite e con uno strato nascosto composto da un numero di neuroni inferiore. L'autoassociatore scelto ha 20 ingressi e 20 uscite. Il numero di ingressi e di uscite è quindi uguale al numero di elementi della matrice del carattere. La scelta del numero di neuroni nascosti è invece più critica. Poiché un autoassociatore effettua una compressione dell'informazione, 3 o 4 neuroni nascosti potrebbero essere sufficienti per il nostro problema in quanto per codificare 10 classi bastano 4 neuroni. Scegliere l'architettura più piccola che riesce ad apprendere i dati può però portare problemi di sensitività ai parametri iniziali: occorrono quindi dei test per individuare il numero di neuroni nascosti della rete sufficiente a risolvere il problema.

1.3 Il problema della generalizzazione a nuovi esempi

Quando un sistema viene addestrato per mezzo di esempi, una questione di importanza fondamentale è quella della sua capacità di riconoscere e classificare correttamente pattern che non appartengono all'ambiente di apprendimento. Se il sistema ha semplicemente acquisito l'attitudine a memorizzare gli esempi noti, avrà buone prestazioni in fase di apprendimento ma fallirà di fronte ad esempi nuovi che pure si discostano di poco da quelli conosciuti. Per ottenere un'elevata capacità di generalizzazione, deve essere immagazzinata nella rete quanta più conoscenza a priori possibile sul problema, limitando contemporaneamente il numero degli elementi costitutivi dell'architettura (neuroni e connessioni). Indipendentemente dall'algoritmo utilizzato per costruire e addestrare la rete neurale, l'apprendimento ottimo e la generalizzazione restano traguardi difficili da raggiungere contemporaneamente: si può dire che l'uno è inversamente proporzionale all'altro, tanto che la scelta di una rete che realizzi un compromesso soddisfacente fra caratteristiche così contrastanti rimane sempre molto difficile.

"apprendimento ottimo" $\propto$ *1 /* *"generalizzazione"* $\Rightarrow$ *compromesso*

Un approccio per evitare il fenomeno dell'overfitting dei dati è quello di valutare la capacità di generalizzazione della rete durante l'addestramento e di interrompere il processo quando si verifica una tendenza al ribasso: si suddivide approssimativamente in due parti uguali l'insieme di esempi a disposizione dette rispettivamente training set e test set e se ne utilizza una metà per l'addestramento e l'altra per la fase di verifica. L'addestramento termina quando l'errore di validazione comincia a crescere.

Il software bp

Il software bp è un simulatore di reti neurali che implementa l'algoritmo di propagazione all'indietro (backpropagation) per l'addestramento della rete. Utilizza vari file da cui legge le informazioni necessarie e stampa sullo schermo gli output relativi alle diverse elaborazioni. Sono di seguito descritti i vari file necessari a risolvere il problema.

2.1 Network file

Il network file descrive la struttura della rete ed ha generalmente estensione *.net*. Nel file devono essere indicati il numero di neuroni della rete compresi quelli fittizi di ingresso, divisi per strati e ordinati. Sui pesi possono essere definite delle limitazioni. Il file si divide in varie sezioni ognuna delle quali termina con il separatore *end*. Ecco il file per un autoassociatore con 20 ingressi, 20 uscite ed 8 neuroni nascosti:

```
definitions:
nunits   48
ninputs  20
noutputs 20
end
network:
%r 20  8  0 20
%r 28 20 20  8
end
biases:
%r 20 28
end
```

La prima sezione descrive una rete con 48 unità di cui 20 sono ingressi e 20 sono uscite. La sezione riguardante i vincoli sui pesi non è presente. La successiva sezione descrive le connessioni della rete. Indica che dal neurone numero 20 della rete per 8 neuroni, si hanno delle connessioni dal neurone 0 per 20 neuroni e tali connessioni sono inizializzate in modo random; inoltre dal neurone 28 per 20 neuroni, si hanno delle connessioni dal neurone 20 per 8 neuroni anch'esse inizializzate in modo random. L'ultima sezione pone delle soglie (bias) dal neurone 20 per 28 neuroni inizializzate in modo random.

2.2 Weights file

Il weights file contiene i valori dei pesi della rete ed ha generalmente estensione *.wts*. Può essere letto o scritto direttamente dal programma bp tramite i comandi *get / weights* e *save / weigths* dopo che il file *.net* è stato caricato per costruire l'architettura. Consiste in una lista di pesi seguita da una lista di soglie. La lista di pesi è formata da una serie di righe, una per ogni unità con connessioni convergenti. Le righe sono ordinate in base al numero dell'unità e consistono in una serie di valori ordinati secondo il numero dell'unità a cui sono collegati. Il comando *save / weights* salva un solo peso in ogni riga. I pesi non contengono nessuna indicazione relativamente al link al quale si applicano, né è segnato il confine fra pesi effettivi e bias, perché bp è in grado di trarre queste informazioni dal file di descrizione della rete. Bias inesistenti (come quelli dei neuroni in ingresso) sono posti a 0.

2.3 Template file

Il template file stabilisce il layout di visualizzazione dei risultati ed ha generalmente estensione *.tem*. È suddiviso in due sezioni: una prima opzionale in cui si cui si descrive il layout dello schermo e che termina con il separatore *end* ed una seconda in cui si danno le specifiche dei vari template[1]. Di seguito è riportato tale file:

```
define: layout
                                patno  ipatterns   tpatterns
epoch $    tss  $                $     $           $
           gcor $                      $           $
pname $    pss  $                      $           $
                                       $           $
                                       $           $
hidden   $

output1  $
target1  $

output2  $
target2  $

end

epochno    variable  1   $  0  epochno        4    1.0
tss        floatvar  1   $  1  tss            6    1.0
gcor       floatvar  2   $  5  gcor           6    1.0
cpname     variable  2   $  8  cpname        -4    1.0
pss        floatvar  2   $  9  pss            6    1.0

env.patno  variable  3   $  2  patno          3    1.0
env.ipat   vector    3   $  3  activation  h  2    1.0   0   4
env.ipat   vector    3   $  6  activation  h  2    1.0   4   4
env.ipat   vector    3   $ 10  activation  h  2    1.0   8   4
env.ipat   vector    3   $ 12  activation  h  2    1.0  12   4
env.ipat   vector    3   $ 14  activation  h  2    1.0  16   4
```

[1] Un template è un qualunque oggetto disegnabile sullo schermo.

```
env.tpat   vector    3   $  4   target      h  2     1.0   0   4
env.tpat   vector    3   $  7   target      h  2     1.0   4   4
env.tpat   vector    3   $ 11   target      h  2     1.0   8   4
env.tpat   vector    3   $ 13   target      h  2     1.0  12   4
env.tpat   vector    3   $ 15   target      h  2     1.0  16   4

hidden     vector    3   $ 16   activation  h  5  1000.0  20   3
output1    vector    3   $ 17   activation  h  5  1000.0  23  10
target1    vector    3   $ 18   target      h  5  1000.0   0  10
output2    vector    3   $ 19   activation  h  5  1000.0  33  10
target2    vector    3   $ 20   target      h  5  1000.0  10  10
```

Ogni simbolo `$` nella parte dedicata al layout indica l'angolo in alto a sinistra da cui verrà stampato il template. Le stringhe che compaiono nel layout vengono stampate così come compaiono. Il file template produce un layout di uscita in cui si mostrano il numero di epoche, gli errori quadratici relativi al pattern corrente (l'ultimo di ogni epoca) e totale, la quantità *gcor*, il nome e il numero del pattern corrente insieme al pattern stesso ed al target atteso. Nella parte bassa dello schermo si mostrano le uscite dei neuroni nascosti, dei neuroni di output insieme ai target attesi, suddivise in varie righe. Nelle sezione successiva dedicata alle specifiche, ogni template è descritto con tutti i suoi vari attributi. Consideriamo il seguente template:

```
hidden     vector    3   $ 16   activation  h  5  1000.0  20   3
```

La prima stringa indica il nome del template che identifica l'oggetto visivo, segue poi il tipo: in questo caso abbiamo a che fare con un vettore. Il numero `3` rappresenta il valore del *dlevel* per questo template. La coppia `$ 16` indica che il template deve essere stampato a partire dal sedicesimo `$` del layout. `activation` è il nome della variabile alla quale si riferisce il template così come è nota al programma. `h` indica l'orientamento del vettore (`h` per orizzontale e `v` per verticale). Il numero `5` fissa il numero di spazi per la stampa del template, mentre il valore `1000.0` è il fattore di scala. Gli ultimi due valori specificano di stampare 3 elementi del vettore a partire dal ventesimo.

2.4 Pattern file

Il pattern file memorizza le coppie di esempio per la rete ed ha generalmente estensione *.pat*. Ogni coppia consiste di un nome, di *ninputs* ingressi che specificano il pattern in ingresso e di *noutput* uscite che indicano il target atteso. I valori numerici sono separati da spazi e non ci sono separatori speciali tra i pattern ed i target. Ecco un esempio di pattern file per un autoassociatore con 4 ingressi e 4 uscite:

```
one    1 0 0 0   1 0 0 0
two    0 1 0 0   0 1 0 0
three  0 0 1 0   0 0 1 0
four   0 0 0 1   0 0 0 1
```

Il pattern file per il nostro problema è troppo complesso ed è quindi generato in modo automatico da un programma.

2.4.1 Il generatore del pattern file

Il programma genera un file di testo con tutte le informazioni utili sull'insieme di addestramento. Si utilizzano i seguenti tipi di dato definiti dall'utente:

```
typedef int Numero[NRIG][NCOL];

typedef struct coppia
{
     char nome[10];
     Numero pattern;
     Numero target;
} Coppia;

typedef Coppia Allenamento[NUMCOPPIE];
```

Il tipo `Numero` rappresenta la matrice della cifra, la struttura `Coppia` contiene il nome del pattern in ingresso, il pattern e il target di esempio e il vettore `Allenamento` memorizza l'intero ambiente di addestramento. La procedura fondamentale del programma è quella che aggiunge del disturbo ad un carattere che ne è privo. Opera in tre diversi modi: può togliere, aggiungere o spostare un pixel alla cifra. Ecco la sua dichiarazione:

```
void rumore(const Numero originale, Numero disturbato);
```

Un'altra procedura si occupa poi di creare l'insieme di allenamento ed ha la seguente dichiarazione:

```
void creaAllenamento(Allenamento all);
```

Le prime dieci posizioni del vettore `all` sono occupate dalle coppie prive di disturbo, mentre le rimanenti dagli esempi rumorosi. Infine un'apposita procedura si occupa di generare il pattern file, scrivendo su disco il vettore precedentemente creato.

