Pattern recognition

III Parte

QUANTIZZAZIONE VETTORIALE

Si può pensare come un algoritmo derivato da K-MEANS e da ISODATA.

1. All’inizio tutti i vettori sono attribuiti ad un unico cluster. Si calcola l’unico rappresentante come valor medio di tutti i vettori;

2. Per ciascun rappresentante finora trovato, si crea una perturbazione (casuale o meglio ancora lungo la direzione di massima dispersione) in modo da determinare due nuovi rappresentanti (il vecchio viene scaricato, e si genera un numero doppio di nuovi rappresentanti);

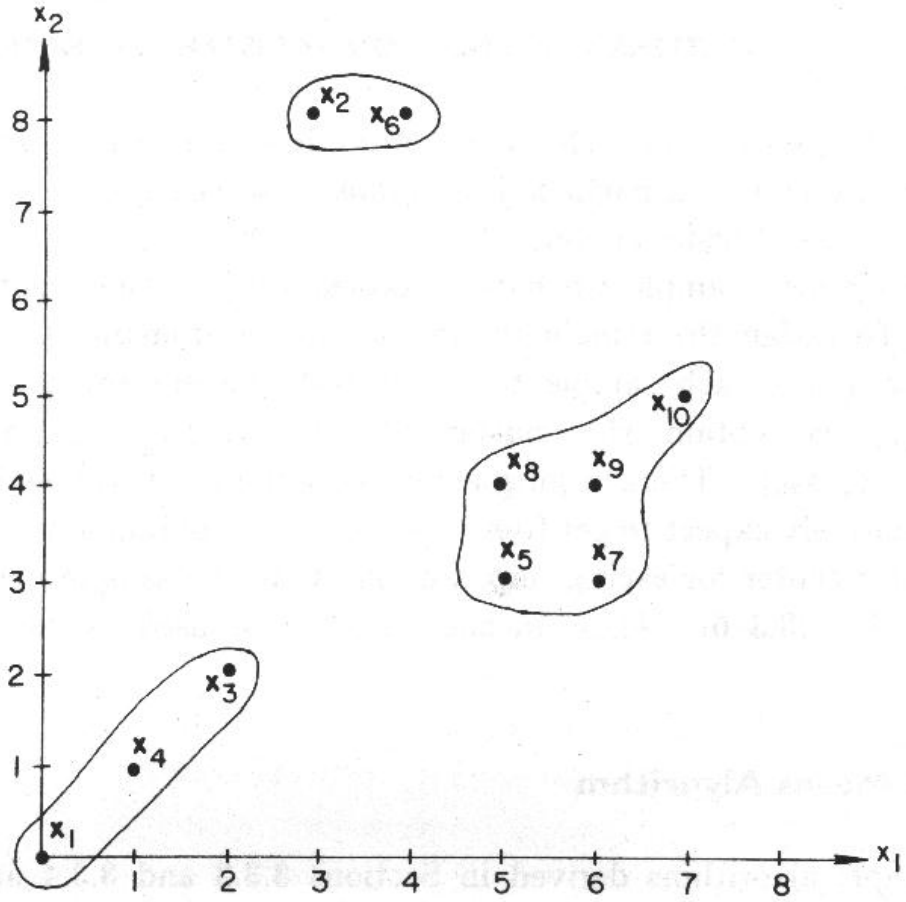
3. si assegna ciascun vettore ad uno dei nuovi rappresentanti, secondo il criterio della minima distanza, formando nuovi cluster;

4. si ricalcolano i nuovi rappresentanti come il vettore medio dei vettori assegnati a ciascun cluster: saranno quelli i nuovi centri di cluster (si abbandonano i vettori "perturbati").

5. Se *non* è soddisfatto il "criterio di fine", si va a 2.

Alla fine i vettori rappresentanti, detti *codeword*, sono etichettati con un intero da 0 a 2^N-1, dove N è il numero di bit utilizzato per codificare i vettori.