2.5 Start file

Lo start file fornisce le principali informazioni di configurazione al software bp ed ha generalmente estensione *.str*. Per utilizzare il package che implementa backpropagation, si deve lanciare il comando:

bp template.tem start.str

dove *start.str* rappresenta il file di start. I principali comandi contenuti nello start file sono i seguenti:

```
get network  network.net
get patterns pattern.pat
get weights  weights.wts

set mode lgrain pattern
set ecrit   0.4
set nepochs 500
set param lrate  0.5
set param wrange 1.0

set mode follow  1
set mode cascade 0
set dlevel 3
set slevel 2
set lflag  1

set single 1
set stepsize epoch

disp opt standout 1

log logfile.log
```

I primi tre comandi si occupano di caricare l'architettura della rete, l'insieme iniziale di pesi e l'ambiente di addestramento. Si fissa a 500 il numero di epoche dell'addestramento on-line, con un learning rate uguale a 0.5; l'addestramento si ferma se l'errore quadratico relativo ad un'epoca è inferiore a 0.4. I pesi sono inizializzati in modo casuale con valori compresi tra –0.5 e 0.5. Si richiede il calcolo della correlazione del gradiente e si indica di fermarsi dopo ogni epoca. I valori negativi vengono stampati sullo schermo senza il segno meno in reverse video. Viene creato anche un logfile in cui vengono memorizzati certi template ed infine si inizializzano alcune variabili che tengono conto dell'aggiornamento dello schermo.

Test

I test svolti per riuscire ad ottenere un'architettura di rete addestrata in grado di classificare correttamente i vari input, riguardano la compressione dell'informazione, l'apprendimento delle dieci cifre in presenza ed in assenza di rumore e la capacità di generalizzazione a cifre rumorose non note. Il criterio seguito è di partire da un'architettura minimale e di incrementarla se necessario.

3.1 Le variabili *pss* e *tss*

Per capire quant'è l'errore che si sta commettendo nell'addestrare la rete, sono utili le due quantità *pss* e *tss* che rappresentano rispettivamente la somma dei quadrati degli errori ottenuti su ciascun neurone di uscita per un singolo pattern e per tutti i pattern appartenenti ad un'epoca:

$$pss = \tfrac{1}{2} \sum_{j(L)=1}^{n(L)} \left[x_{j(L)}(t) - d_{j(L)}(t) \right]^2$$

$$tss = \tfrac{1}{2} \sum_{t=1}^{T} \sum_{j(L)=1}^{n(L)} \left[x_{j(L)}(t) - d_{j(L)}(t) \right]^2$$

Nel nostro caso il numero di neuroni di output *n(L)* è 20. Supponendo di stabilire una soglia di uscita uguale a 0.5 per cui tutte le uscite inferiori a 0.5 sono assimilate a 0 mentre quelle superiori a 1, vogliamo calcolare quei valori di *pss* per cui certamente ci saranno o meno degli errori nell'apprendimento di un pattern. Il caso più sfortunato che può capitare è quello in cui 19 uscite sono proprio uguali a quelle del target atteso mentre la ventesima vale proprio 0.5. In questo caso si ha che:

$$pss = \tfrac{1}{2}(0.5)^2 = 0.125$$

Il caso più fortunato invece è quello in cui tutte le uscite sono uguali a 0.5. Si ha che:

$$pss = \tfrac{1}{2}(0.5)^2 * 20 = 2.5$$

Per quanto riguarda l'errore su un'intera epoca *tss*, si possono fare solo delle osservazione valide in media e non in assoluto supponendo che la distribuzione degli errori sui vari pattern sia di tipo uniforme. Detto *T* il numero dei pattern in un'epoca, si può affermare che:

- se *pss* ≤ 0.125, allora tutti i pixel del pattern sono stati appresi correttamente;
- se *pss* ≥ 2.5, allora almeno un pixel del pattern è sbagliato;
- se *tss* ≤ 0.125 * *T*, allora *probabilmente* tutti i pattern sono corretti;
- se *tss* ≥ 2.5 * *T*, allora *sicuramente* almeno un pattern non è corretto.

3.2 Compressione dell'informazione

Poiché un autoassociatore fa compressione dell'informazione, vediamo quale codice binario viene associato a ciascuna cifra numerica. Dato che le cifre sono 10, ci vogliono almeno 4 bit per codificarle: usiamo quindi 4 neuroni interni per la rete e vediamo quali sono le uscite di tali unità nascoste dopo che la rete è stata addestrata. Di seguito si riporta il risultato ottenuto, illustrando anche se la rete ha imparato correttamente la cifra e riportando il *pss* corrispondente:

cifra	0	1	2	3	4	5	6	7	8	9
codice	1110	0100	1001	0011	0000	0011	0010	0001	1010	1011
appresa	si	si	si	no	si	no	si	no	si	si
pss	0.099	0.099	0.099	2.280	0.103	1.171	0.098	0.477	0.099	0.099

Il valore finale di *tss* dopo 1500 epoche è di poco superiore a 4.5. Si può osservare che la rete ha troppi pochi parametri per riuscire nel suo compito: le cifre 3 e 5 che non sono correttamente apprese, presentano anche lo stesso codice. Anche addestrando la rete con dei nuovi pesi iniziali, i risultati variano di poco: occorre aumentare il numero dei neuroni nascosti per dare più gradi di libertà al sistema.

3.3 Apprendimento senza rumore

Si cerca adesso di far apprendere alla rete l'insieme delle 10 cifre non disturbate da rumore. Certamente 4 neuroni nascosti non sono sufficienti ad apprendere l'insieme di allenamento: vari test effettuati con 5 neuroni hidden forniscono più o meno gli stessi risultati, con le cifre 3, 5 e 7 che non sono classificate correttamente. Effettuando delle prove con 6 neuroni nascosti, si giunge alla corretta classificazione di tutte le cifre dopo circa 50 epoche. Riportiamo di seguito i dati relativi ad un allenamento di 200 epoche:

cifra	0	1	2	3	4
codice	000101	101100	000000	010010	100100
appresa	si	si	si	si	si
pss	0.034	0.036	0.086	0.036	0.031

cifra	5	6	7	8	9
codice	011100	011101	110000	010111	010001
appresa	si	si	si	si	si
pss	0.026	0.028	0.033	0.024	0.053

Questa volta anche il codice associato alle cifre non è ambiguo.

3.4 Apprendimento con rumore

Vediamo ora di far apprendere alla rete delle cifre rumorose senza preoccuparsi del problema della generalizzazione. Oltre ai 10 pattern non rumorosi, vengono presentati alla rete 100 pattern disturbati da rumore generati con l'apposita funzione che aggiunge, toglie o sposta un pixel. I 100 pattern rumorosi sono ripartiti uniformemente tra le varie cifre.

3.4.1 Rete con 6 neuroni nascosti

I test effettuati su una rete con 6 neuroni hidden, mostrano che tale rete è in grado di apprendere correttamente tutti i pattern ma solo in corrispondenza di certe condizioni iniziali. Si può dunque affermare che tale rete è la più piccola in grado di risolvere tale compito e quindi mostra una certa dipendenza dai parametri iniziali. Per giungere ad un corretto addestramento della rete, occorre innanzitutto eseguire un addestramento on-line della rete per circa 50 epoche fino a che l'errore totale *tss* non è vicino a 10. Un addestramento batch sulle epoche iniziali non consegue l'effetto voluto in quanto l'errore accumulato su un'intera epoca è troppo grande e l'algoritmo di addestramento non riesce a ripartirlo correttamente: l'errore oscilla tra dei valori che sembrano casuali senza mai scendere. Dopo essere giunti ad un errore accettabile con un addestramento on-line, si può applicare un addestramento batch che porta velocemente la rete nella sua condizione finale. Spesso occorre variare il learning rate per uscire da alcuni minimi locali ma nonostante tutto non sempre si arriva ad un corretto apprendimento di tutti i pattern. Quando un pixel non è correttamente appreso, quasi nella totalità dei casi si registra un valore opposto rispetto a quello atteso (questo porta ad un errore superiore ad 1 su quel pattern) e di solito è un uno che viene scambiato per uno zero e non viceversa. Due situazioni molto comuni che si verificano sono le seguenti: se alla fine dell'addestramento si ottiene un valore di *tss* di circa 2.5, molto spesso l'unico pattern sbagliato è il numero 29 che rappresenta un 9 disturbato in cui manca il pixel nell'angolo in basso a sinistra; se invece si giunge ad un valore di *tss* di circa 13 o 14, certamente un pattern è stato appreso male e tutte le volte che compare, rumoroso o meno, incrementa di 1 l'errore totale: spesso è la cifra 3 che dà origine a questa situazione.

3.4.2 Rete con 7 neuroni nascosti

Dato che la rete con 6 neuroni interni è sensibile ai parametri iniziali, vediamo come si comporta una rete con 7 neuroni hidden. Si può subito osservare che scompare completamente la dipendenza dai parametri iniziali ed non occorre più variare neanche il learning rate: qualunque sia la configurazione iniziale, è garantito il completo apprendimento di tutti i pattern. Inoltre solo raramente occorre applicare in cascata un addestramento batch ad uno on-line. Il numero di epoche necessarie per giungere ad un configurazione stabile diminuisce di 4-5 volte, attestandosi a circa 50. Il valore di *tss* può essere reso arbitrariamente piccolo aumentando il numero di epoche.

3.5 Generalizzazione con rumore

Per ultima cosa, cerchiamo di costruire una rete in grado di comportarsi "bene" anche in presenza di ingressi non noti. Si utilizza la cross-validation per evitare il fenomeno dell'overfitting dei dati interrompendo l'apprendimento quando si verifica una tendenza al ribasso della capacità di generalizzazione. Il training set è composto dalle 10 cifre non rumorose e da 100 cifre disturbate da rumore, mentre il test set contiene 50 cifre rumorose. I pattern rumorosi sono ripartiti in modo uniforme tra le varie cifre.

3.5.1 Rete con 7 neuroni nascosti

Iniziamo vedendo come si comporta una rete con 7 neuroni nascosti. Riportiamo di seguito una tabella in cui si evidenziano i valori di *tss* sul training set e sul test set al variare del numero di epoche ottenuti con un addestramento on-line. Tali valori sono stati stampati su un file di tipo *.log* alternando un passo di addestramento ad uno di test:

epoca	0	1	2	3	4	5	6	7	8
training	557,508	251,451	131,851	97,759	69,122	61,058	89,189	45,254	49,058
test	253,412	79,905	44,451	42,512	40,224	17,329	29,498	19,832	21,400

9	10	11	12	13	14	15	16	17	18
27,521	39,370	38,699	21,970	51,094	20,377	10,116	32,402	21,506	13,867
20,722	25,083	42,519	16,082	33,633	19,627	18,860	19,239	23,171	19,564

19	20	21	22	23	24	25	26	27	28
25,861	16,166	16,916	4,444	4,340	4,069	3,555	7,051	3,931	3,873
22,169	23,855	15,881	16,100	15,714	16,303	19,683	7,711	7,894	8,252

29	30	31	32	33	34	35	36	37	38
3,818	3,757	3,628	1,989	6,312	4,078	3,682	3,577	3,553	3,534
9,197	11,582	16,367	20,083	16,244	5,325	5,291	5,315	5,326	5,325

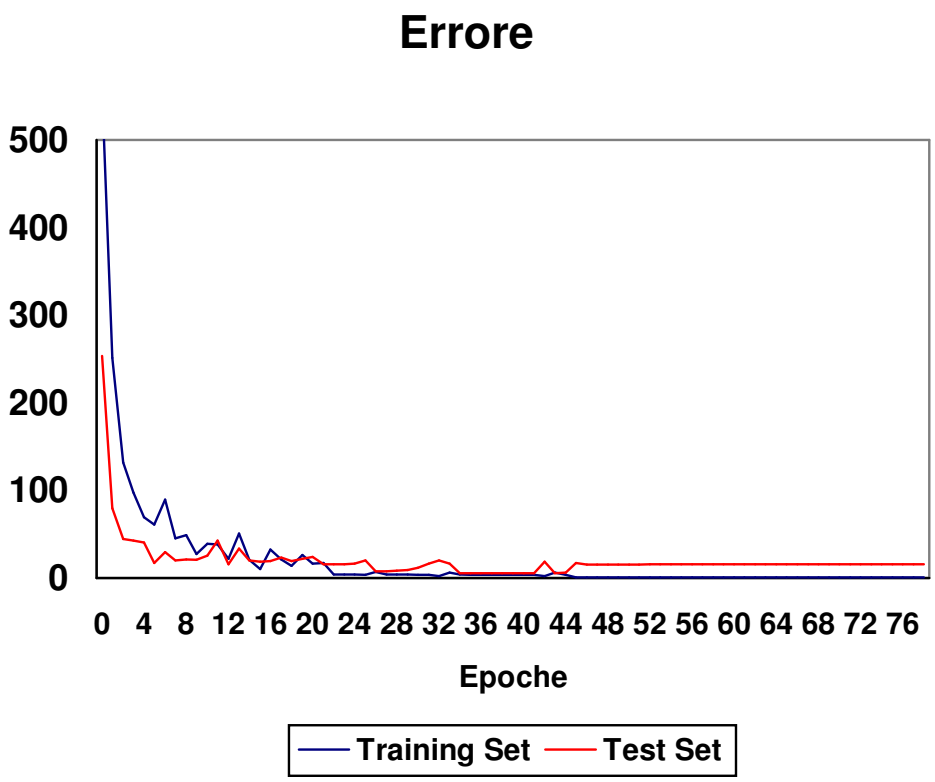
39	40	41	42	43	44	45	46	47	48
3,511	3,479	3,409	2,117	5,941	3,170	0,890	0,703	0,661	0,638
5,321	5,325	5,569	18,809	5,394	6,171	17,425	14,965	15,120	15,229

49	50	51	52	53	54	55	56	57	58
0,619	0,601	0,585	0,570	0,557	0,544	0,531	0,520	0,509	0,498
15,312	15,379	15,435	15,484	15,526	15,564	15,598	15,628	15,656	15,681

59	60	61	62	63	64	65	66	67	68
0,488	0,479	0,470	0,461	0,453	0,444	0,437	0,429	0,422	0,415
15,705	15,726	15,746	15,764	15,781	15,797	15,812	15,825	15,838	15,850

69	70	71	72	73	74	75	76	77	78
0,408	0,402	0,395	0,389	0,383	0,378	0,372	0,367	0,361	0,356
15,861	15,872	15,882	15,891	15,900	15,909	15,917	15,924	15,931	15,938

Ecco il grafico di tali valori:



Si può subito vedere che l'errore sul training set ha un andamento mediamente decrescente, mentre quello sul test set presenta dei minimi e da un certo punto in poi incomincia a crescere. Per quanto riguarda l'apprendimento dell'insieme di addestramento, si vede che il passaggio dall'epoca 44 alla 45 provoca il completo apprendimento di tutto l'insieme e l'errore da questo punto in poi tende in modo monotono a zero (nel grafico non compare più nessuna linea nera). L'errore sul test set raggiunge il suo minimo all'epoca 35 quando ancora non si ha un completo apprendimento del training set (solo il pattern 106, che rappresenta un 6 con un pixel aggiunto a destra che lo fa sembrare un 8, non è stato appreso correttamente) e tende poi asintoticamente a crescere. L'unica cifra del test set che non è stata appresa all'epoca 35 è la 112 che rappresenta un 2 rumoroso in cui il pixel in basso a destra è stato spostato di una casella verso l'alto. All'epoca 50 tutti i 110 pattern di allenamento sono appresi correttamente, mentre 5 pattern rumorosi del test set sono sbagliati. Andando avanti con le epoche, si registra una certa costanza dei valori che tendono a modificarsi molto lentamente (questo conferma la fine dell'apprendimento).

3.5.2 Rete con 8 neuroni nascosti

Vediamo quali risultati si ottengono aggiungendo un neurone nascosto ed effettuando un addestramento on-line:

epoca	0	1	2	3	4	5	6	7	8
training	556,797	248,783	117,939	85,187	87,768	34,031	34,199	20,540	33,999
test	252,910	47,974	50,455	56,976	27,210	26,518	25,700	11,146	21,304

9	10	11	12	13	14	15	16	17	18
21,093	31,638	36,938	24,091	10,533	25,278	44,524	33,870	43,972	53,213
25,156	16,613	22,224	24,106	17,966	24,004	26,305	24,234	26,288	41,498

19	20	21	22	23	24	25	26	27	28
30,157	77,324	63,307	35,742	19,593	20,101	26,128	10,650	6,949	2,009
33,339	34,574	23,534	15,929	18,058	24,326	18,737	12,373	11,644	15,359

29	30	31	32	33	34	35	36	37	38
1,321	1,166	1,095	1,038	0,990	0,947	0,908	0,873	0,841	0,812
14,210	14,273	14,283	14,277	14,264	14,247	14,230	14,213	14,196	14,179

39	40	41	42	43	44	45	46	47	48
0,785	0,760	0,737	0,715	0,694	0,675	0,657	0,640	0,624	0,608
14,164	14,149	14,136	14,123	14,111	14,099	14,089	14,078	14,069	14,060

49	50	51	52	53	54	55	56	57	58
0,594	0,580	0,567	0,554	0,542	0,530	0,519	0,509	0,499	0,489
14,051	14,043	14,035	14,027	14,020	14,013	14,006	13,999	13,992	13,986

59	60	61	62	63	64	65	66	67	68
0,480	0,471	0,462	0,454	0,446	0,438	0,431	0,424	0,417	0,410
13,979	13,973	13,967	13,961	13,955	13,949	13,943	13,937	13,932	13,926

69	70	71	72	73	74	75	76	77	78
0,404	0,397	0,391	0,385	0,380	0,374	0,369	0,363	0,358	0,353
13,920	13,914	13,909	13,903	13,898	13,892	13,887	13,881	13,876	13,871

L'errore sul training set e sul test hanno degli andamenti simili ed a differenza di prima, decrescono asintoticamente entrambi (praticamente raggiungono dei valori costanti). Il training set è correttamente appreso nella sua totalità dopo circa 30 epoche, mentre 5 elementi del test set non vengono classificati correttamente; aumentando il numero di epoche, si scopre che la rete ha raggiunto una configurazione stabile. Tale rete ha probabilmente troppi parametri, mentre quella con 7 neuroni nascosti è quella ottimale: infatti non si verifica neppure un aumento del valore dell'errore sul test set se si continua ad addestrare la rete, cosa invece molto comune se l'architettura di rete ha dimensioni adeguate.