Esempio numerico:



$X_1=(0,0)$
$X_2=(3,8)$
$X_3=(2,2)$
$X_4=(1,1)$
$X_5=(5,3)$
$X_6=(4,8)$
$X_7=(6,3)$
$X_8=(5,4)$
$X_9=(6,4)$
$X_{10}=(7,5)$

Al primo passo, c'è un unico cluster.

Il centro di questo cluster è:

$R_1 = (3.9 , 3.8)$

Si crea una perturbazione:

$R_1^- = (3.8 , 3.7)$

$R_1^+ = (4.0 , 3.9)$

Questi sono i nuovi centri di cluster (provvisori): R_1 viene scaricato.

Si rifanno le assegnazioni dei punti al valore perturbato più vicino:

$\{X_1, X_3, X_4\}$

$\{X_2, X_5, X_6, X_7, X_8, X_9, X_{10}\}$

Si calcolano i nuovi centri di cluster:

R_{11}=(1.0, 1.0)

R_{12}=(5.14, 5.0)

Si ripete il procedimento (perturbazione di ciascun centro + riassegnazione + calcolo_nuovo_centro), ottenendo quattro nuovi cluster.

Il “criterio di fine” può essere legato:

1. al raggiungimento del numero di bit N stabilito a priori per la codifica;

2. al superamento di una soglia minima di “distorsione” accettata, se si è definito un opportuno indice di distorsione riferita all’applicazione che si sta trattando.

NB: sostituire i vettori veri con i rispettivi rappresentanti può essere considerato come una *distorsione* dello spazio vero;

ovvero, se si stanno trattando sequenze di vettori, ciascuno dei quali è un punto nello spazio, al *pattern* (traiettoria) vero si sostituisce un *pattern distorto*, quello che è costretto a passare per i punti (vettori) rappresentanti piuttosto che per i punti veri.

In quest'ultimo caso il numero di bit N utilizzato per la codifica non è stabilito a priori, ma è quello trovato.

Vantaggi:

- non è richiesto di assegnare a priori i K rappresentanti iniziali come negli algoritmi primitivi K-MEANS e ISODATA;

- è basato su criteri oggettivi (geometrici) e non su euristiche (come, ad esempio, ISODATA)

Limiti:

- fissare a priori il numero N di bit su cui rappresentare i codici può essere inefficiente o inadeguato

- trattare tutto lo spazio dei vettori alla stessa stregua può essere inefficiente o inadeguato (qui non si effettuano “aggiustamenti” se un cluster è poco rappresentativo o troppo ricco)

Campi di utilizzo:

1. dove serve una *riduzione di dimensionalità* (compressione di banda per qualsiasi applicazione)

2. dove serve passare da variabili continue a variabili simboliche (associando etichette ai codici)

Programmazione dinamica

Si consideri il seguente problema: accettare i comandi MSDOS anche se scorretti.

Ad esempio, invece di COPY, si è battuto CIPY (nella tastiera ‘i’ è accanto ad ‘o’) oppure CVOPY (tasti contigui battuti contemporaneamente).

Il problema si può generalizzare se, invece che a lettere, si pensa a stringhe di simboli.

Nel confronto con modelli, sorge un problema del tutto simile: confrontare sequenze di eventi che non si presentano del tutto identici.

Sia dato il set di simboli {a÷z} (stanno al posto di features: potrebbero essere i caratteri battuti sulla tastiera).

Si debba confrontare la stringa **abcde** (stringa di riferimento) con la stringa **lmnopq** (stringa da confrontare).

	a	**b**	**c**	**d**	**e**
l	• 1				
m		• 2			
n					
o			• 3		
p					
q					• 4

Osservazione:

le stringhe si sa dove iniziano e dove finiscono: i punti estremi (1 e 4) sono fissi.

Si supponga di aver individuato un criterio di confronto (costo) e di aver trovato (per esempio mediante una ricerca esaustiva) l'accoppiamento ottimo nel percorso 1-2-3-4. Quindi il percorso 1÷4 è il percorso ottimo (di costo minimo).