3.6 Conclusioni

Il problema del riconoscimento delle 10 cifre numeriche è stato trattato usando un autoassociatore con 20 ingressi e 20 uscite. Ovviamente questo non è l'unico modo possibile, ma ad esempio poteva essere usata una rete con 10 uscite, una per ogni cifra. Come si è visto, la scelta critica riguarda il numero di neuroni nascosti dell'autoassociatore: se tale numero è superiore o inferiore al numero ottimale, allora nascono dei problemi durante l'apprendimento ed in fase di generalizzazione. A differenza di quanto potrebbe suggerire l'intuizione, 4 o 5 neuroni nascosti non sono sufficienti ad apprendere e codificare correttamente tutte e 10 le cifre. Occorrono 6 neuroni nascosti per imparare le 10 cifre numeriche e per ottenere un codice non ambiguo. Se si hanno anche dei pattern rumorosi, un autoassociatore con 7 neuroni nascosti è l'architettura ideale per riuscire ad avere una buona capacità di apprendimento e di generalizzazione. Non si devono effettuare molte epoche di addestramento in quanto un valore troppo basso dell'errore sul training set porta ad un errore elevato sul test set. Infine, una rete con 8 neuroni nascosti ha troppi gradi di libertà e mostra un comportamento peggiore di quella con 7.

```
/*****************************************************************************

                        GENERATORE DI CARATTERI RUMOROSI

*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <time.h>

/********************************** COSTANTI **********************************/

const int      NRIG =  5;  // numero di righe del carattere
const int      NCOL =  4;  // numero di colonne del carattere
const int NUMCOPPIE = 50;  // numero di coppie patter-target generate

const  char  *const  NOMEFILE  =  "ricnum.pat";     //  file  degli  esempi  per
l'addestramento

/************************** DEFINIZIONI DEI TIPI ******************************/

typedef int Numero[NRIG][NCOL];  // matrice del carattere

typedef struct coppia            // coppia di esempio con nome
{
       char nome[10];
       Numero pattern;
       Numero target;
} Coppia;

typedef Coppia Allenamento[NUMCOPPIE]; // vettore delle coppie di esempio

/******************** NUMERI DA ZERO A NOVE SENZA DISTURBO ********************/

const Numero numeri[10] = {

       {       {0, 1, 1, 0},   /* 0 */
               {1, 0, 0, 1},
               {1, 0, 0, 1},
               {1, 0, 0, 1},
               {0, 1, 1, 0}   },

       {       {0, 0, 1, 0},   /* 1 */
               {0, 1, 1, 0},
               {1, 0, 1, 0},
               {0, 0, 1, 0},
               {0, 0, 1, 0}   },

       {       {0, 1, 1, 0},   /* 2 */
               {1, 0, 0, 1},
               {0, 0, 1, 0},
               {0, 1, 0, 0},
               {1, 1, 1, 1}   },

       {       {1, 1, 1, 1},   /* 3 */
               {0, 0, 0, 1},
               {0, 1, 1, 0},
               {0, 0, 0, 1},
               {1, 1, 1, 1}   },

       {       {0, 0, 1, 0},   /* 4 */
               {0, 1, 1, 0},
               {1, 0, 1, 0},
               {1, 1, 1, 1},
               {0, 0, 1, 0}   },

       {       {1, 1, 1, 1},   /* 5 */
               {1, 0, 0, 0},
               {1, 1, 1, 0},
               {0, 0, 0, 1},
               {1, 1, 1, 0}   },
```

```
        {       {0, 1, 1, 1},   /* 6 */
                {1, 0, 0, 0},
                {1, 1, 1, 0},
                {1, 0, 0, 1},
                {0, 1, 1, 0}   },

        {       {1, 1, 1, 1},   /* 7 */
                {0, 0, 0, 1},
                {0, 0, 1, 0},
                {0, 1, 0, 0},
                {1, 0, 0, 0}   },

        {       {0, 1, 1, 0},   /* 8 */
                {1, 0, 0, 1},
                {0, 1, 1, 0},
                {1, 0, 0, 1},
                {0, 1, 1, 0}   },

        {       {0, 1, 1, 0},   /* 9 */
                {1, 0, 0, 1},
                {0, 1, 1, 1},
                {0, 0, 0, 1},
                {1, 1, 1, 0}   },
};

/********************** STAMPA DEI NUMERI DA ZERO A NOVE ************************/

void stampaNumeri(void)
{
        for (int i = 0; i < 10; ++i)
        {
                for (int n = 0; n < NRIG; ++n)
                {
                        for (int m = 0; m < NCOL; ++m)
                                printf("%d ", numeri[i][n][m]);
                        printf("\n");
                }
                printf("\n");
                getch();
        }
}

/********************** COPIA DI UN NUMERO IN UN ALTRO ************************/

void copiaNumero(numero destinazione, const Numero origine)
{
        for (int i = 0; i < NRIG; ++i)
                for (int j = 0; j < NCOL; ++j)
                        destinazione[i][j] = origine[i][j];
}

/********************** AGGIUNTA DEL RUMORE AD UN CARATTERE ************************/

/*
        ci sono tre modi per aggiungere del rumore ad un carattere:
        1) spostare una casella nera da un posto in un altro
        2) aggiungere una casella nera
        3) eliminare una casella nera
*/

void rumore(const Numero originale, Numero disturbato)
{
        copiaNumero(disturbato, originale);

        time_t t;
        srand((unsigned) time(&t));

        int rig;
        int col;

        int r = rand() % 3;

        if (r == 0) // se r è 0, si sposta una casella nera
        {
                do
                {
                        rig = rand() % NRIG;
```

```
                        col = rand() % NCOL;
                }
                while (originale[rig][col] == 0);

                disturbato[rig][col] = 0; // la casella nera diventa bianca

                do
                {
                        rig = rand() % NRIG;
                        col = rand() % NCOL;
                }
                while (originale[rig][col] == 1);

                disturbato[rig][col] = 1; // la casella bianca diventa nera
        }

        if (r == 1) // se r è 1, si aggiunge una casella nera
        {
                do
                {
                        rig = rand() % NRIG;
                        col = rand() % NCOL;
                }
                while (originale[rig][col] == 1);

                disturbato[rig][col] = 1; // la casella bianca diventa nera
        }

        if (r == 2) // se r è 2, si toglie una casella nera
        {
                do
                {
                        rig = rand() % NRIG;
                        col = rand() % NCOL;
                }
                while (originale[rig][col] == 0);

                disturbato[rig][col] = 0; // la casella nera diventa bianca
        }
}

/******************** CREAZIONE DELL'INSIEME DI ALLENAMENTO **********************/

void creaAllenamento(Allenamento all)
{
        for (int i = 0; i < 10; ++i) // le prime dieci coppie sono senza rumore
        {
                char nome[10];
                sprintf(nome, "p%d", i);
                strcpy(all[i].nome, nome);
                copiaNumero(all[i].pattern, numeri[i]);
                copiaNumero(all[i].target , numeri[i]);
        }

        for (i = 10; i < NUMCOPPIE; ++i) // seguono poi le coppie con pattern rumo-
rosi
        {
                char nome[10];
                sprintf(nome, "p%d", i);
                strcpy(all[i].nome, nome);
                Numero disturbato;
                rumore(numeri[i % 10], disturbato);
                copiaNumero(all[i].pattern, disturbato);
                copiaNumero(all[i].target , numeri[i % 10]);
        }
}

/********************** STAMPA DELL'INSIEME DI ALLENAMENTO ************************/

void stampaAllenamento(Allenamento all)
{
        for (int i = 0; i < NUMCOPPIE; ++i)
        {
                printf("%d) %s\n", i + 1, all[i].nome); // stampa il nome
                for (int n = 0; n < NRIG; ++n)          // stampa il pattern
                {
                        for (int m = 0; m < NCOL; ++m)
                                printf("%d ", all[i].pattern[n][m]);
```

```
                        printf("\n");
                }
                printf("\n");
                for (n = 0; n < NRIG; ++n)  // stampa il target
                {
                        for (int m = 0; m < NCOL; ++m)
                                printf("%d ", all[i].target[n][m]);
                        printf("\n");
                }
                getch();
        }
}

/******************* CREAZIONE DEL FILE PER L'ALLENAMENTO *********************/

void salvaAllenamento(const Allenamento all)
{
        FILE *fp;

        if ((fp = fopen(NOMEFILE, "w")) == NULL)
        {
                fprintf(stderr, "\n Non posso scrivere su %s", NOMEFILE);
                exit(1);
        }
        else
        {
                for (int i = 0; i < NUMCOPPIE; ++i)
                {
                        fprintf (fp, "%s  ", all[i].nome);  // salva il nome
                        for (int n = 0; n < NRIG; ++n)      // salva il pattern
                        {
                                for (int m = 0; m < NCOL; ++m)
                                        fprintf (fp, "%d ", all[i].pattern[n][m]);
                                fprintf(fp, " ");
                        }
                        fprintf(fp, "  ");
                        for (n = 0; n < NRIG; ++n)  // salva il target
                        {
                                for (int m = 0; m < NCOL; ++m)
                                        fprintf (fp, "%d ", all[i].target[n][m]);
                                fprintf(fp, " ");
                        }
                        fprintf(fp, "\n");
                }

                if (ferror(fp))
                {
                        fprintf(stderr, "\n Errore di scrittura");
                        fprintf(stderr, " su %s", NOMEFILE);
                        exit(2);
                }
                fclose(fp);
        }
}

/*************************** PROGRAMMA PRINCIPALE ****************************/

void main(void)
{
        clrscr();
        stampaNumeri();
        Allenamento allena;
        creaAllenamento(allena);
        stampaAllenamento(allena);
        salvaAllenamento(allena);
}

/************************  FINE PROGRAMMA PRINCIPALE ***************************/
```

UNIVERSITA' DEGLI STUDI DI FIRENZE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

Esercitazioni per l'esame di
"Intelligenza Artificiale"
Prof. G. Soda

Bayesian Networks

di Sauro Menchetti

A.A.1998-99

Costruzione della Rete Bayesiana

Le reti bayesiane sono uno strumento utile in quelle situazioni in cui non è possibile agire con certezza. Un problema della logica del primo ordine è che non si ha quasi mai accesso all'intera verità: un fatto non è vero o falso in assoluto. Nella maggioranza dei casi, anche in situazioni semplici, ci saranno domande importanti a cui non si potrà dare una risposta categorica. Si deve quindi agire in presenza di incertezza. L'incertezza è causata principalmente da dati inaffidabili, mancanti o imprecisi.

1.1 Situazione da modellare

Qualsiasi studente iscritto ad una facoltà, per poter giungere alla laurea, deve superare tutti gli esami previsti dal proprio piano di studi. Lo studente può scegliere di frequentare o meno un determinato corso a seconda del tempo che ha a disposizione e dell'interesse verso la materia insegnata: se decide di frequentare il corso, sarà più facile superare l'esame. Quando lo studente decide di sostenere l'esame, si suppone che abbia una sufficiente padronanza della materia: il tempo a sua disposizione e la voglia di studiare incidono sulla preparazione globale e quest'ultima, insieme alla fortuna (che non guasta mai), possono determinare l'esito dell'esame.

1.2 Analisi del dominio

Si analizzano le caratteristiche salienti del dominio in esame che porteranno alla costruzione della rete bayesiana.

1.2.1 Variabili di dominio

Da un'analisi della precedente situazione, si possono ricavare le seguenti variabili aleatorie con i rispettivi stati:

- E = superamento dell'esame (no = e_0, si = e_1);
- F = frequenza del corso (scarsa = f_0, assidua = f_1);
- T = tempo a disposizione (insufficiente = t_0, sufficiente = t_1);
- I = interesse per la materia (scarso = i_0, sufficiente = i_1, molto = i_2);
- P = preparazione (insufficiente = p_0, sufficiente = p_1, buona = p_2);

V = voglia di studiare (scarsa = v_0, media = v_1, molta = v_2);
C = fortuna (no = c_0, si = c_1).

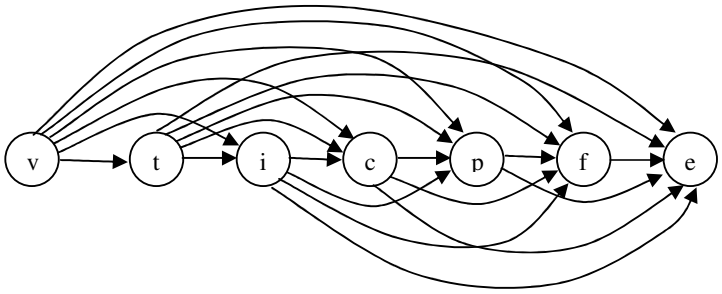
Gli indici assegnati ai vari stati servono a sostituire il nome dello stato della variabile aleatoria nelle tabelle che seguiranno.

1.2.2 Ordinamento delle variabili

Per costruire la rete bayesiana corrispondente alla precedente situazione, occorre dare un ordinamento alle variabili: l'ordinamento è cruciale, perché può indurre a ricercare condizioni di dipendenza non semplicemente esplicitabili. Tuttavia qualunque ordinamento garantisce la rappresentazione del dominio. La rete finale dipende comunque dall'ordinamento delle variabili. L'ordinamento scelto è il seguente:

$$V, T, I, C, P, F, E.$$

Come si può osservare, le variabili che non presentano dipendenze dirette con altre variabili vengono prima nell'ordinamento, seguite poi dalle rimanenti. La precedente sequenza di variabili corrisponde ad un ordinamento topologico del grafo diretto aciclico.



In un ordinamento topologico tutti i discendenti sono alla destra del nodo padre; inoltre ogni nodo è collegato con tutti i suoi discendenti in modo opportuno. Tale ordinamento topologico deve essere semplificato con le condizioni rilevate dal dominio.

1.2.3 Condizioni rilevate dal dominio

Esaminando il dominio da modellare, si possono rilevare le seguenti condizioni di dipendenza e di indipendenza:

$$V \perp \{T, I, C\} \mid \emptyset \text{ ed } E \perp \{V, T, I\} \mid \{P, F\}$$

cioè le variabili V, T, I, C sono indipendenti, mentre E dipende solo da V, T, I date P, F.

1.2.4 Legami tra le variabili

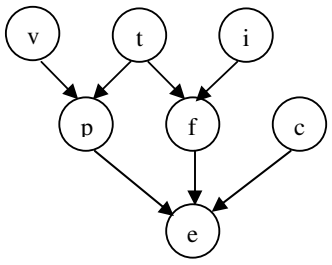
Rispettando l'ordine imposto alle variabili, si possono determinare quali variabili influenzano direttamente un'altra variabile:

variabili	dipendenza
V	$P(V)$;
T	$P(T \mid V) = P(T)$;
I	$P(I \mid T, V) = P(I)$;
C	$P(C \mid I, T, V) = P(C)$;
P	$P(P \mid C, I, T, V) = P(P \mid T, V)$;
F	$P(F \mid P, C, I, T, V) = P(F \mid I, T)$;
E	$P(E \mid F, P, C, I, T, V) = P(E \mid F, P, C)$.

Come si vede, non tutti i legami dell'ordinamento topologico sono effettivamente presenti.

1.3 La Rete Bayesiana

Dalla precedente analisi, eliminando gli archi superflui dall'ordinamento topologico, si giunge alla seguente rete bayesiana dove le frecce indicano delle dipendenze dirette tra le variabili:



Le variabili V e T influenzano la variabile P, così come le variabili T ed I influenzano P. Inoltre la variabile E è influenzata direttamente da P, F e C.

1.3.1 Le tabelle di probabilità

Si riportano di seguito le tabelle di probabilità associate alla rete di Bayes. Vediamo prima le tabelle di probabilità a priori delle variabili V, I, T e C.

Voglia di studiare	$P(V)$
Scarsa	0.1
Media	0.2
Molta	0.7

Interesse	$P(I)$
Scarso	0.1
Sufficiente	0.3
Molto	0.6

Tempo a disposizione	$P(T)$
Insufficiente	0.3
Sufficiente	0.7

Fortuna	$P(C)$
No	0.5
Si	0.5

Ecco infine le probabilità condizionate delle variabili P, F ed E.

Preparazione		$P(P \mid T, V)$		
Tempo	Voglia	Insufficiente	Sufficiente	Buona
Insufficiente	Scarsa	0.85	0.1	0.05
Insufficiente	Media	0.7	0.2	0.1
Insufficiente	Molta	0.5	0.3	0.2
Sufficiente	Scarsa	0.5	0.4	0.1
Sufficiente	Media	0.2	0.6	0.2
Sufficiente	Molta	0.02	0.08	0.9

Frequenza del corso		$P(F \mid I, T)$	
Interesse	Tempo	Scarsa	Assidua
Scarso	Insufficiente	0.8	0.2
Scarso	Sufficiente	0.65	0.35
Sufficiente	Insufficiente	0.7	0.3
Sufficiente	Sufficiente	0.35	0.65
Molto	Insufficiente	0.55	0.45
Molto	Sufficiente	0.1	0.9

Superamento dell'esame			$P(E \mid F, P, C)$	
Frequenza	Preparazione	Fortuna	No	Si
Scarsa	Insufficiente	No	0.95	0.05
Scarsa	Insufficiente	Si	0.9	0.1
Scarsa	Sufficiente	No	0.3	0.7
Scarsa	Sufficiente	Si	0.25	0.75
Scarsa	Buona	No	0.15	0.85
Scarsa	Buona	Si	0.1	0.9
Assidua	Insufficiente	No	0.9	0.1
Assidua	Insufficiente	Si	0.8	0.2
Assidua	Sufficiente	No	0.2	0.8
Assidua	Sufficiente	Si	0.1	0.9
Assidua	Buona	No	0.05	0.95
Assidua	Buona	Si	0.03	0.97

Costruzione del Junction Tree

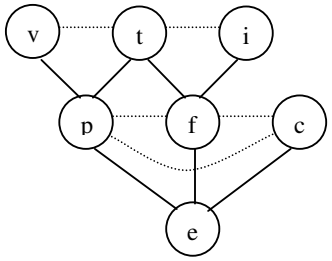
Le reti bayesiane non sono adatte per fare inferenza sul dominio di interesse: le tabelle di probabilità crescono in modo esponenziale e diventano presto intrattabili. Bisogna quindi trovare un tipo di rappresentazione più conveniente: il junction tree risolve il problema.

2.1 Il Junction Tree

Ci sono vari passi che portano alla costruzione del junction tree: esaminiamoli di seguito uno ad uno.

2.1.1 Costruzione del grafo moralizzato

Il grafo moralizzato viene costruito a partire dalla rete di Bayes: per ogni variabile A si crea un collegamento tra tutte le variabili in $\text{pr}(A) \bigcup \{A\}$. Si ottiene il seguente grafo non diretto:



Gli archi tratteggiati indicano i link di moralizzazione.

Osservazione: la rete di Bayes originaria è un grafo a connessione multipla. Tuttavia non occorre triangolarizzare il grafo moralizzato in quanto i link di moralizzazione svolgono tale compito.

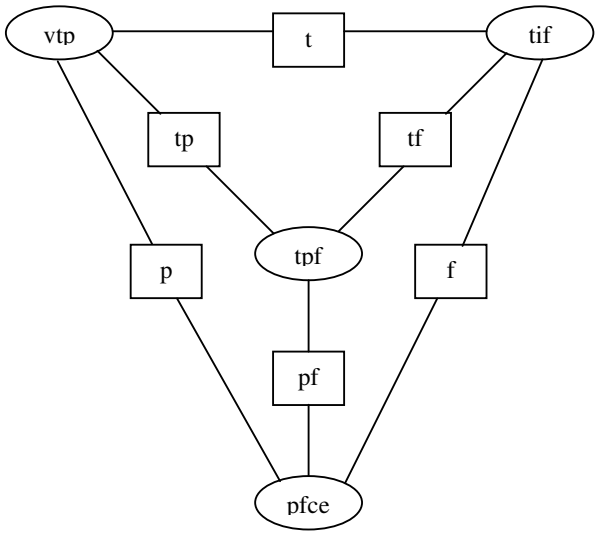
2.1.2 Individuazione delle cricche

In questa fase si cerca di individuare tutte le cricche del grafo moralizzato che andranno a costituire i nodi del junction tree. Si identificano le seguenti cricche:

che serviranno per il passo successivo.