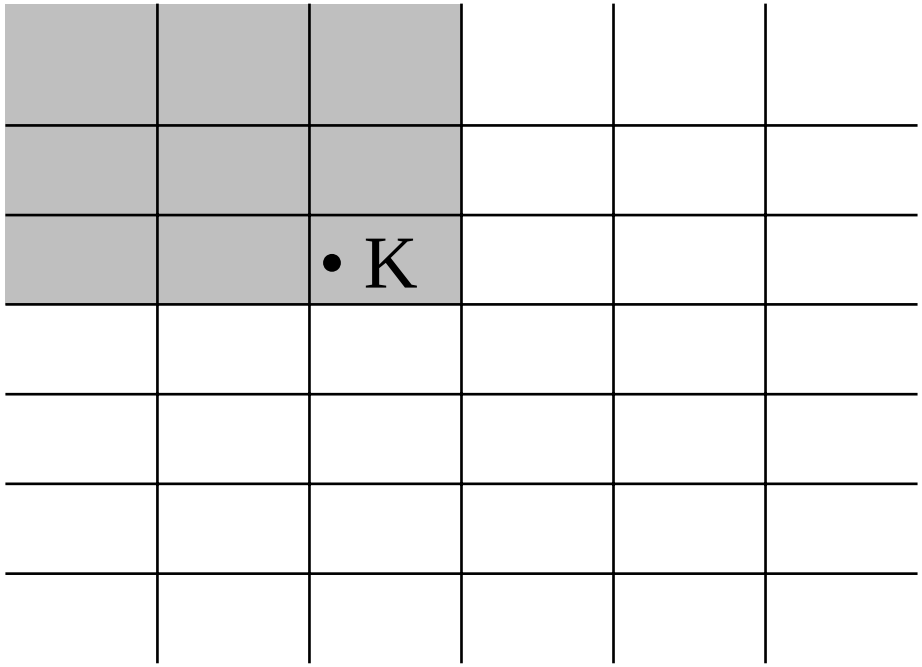
Se si considerano i tratti 1÷3 e 3÷4, ognuno di essi è a costo minimo:

ad esempio, se si fosse arrivati a 4 da un altro punto 3’, sarebbe stato quest’ultimo, e non 3 ad essere inserito nel percorso di costo minimo (vale infatti il principio di ottimalità).

Questo vale per ogni suddivisione.

Da questa considerazione si può concludere che si è arrivati a 4 partendo dal punto 3, che era di costo minimo, avendo “aggiunto” un percorso di costo minimo.

Invece di partire da 1 e guardare in avanti per cercare il percorso di costo minimo (sarebbe richiesta una ricerca (cieca?) con backtracking), si può partire da 1, si va in avanti analizzando tutti (per ora) i punti, ma si guarda all’indietro.



Se si considera il punto K, si può pensare di esserci arrivati da uno dei punti precedenti (rettangolo ombreggiato): in ognuno dei punti precedenti è memorizzato il costo, già calcolato, dall'inizio fino a quel punto, e pertanto il costo per arrivare a K è il costo del punto di partenza più il costo per arrivare a K.

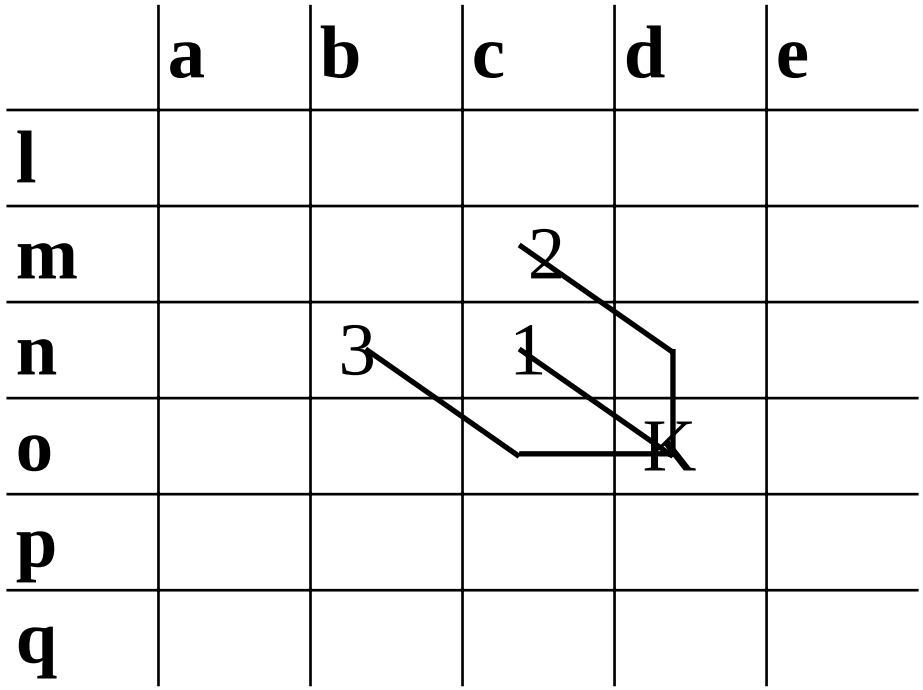
Tra tutte le possibilità, si sceglie quella di costo minimo.

Nel punto K si memorizza il costo così calcolato e il puntatore al punto da cui si è arrivati (quello che ha fornito il minimo).

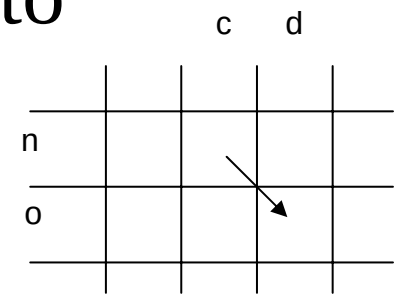
Facendo così per tutti i punti, al punto finale (in basso a destra) si ha il costo del percorso minimo: ripercorrendo all'indietro i puntatori, si può ricostruire il percorso.

Nella pratica, quando si è in K e si guarda all'indietro, non si accetta come punto di provenienza "qualsiasi" punto precedente.

Vediamo di fornire una interpretazione di un percorso parziale.



a) Il percorso 1 dice che si è accettato l'accoppiamento



stringa di riferimento **...cd**

stringa da confrontare **...no**

Quindi si tratta di aver assimilato **d** con **o**: nel caso più generale, il simbolo **o** sostituisce il simbolo **d** (*sostituzione*) e, se coincidono, è il caso di accoppiamento perfetto.

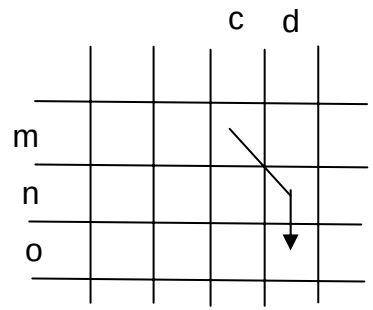
Esempio: ‘l’ ‘o’ ‘**a**’ ‘d’

‘l’ ‘o’ ‘**z**’ ‘d’ (sostituzione)

oppure ‘l’ ‘o’ ‘**a**’ ‘d’

‘l’ ‘o’ ‘**a**’ ‘d’ (uguaglianza)

b) Il percorso 2 dice che si è accettato l'accoppiamento



stringa di riferimento **...cd**

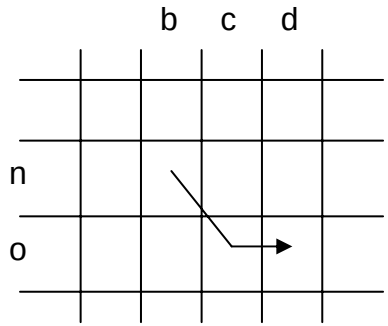
stringa da confrontare **...mo**

con il "salto" del simbolo **n** nella stringa da confrontare (*inserzione*).