2.1.3 Costruzione del Junction Graph

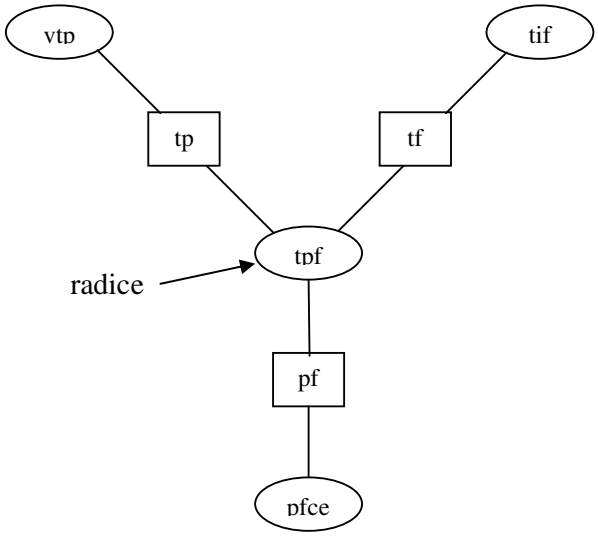
I nodi del junction graph sono le cricche precedentemente individuate: tra ogni coppia di nodi che hanno almeno una variabile in comune, si mette un link con un separatore che contiene l'intersezione delle variabili. Ecco cosa si ottiene:



I nodi sono indicati con delle ellissi, mentre i separatori con dei rettangoli.

2.1.4 Costruzione del Junction Tree

Dato il junction graph, si devono eliminare i link che costituiscono dei cicli in modo da ottenere un albero. Nell'eliminazione, vengono mantenuti i link a massima intersezione. Si ottiene il seguente albero:



Come si può vedere, per ogni coppia di nodi V, W tutti i nodi del cammino che li congiunge contengono l'intersezione di $V \bigcap W$.

2.2 Assegnazione delle tabelle al Junction Tree

Il passo successivo assegna le tabelle di probabilità ad ogni nodo ed ad ogni separatore del junction tree. È suddiviso in due fasi: una prima in cui si inizializzano le tabelle ed una seconda in cui modificano in modo iterativo i valori contenuti in esse.

2.2.1 Inizializzazione delle tabelle

Per prima cosa si inizializzano tutte le tabelle dei nodi e dei separatori ad uno:

tabelle dei nodi	tabelle dei separatori
$t_{VTP} = 1$	$t_{TP} = 1$
$t_{TIF} = 1$	$t_{TF} = 1$
$t_{TPF} = 1$	$t_{PF} = 1$
$t_{PFCE} = 1$	

Gli indici indicano il nodo o il separatore a cui si riferisce la tabella.

2.2.2 Assegnazione delle tabelle

Per ciascuna variabile *A*, si sceglie un nodo V del junction tree contenente $pr(A) \bigcup \{A\}$ (un tale nodo esiste per come è stato costruito l'albero) e si moltiplica $P(A \mid pr(A))$ per la tavola di V: questa tabella aggiornata diventa la nuova tabella di V (pr è l'abbreviazione di parent e sta ad indicare i genitori di un nodo). Le variabili vengono scelte nel seguente ordine: *V*, *T*, *I*, *C*, *P*, *F*, *E*. Esaminiamo cosa succede passo per passo.

Variabile *V*
parent: nessuno
nodo: VTP
$t_{VTP} = P(V) * 1$

tabelle dei nodi	tabelle dei separatori
$t_{VTP} = P(V)$	$t_{TP} = 1$
$t_{TIF} = 1$	$t_{TF} = 1$
$t_{TPF} = 1$	$t_{PF} = 1$
$t_{PFCE} = 1$	

Variabile *T*
parent: nessuno
nodo: TPF
$t_{TPF} = P(T) * 1$

tabelle dei nodi	tabelle dei separatori
$t_{VTP} = P(V)$	$t_{TP} = 1$
$t_{TIF} = 1$	$t_{TF} = 1$
$t_{TPF} = P(T)$	$t_{PF} = 1$
$t_{PFCE} = 1$	

Variabile *I*
parent: nessuno
nodo: TIF
$t_{TIF} = P(I) * 1$

tabelle dei nodi	tabelle dei separatori
$t_{VTP} = P(V)$	$t_{TP} = 1$
$t_{TIF} = P(I)$	$t_{TF} = 1$
$t_{TPF} = P(T)$	$t_{PF} = 1$
$t_{PFCE} = 1$	

Variabile *C*
parent: nessuno
nodo: PFCE
$t_{PFCE} = P(C) * 1$

tabelle dei nodi	tabelle dei separatori
$t_{VTP} = P(V)$	$t_{TP} = 1$
$t_{TIF} = P(I)$	$t_{TF} = 1$
$t_{TPF} = P(T)$	$t_{PF} = 1$
$t_{PFCE} = P(C)$	

Variabile *P*
parent: VT
nodo: VTP
$t_{VTP} = P(P \mid V, T)\ P(V) = P(P, V \mid T)$

tabelle dei nodi	tabelle dei separatori
$t_{VTP} = P(P, V \mid T)$	$t_{TP} = 1$
$t_{TIF} = P(I)$	$t_{TF} = 1$
$t_{TPF} = P(T)$	$t_{PF} = 1$
$t_{PFCE} = P(C)$	

	$P(P, V \mid T)$								
T	$p_0\ v_0$	$p_0\ v_1$	$p_0\ v_2$	$p_1\ v_0$	$p_1\ v_1$	$p_1\ v_2$	$p_2\ v_0$	$p_2\ v_1$	$p_2\ v_2$
t_0	0.085	0.14	0.35	0.01	0.04	0.21	0.005	0.02	0.14
t_1	0.05	0.04	0.014	0.04	0.12	0.056	0.01	0.04	0.63

La somma delle due righe della tabella fa uno.

Variabile *F*
parent: TI

nodo: TIF
$t_{TIF} = P(F \mid T, I)\ P(I) = P(F, I \mid T)$

tabelle dei nodi		tabelle dei separatori	
t_{VTP}	$= P(P, V \mid T)$	t_{TP}	$= 1$
t_{TIF}	$= P(F, I \mid T)$	t_{TF}	$= 1$
t_{TPF}	$= P(T)$	t_{PF}	$= 1$
t_{PFCE}	$= P(C)$		

	$P(F, I \mid T)$					
T	$f_0\ i_0$	$f_0\ i_1$	$f_0\ i_2$	$f_1\ i_0$	$f_1\ i_1$	$f_1\ i_2$
t_0	0.08	0.21	0.33	0.02	0.09	0.27
t_1	0.065	0.105	0.06	0.035	0.195	0.54

La somma delle due righe della tabella fa uno.

Variabile *E*
parent: PFC
nodo: PFCE
$t_{PFCE} = P(E \mid P, F, C)\ P(C) = P(E, C \mid P, F)$

tabelle dei nodi		tabelle dei separatori	
t_{VTP}	$= P(P, V \mid T)$	t_{TP}	$= 1$
t_{TIF}	$= P(F, I \mid T)$	t_{TF}	$= 1$
t_{TPF}	$= P(T)$	t_{PF}	$= 1$
t_{PFCE}	$= P(E, C \mid P, F)$		

		$P(E, C \mid P, F)$			
P	F	$e_0\ c_0$	$e_0\ c_1$	$e_1\ c_0$	$e_1\ c_1$
p_0	f_0	0.475	0.45	0.025	0.05
p_0	f_1	0.45	0.4	0.05	0.1
p_1	f_0	0.15	0.125	0.35	0.375
p_1	f_1	0.1	0.05	0.4	0.45
p_2	f_0	0.075	0.05	0.425	0.45
p_2	f_1	0.025	0.015	0.475	0.485

La somma delle sei righe della tabella fa uno.

Come si vede, tutte le tabelle dei separatori sono rimaste inizializzate ad uno: questo perché, quando iniziamo a muovere informazione sul junction tree, il prodotto di tutte le tabelle dei nodi diviso il prodotto di tutte le tabelle dei separatori è invariante. Infatti:

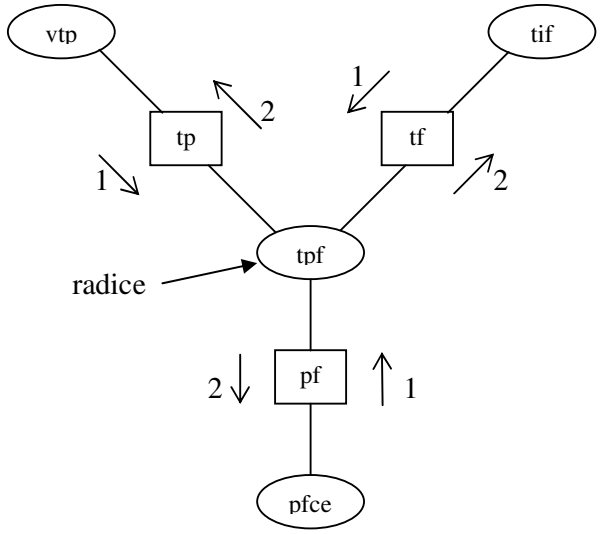
$$
\begin{aligned}
P(U) &= P(V, T, I, C, P, F, E) \\
&= (t_{VTP}\ t_{TIF}\ t_{TPF}\ t_{PFCE}) \,/\, (t_{TP}\ t_{TF}\ t_{PF}) \\
&= P(V)\ P(T)\ P(I)\ P(C)\ P(P \mid V, T)\ P(F \mid T, I)\ P(E \mid P, F, C) \\
&= P(P, V \mid T)\ P(T)\ P(F, I \mid T)\ P(E, C \mid P, F).
\end{aligned}
$$

2.3 Consistenza nel Junction Tree

Dato il precedente junction tree con tutte le tabelle di probabilità dei nodi e dei separatori, il passo successivo consiste nel renderlo globalmente consistente. In un junction tree la consistenza locale implica la consistenza globale. Un junction tree si dice localmente consistente se per ogni coppia di nodi contigui V, W si verifica che:

$$\sum_{V \setminus S} t_V = t_S = \sum_{W \setminus S} t_W$$

dove S rappresenta il nodo separatore contenente l'intersezione $V \bigcap W$; ciò significa che V e W contengono la stessa informazione su S e il link tra i due nodi è detto consistente. Per rendere consistente un link, occorre far transitare un messaggio in entrambe le direzioni. Si usa il seguente schema di trasmissione di messaggi:



con le seguenti tabelle di probabilità dei nodi e dei separatori:

tabelle dei nodi		tabelle dei separatori	
t_{VTP}	$= P(P, V \mid T)$	t_{TP}	$= 1$
t_{TIF}	$= P(F, I \mid T)$	t_{TF}	$= 1$
t_{TPF}	$= P(T)$	t_{PF}	$= 1$
t_{PFCE}	$= P(E, C \mid P, F)$		

Come si vede, ogni link è percorso in entrambe le direzioni e questo ne garantisce la consistenza.