Esempio: ‘l’ ‘o’ ‘a’ ‘d’

‘l’ ‘o’ ‘**p**’ ‘a’ ‘d’

(inserzione)



c) Il percorso 3 dice che si è accettato l'accoppiamento

stringa di riferimento **...bd**

stringa da confrontare **...no**

con il "salto" del simbolo **c** nella stringa di riferimento (*cancellazione*).

esempio: 'l' 'o' '**a**' 'd'

'l' 'o' 'd' (omissione o cancellazione)

Si possono ipotizzare altre possibilità, con interpretazioni analoghe (inserzione o cancellazione fino a 2, a 3 o più simboli), comunque in numero e con traiettorie prefissate, in funzione del problema trattato.

NB: a volte vale anche “sostituzione = cancellazione + inserzione”.

Per ogni punto K, si ipotizza di essere arrivati da un numero limitato di punti: il costo del punto K è quindi valutato come il costo del punto di partenza + il costo del salto al punto di arrivo, pesato con un coefficiente che tiene conto del fatto che si effettua un'inserzione, una cancellazione o una sostituzione.

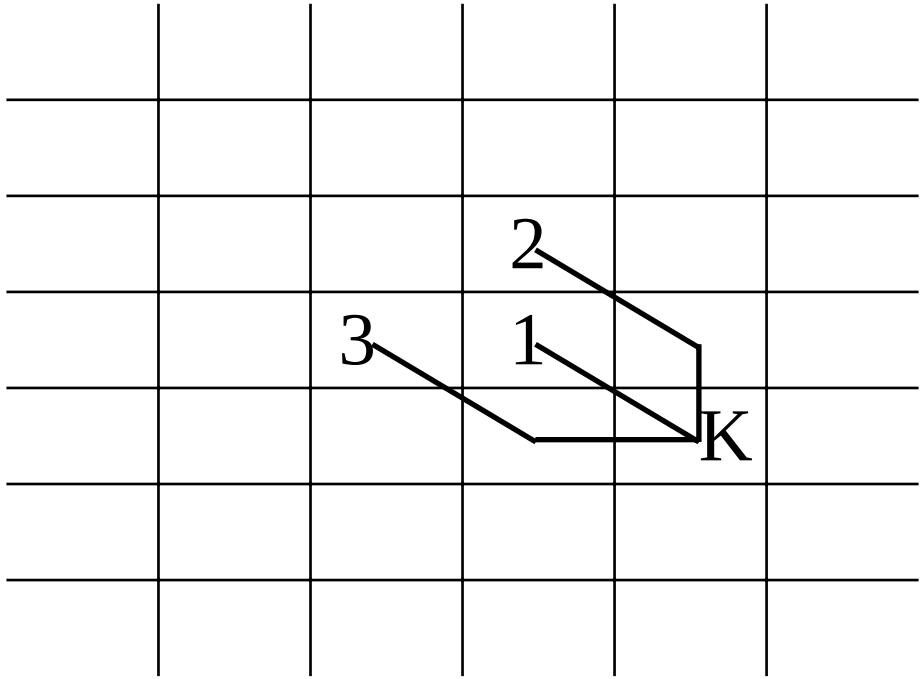
In realtà, si deve tener conto che anche nell'inserzione e nella cancellazione si ha una sostituzione (nel punto K).

coeff $*$ distanza(**d**, **o**)

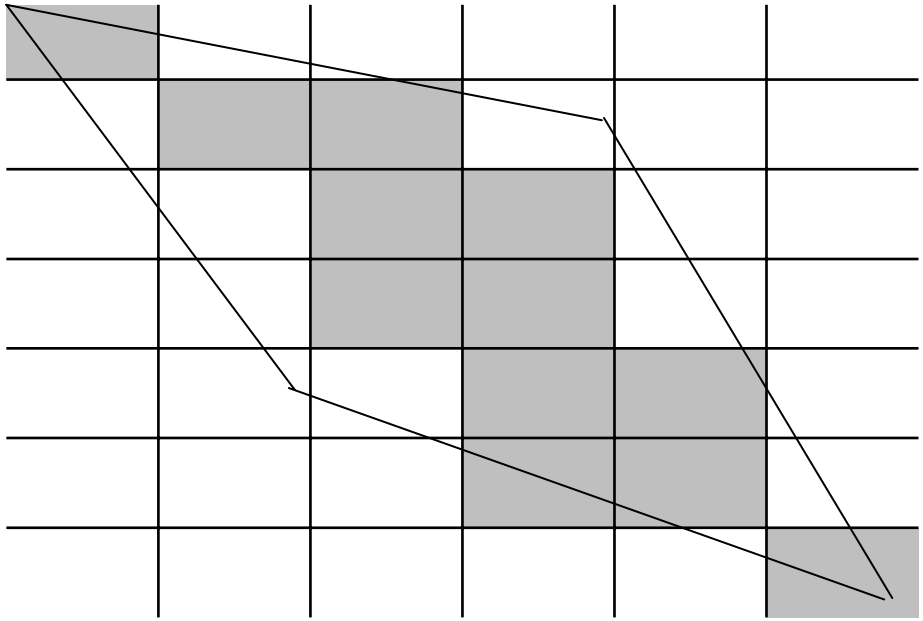
dove *coeff* è 1.0 nel caso a), e un valore maggiore nel caso di cancellazione o inserzione.

Essendo stati posti dei vincoli nei punti di partenza, nei punti di arrivo e nei salti possibili, i punti di valutazione cadono in uno spazio limitato.

Ad esempio, se i “passi” permessi sono i seguenti

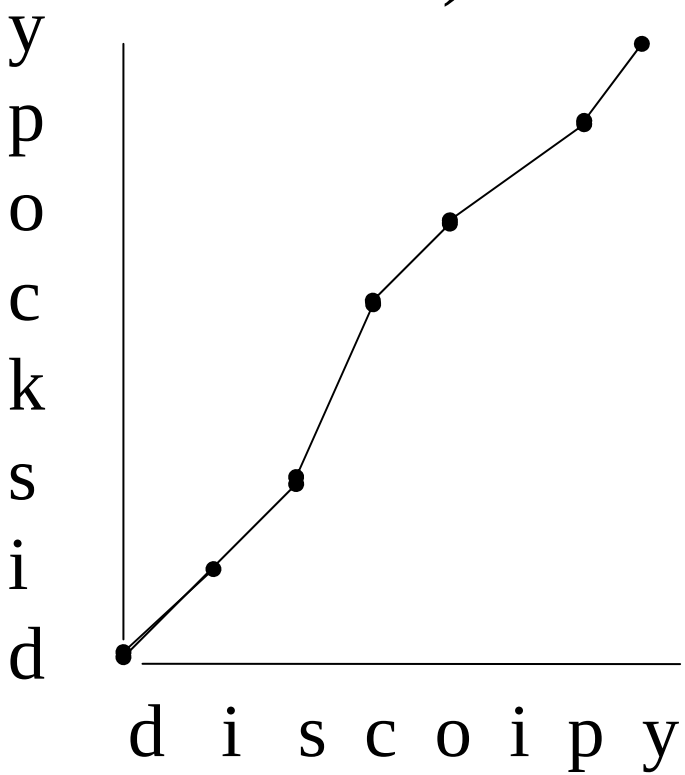


lo spazio che viene esplorato (dati anche i vincoli di inizio e fine) è il seguente