2.3.1 Trasmissione del messaggio da VTP a TPF

1) $t_{TP}^* = \sum_V P(P,V \mid T) = P(P \mid T)$

2) $t_{TPF}^* = P(T)\, P(P \mid T) = P(T, P)$

tabelle dei nodi	tabelle dei separatori
$t_{VTP} = P(P, V \mid T)$	$t_{TP} = P(P \mid T)$
$t_{TIF} = P(F, I \mid T)$	$t_{TF} = 1$
$t_{TPF} = P(T, P)$	$t_{PF} = 1$
$t_{PFCE} = P(E, C \mid P, F)$	

	$P(P \mid T)$		
T	p_0	p_1	p_2
t_0	0.575	0.26	0.165
t_1	0.104	0.216	0.68

	$P(T, P)$		
T	p_0	p_1	p_2
t_0	0.1725	0.078	0.0495
t_1	0.0728	0.1512	0.476

2.3.2 Trasmissione del messaggio da TIF a TPF

1) $t_{TF}^* = \sum_I P(F, I \mid T) = P(F \mid T)$

2) $t_{TPF}^* = P(T, P)\, P(F \mid T) = P(P \mid T)\, P(T)\, P(F \mid T) = P(T, P, F)$

tabelle dei nodi	tabelle dei separatori
$t_{VTP} = P(P, V \mid T)$	$t_{TP} = P(P \mid T)$
$t_{TIF} = P(F, I \mid T)$	$t_{TF} = P(F \mid T)$
$t_{TPF} = P(T, P, F)$	$t_{PF} = 1$
$t_{PFCE} = P(E, C \mid P, F)$	

	$P(F \mid T)$	
T	f_0	f_1
t_0	0.62	0.38
t_1	0.23	0.77

		$P(T, P, F)$		
T	F	p_0	p_1	p_2
t_0	f_0	0.10695	0.04836	0.03069
t_0	f_1	0.06555	0.02964	0.01881
t_1	f_0	0.016744	0.034776	0.10948
t_1	f_1	0.056056	0.116424	0.36652

2.3.3 Trasmissione del messaggio da PFCE a TPF

1) $t_{PF}^* = \sum_{EC} P(E, C \mid P, F) = 1$

2) $t_{TPF}^* = P(T, P, F)$

non cambia nessuna tabella

tabelle dei nodi		tabelle dei separatori	
t_{VTP}	$= P(P, V \mid T)$	t_{TP}	$= P(P \mid T)$
t_{TIF}	$= P(F, I \mid T)$	t_{TF}	$= P(F \mid T)$
t_{TPF}	$= P(T, P, F)$	t_{PF}	$= 1$
t_{PFCE}	$= P(E, C \mid P, F)$		

2.3.4 Trasmissione del messaggio da TPF a VTP

1) $t_{TP}^* = \sum_F P(T, P, F) = P(T, P)$

2) $t_{VTP}^* = P(P, V \mid T)\ P(T, P) / P(P \mid T) = P(P, V \mid T)\ P(P \mid T)\ P(T) / P(P \mid T) =$
$= P(P, V \mid T)\ P(T) = P(V, T, P)$

tabelle dei nodi		tabelle dei separatori	
t_{VTP}	$= P(V, T, P)$	t_{TP}	$= P(T, P)$
t_{TIF}	$= P(F, I \mid T)$	t_{TF}	$= P(F \mid T)$
t_{TPF}	$= P(T, P, F)$	t_{PF}	$= 1$
t_{PFCE}	$= P(E, C \mid P, F)$		

	$P(T, P)$		
T	p_0	p_1	p_2
t_0	0.1725	0.078	0.0495
t_1	0.0728	0.1512	0.476

		$P(V, T, P)$		
T	V	p_0	p_1	p_2
t_0	v_0	0.0255	0.003	0.0015
t_0	v_1	0.042	0.012	0.006
t_0	v_2	0.105	0.063	0.042
t_1	v_0	0.035	0.028	0.007
t_1	v_1	0.028	0.084	0.028
t_1	v_2	0.0098	0.0392	0.441

2.3.5 Trasmissione del messaggio da TPF a TIF

1) $t_{TF}^* = \sum_P P(T, P, F) = P(T, F)$

2) $t_{TIF}^* = P(F, I \mid T)\ P(T, F)\ P(F \mid T) = P(F, I \mid T)\ P(F \mid T)\ P(T) / P(F \mid T) =$
$= P(F, I \mid T)\ P(T) = P(T, I, F)$

tabelle dei nodi		tabelle dei separatori	
t_{VTP}	$= P(V, T, P)$	t_{TP}	$= P(T, P)$
t_{TIF}	$= P(T, I, F)$	t_{TF}	$= P(T, F)$
t_{TPF}	$= P(T, P, F)$	t_{PF}	$= 1$
t_{PFCE}	$= P(E, C \mid P, F)$		

	$P(T, F)$	
T	f_0	f_1
t_0	0.186	0.114
t_1	0.161	0.539

		$P(T, I, F)$		
T	F	i_0	i_1	i_2
t_0	f_0	0.024	0.063	0.099
t_0	f_1	0.006	0.027	0.081
t_1	f_0	0.0455	0.0735	0.042
t_1	f_1	0.0245	0.1365	0.378

2.3.6 Trasmissione del messaggio da TPF a PFCE

1) $t_{PF}^* = \sum_T P(T, P, F) = P(P, F)$

2) $t_{PFCE}^* = P(E, C \mid P, F)\, P(P, F) = P(P, F, C, E)$

tabelle dei nodi

$t_{VTP} = P(V, T, P)$

$t_{TIF} = P(T, I, F)$

$t_{TPF} = P(T, P, F)$

$t_{PFCE} = P(P, F, C, E)$

tabelle dei separatori

$t_{TP} = P(T, P)$

$t_{TF} = P(T, F)$

$t_{PF} = P(P, F)$

	$P(P, F)$	
P	f_0	f_1
p_0	0.123694	0.121606
p_1	0.083136	0.146064
p_2	0.14017	0.38533

		$P(P, F, C, E)$			
P	F	$e_0\, c_0$	$e_0\, c_1$	$e_1\, c_0$	$e_1\, c_1$
p_0	f_0	0.05875465	0.0556623	0.00309235	0.0061847
p_0	f_1	0.0547227	0.0486424	0.0060803	0.0121606
p_1	f_0	0.0124704	0.010392	0.0290976	0.031176
p_1	f_1	0.0146064	0.0073032	0.0584256	0.0657288
p_2	f_0	0.01051275	0.0070085	0.05957225	0.0630765
p_2	f_1	0.00963325	0.00577995	0.18303175	0.18688505

Osservazione: ogni tabella di ogni nodo e di ogni separatore contiene la probabilità congiunta delle variabili che si trovano nel nodo e nel separatore.

Inferenza

Fare inferenza su una rete bayesiana significa dare evidenza ad alcune variabili ed osservare come si modificano le rimanenti. Si possono fare tre tipi di inferenza: il ragionamento causale in cui, data la causa, si cerca la probabilità dell'effetto, il ragionamento diagnostico in cui, dato un effetto e sapendo che può essere provocato da una certa causa, si cerca di determinare la probabilità di tale causa e l'explaining away in cui, dato un certo effetto e sapendo che può essere provocato da due cause, il manifestarsi di una rende meno probabile l'altra. In generale la procedura di inferenza è esponenziale: se m è il numero di stati e k il numero di variabili, la complessità è $O(m^k)$.

3.1 Inferenza senza evidenza

Dato il precedente junction tree consistente senza alcuna evidenza, si calcolano le seguenti probabilità non presenti tra le probabilità assegnate alla rete bayesiana.

$$P(P) = \sum_T P(T,P) = (p_0 = 0.2453\ p_1 = 0.2292\ p_2 = 0.5255)$$
$$P(F) = \sum_T P(T,F) = (f_0 = 0.347\ f_1 = 0.653)$$
$$P(E) = \sum_{PFC} P(P,F,C,E) = (e_0 = 0.2954885\ e_1 = 0.7045115)$$

Dopo aver calcolato queste tre probabilità, si hanno tutte le probabilità delle sette variabili: è interessante vedere come variano tali valori quando viene inserita l'evidenza su alcune variabili.

3.2 Inserimento dell'evidenza

Dato il precedente junction tree consistente, si dà evidenza ad alcune variabili e si vuole vedere come variano le probabilità delle altre. Innanzitutto premettiamo la seguente definizione:

Definizione: Un finding su una variabile aleatoria A con n stati è un vettore n-dimensionale di zero e di uno: indica quali stati di una variabile sono impossibili.

C'è un teorema che gestisce l'inserimento dell'evidenza:

Teorema: Sia BN una rete bayesiana rappresentante $P(U)$ e sia T il junction tree corrispondente a BN. Sia $e = \{f_1,\ldots,f_m\}$ il finding sulle variabili $\{A_1,\ldots,A_m\}$. Per ogni i trovare un nodo contenente A_i e si moltiplichi la sua tabella per f_i. Dopo un trasferimento completo di messaggi si ha:

$$t_V = P(V, e) \qquad t_S = P(S, e) \qquad P(e) = \sum_V t_V \qquad P(W \mid e) = P(W, e) / P(e)$$

per ogni nodo V e per ogni separatore S (W indica V oppure S).

3.2.1 Assegnazione dell'evidenza

Si assegna evidenza alla variabile aleatoria E che indica superamento o meno dell'esame: tale variabile contiene i due stati no = e_0 e si = e_1. Il finding su tale variabile è $e = (0, 1)$ con $P(e) = 0.7045115$. L'unico nodo del junction tree che contiene la variabile E è PFCE, quindi si moltiplica la tabella di PFCE per e, ottenendo la seguente tabella:

		$P(P, F, C, E, e)$			
P	F	$e_0\ c_0$	$e_0\ c_1$	$e_1\ c_0$	$e_1\ c_1$
p_0	f_0	0	0	0.00309235	0.0061847
p_0	f_1	0	0	0.0060803	0.0121606
p_1	f_0	0	0	0.0290976	0.031176
p_1	f_1	0	0	0.0584256	0.0657288
p_2	f_0	0	0	0.05957225	0.0630765
p_2	f_1	0	0	0.18303175	0.18688505

Se i valori della tabella vengono divisi per $P(e)$, la somma fa di nuovo uno.

3.3 Propagazione dell'evidenza

Dopo aver dato evidenza ai nodi del junction tree, tale evidenza deve essere propagata agli altri nodi. Per fare questo si usa lo schema di propagazione di Hugin che consiste nei seguenti passi:

1) Si inserisce l'evidenza all'interno di un nodo a seconda della variabile scelta
2) Si seleziona un nodo radice R
3) Si esegue Collect Evidence a partire dalla radice R
4) Si esegue Distribute Evidence a partire dalla radice R
5) Si effettua una normalizzazione delle tabelle dividendo per P(e).

3.3.1 Trasmissione del messaggio da VTP a TPF

1) $t_{TP}^* = \sum_V P(V, T, P) = P(T, P)$

2) $t_{TPF}^* = P(T, P, F)$

non cambia nessuna tabella

tabelle dei nodi	tabelle dei separatori
$t_{VTP} = P(V, T, P)$	$t_{TP} = P(T, P)$
$t_{TIF} = P(T, I, F)$	$t_{TF} = P(T, F)$
$t_{TPF} = P(T, P, F)$	$t_{PF} = P(P, F)$
$t_{PFCE} = P(P, F, C, E, e)$	

3.3.2 Trasmissione del messaggio da TIF a TPF

1) $t_{TF}^* = \sum_I P(T, I, F) = P(T, F)$

2) $t_{TPF}^* = P(T, P, F)$

non cambia nessuna tabella

tabelle dei nodi	tabelle dei separatori
$t_{VTP} = P(V, T, P)$	$t_{TP} = P(T, P)$
$t_{TIF} = P(T, I, F)$	$t_{TF} = P(T, F)$
$t_{TPF} = P(T, P, F)$	$t_{PF} = P(P, F)$
$t_{PFCE} = P(P, F, C, E, e)$	

3.3.3 Trasmissione del messaggio da PFCE a TPF

1) $t_{PF}^* = \sum_{EC} P(P, F, C, E, e) = P(P, F, e)$

2) $t_{TPF}^* = P(T, P, F)\, P(P, F, e) / P(P, F) = P(T \mid P, F)\, P(P, F)\, P(P, F, e) / P(P, F)$
$= P(T \mid P, F)\, P(P, F, e) = P(T, P, F, e)$

tabelle dei nodi	tabelle dei separatori
$t_{VTP} = P(V, T, P)$	$t_{TP} = P(T, P)$
$t_{TIF} = P(T, I, F)$	$t_{TF} = P(T, F)$
$t_{TPF} = P(T, P, F, e)$	$t_{PF} = P(P, F, e)$
$t_{PFCE} = P(P, F, C, E, e)$	

	$P(P, F, e)$	
P	f_0	f_1
p_0	0.00927705	0.0182409
p_1	0.0602736	0.1241544
p_2	0.12264875	0.3699168

		$P(T, P, F, e)$		
T	F	p_0	p_1	p_2
t_0	f_0	0.00802125	0.035061	0.02685375
t_0	f_1	0.0098325	0.025194	0.0180576
t_1	f_0	0.0012558	0.0252126	0.095795
t_1	f_1	0.0084084	0.0989604	0.3518592

3.3.4 Trasmissione del messaggio da TPF a VTP

1) $t_{TP}^* = \sum_F P(T, P, F, e) = P(T, P, e)$

2) $t_{VTP}^* = P(V, T, P)\, P(T, P, e) / P(T, P) = P(V \mid T, P)\, P(T, P)\, P(T, P, e) / P(T, P)$
$= P(V \mid T, P)\, P(T, P, e) = P(V, T, P, e)$

tabelle dei nodi		tabelle dei separatori	
t_{VTP}	$= P(V, T, P, e)$	t_{TP}	$= P(T, P, e)$
t_{TIF}	$= P(T, I, F)$	t_{TF}	$= P(T, F)$
t_{TPF}	$= P(T, P, F, e)$	t_{PF}	$= P(P, F, e)$
t_{PFCE}	$= P(P, F, C, E, e)$		

	$P(T, P, e)$		
T	p_0	p_1	p_2
t_0	0.01785375	0.060255	0.04491135
t_1	0.0096642	0.124173	0.4476542

		$P(V, T, P, e)$		
T	V	p_0	p_1	p_2
t_0	v_0	0.00263925	0.0023175	0.00136095
t_0	v_1	0.004347	0.00927	0.0054438
t_0	v_2	0.0108675	0.0486675	0.0381066
t_1	v_0	0.00464625	0.022995	0.00658315
t_1	v_1	0.003717	0.068985	0.0263326
t_1	v_2	0.00130095	0.032193	0.41473845

3.3.5 Trasmissione del messaggio da TPF a TIF

1) $t_{TF}^* = \sum_P P(T, P, F, e) = P(T, F, e)$

2) $t_{TIF}^* = P(T, I, F)\, P(T, F, e) / P(T, F) = P(I \mid T, F)\, P(T, F)\, P(T, F, e) / P(T, F) =$
$= P(I \mid T, F)\, P(T, F, e) = P(T, I, F, e)$

tabelle dei nodi		tabelle dei separatori	
t_{VTP}	$= P(V, T, P, e)$	t_{TP}	$= P(T, P, e)$
t_{TIF}	$= P(T, I, F, e)$	t_{TF}	$= P(T, F, e)$
t_{TPF}	$= P(T, P, F, e)$	t_{PF}	$= P(P, F, e)$
t_{PFCE}	$= P(P, F, C, E, e)$		

	$P(T, F, e)$	
T	f_0	f_1
t_0	0.069936	0.0530841
t_1	0.1222634	0.459228

		$P(T, I, F, e)$		
T	F	i_0	i_1	i_2
t_0	f_0	0.009024	0.023688	0.037224
t_0	f_1	0.0027939	0.01257255	0.03771765
t_1	f_0	0.0345527	0.0558159	0.0318948
t_1	f_1	0.020874	0.116298	0.322056

3.3.6 Trasmissione del messaggio da TPF a PFCE

1) $t_{PF}^* = \sum_T P(T, P, F, e) = P(P, F, e)$

2) $t_{PFCE}^* = P(P, F, C, E, e)$

non cambia nessuna tabella

tabelle dei nodi	tabelle dei separatori
$t_{VTP} = P(V, T, P, e)$	$t_{TP} = P(T, P, e)$
$t_{TIF} = P(T, I, F, e)$	$t_{TF} = P(T, F, e)$
$t_{TPF} = P(T, P, F, e)$	$t_{PF} = P(P, F, e)$
$t_{PFCE} = P(P, F, C, E, e)$	

3.4 Inferenza con evidenza

Si calcolano adesso le probabilità calcolate precedentemente per vedere quali variazioni hanno subito in seguito all'inserimento dell'evidenza.

$P(P, e) = \sum_T P(T, P, e) = (p_0 = 0.02751795\ p_1 = 0.184428\ p_2 = 0.49256555)$

$P(P \mid e) = P(P, e) / P(e) = (p_0 = 0.039059618\ p_1 = 0.26178139\ p_2 = 0.699158992)$

$P(F, e) = \sum_T P(T, F, e) = (f_0 = 0.1921994\ f_1 = 0.5123121)$

$P(F \mid e) = P(F, e) / P(e) = (f_0 = 0.272812296\ f_1 = 0.727187704)$

$P(E, e) = \sum_{PFC} P(P, F, C, E, e) = (e_0 = 0\ e_1 = 0.7045115)$

$P(E \mid e) = P(E, e) / P(e) = (e_0 = 0\ e_1 = 1).$

Come si può vedere, le probabilità di essere preparato e di aver frequentato il corso, supposto di aver superato l'esame, sono aumentate. Infatti le probabilità senza evidenza erano le seguenti:

$P(P) = (p_0 = 0.2453\ p_1 = 0.2292\ p_2 = 0.5255)$

$P(F) = (f_0 = 0.347\ f_1 = 0.653)$

$P(E) = (e_0 = 0.2954885\ e_1 = 0.7045115).$

Vediamo anche come si sono modificate le probabilità a priori delle variabili *V*, *T*, *I* e *C*:

$P(V, e) = \sum_{TP} P(V, T, P, e) = (v_0 = 0.0405421\ v_1 = 0.1180954\ v_2 = 0.545874)$
$P(V \mid e) = P(V, e) / P(e) = (v_0 = 0.0575464\ v_1 = 0.167627356\ v_2 = 0.774826244)$

$P(T, e) = \sum_{F} P(T, F, e) = (t_0 = 0.1230201\ t_1 = 0.5814914)$
$P(T \mid e) = P(T, e) / P(e) = (t_0 = 0.17461759\ t_1 = 0.82538241)$

$P(I, e) = \sum_{TF} P(T, I, F, e) = (i_0 = 0.0672446\ i_1 = 0.20837445\ i_2 = 0.42889245)$
$P(I \mid e) = P(I, e) / P(e) = (i_0 = 0.095448549\ i_1 = 0.295771538\ i_2 = 0.608779913)$

$P(C, e) = \sum_{PFE} P(P, F, C, E, e) = (c_0 = 0.33929985\ c_1 = 0.36521165)$
$P(C \mid e) = P(C, e) / P(e) = (c_0 = 0.481610094\ c_1 = 0.518389906)$

Si nota che i valori si sono modificati nel senso che si poteva facilmente prevedere. Le probabilità a priori erano infatti le seguenti:

$P(V) = (v_0 = 0.1\ v_1 = 0.2\ v_2 = 0.7)$
$P(T) = (t_0 = 0.3\ t_1 = 0.7)$
$P(I) = (i_0 = 0.1\ i_1 = 0.3\ i_2 = 0.6)$
$P(C) = (c_0 = 0.5\ c_1 = 0.5)$.