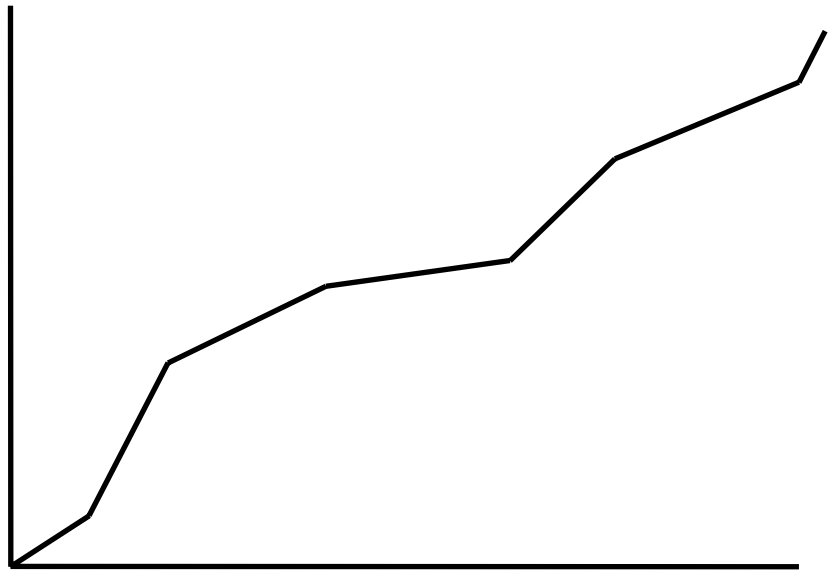
Si pone così una limitazione allo spazio di ricerca, ma anche alle possibili “distorsioni”:

(stringa di riferimento)



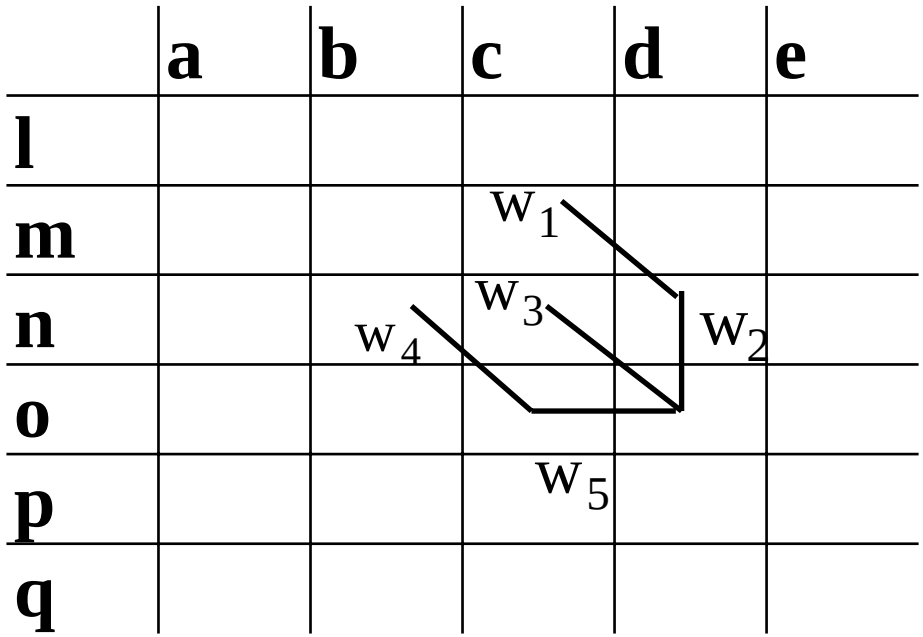
(stringa da confrontare)

In generale:



La determinazione dei costi

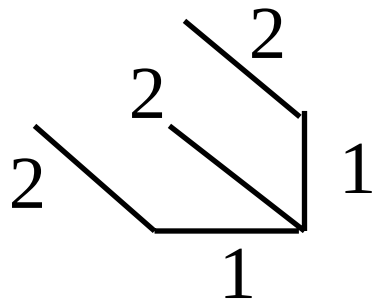
Per i pesi, si può usare uno schema come nella seguente figura



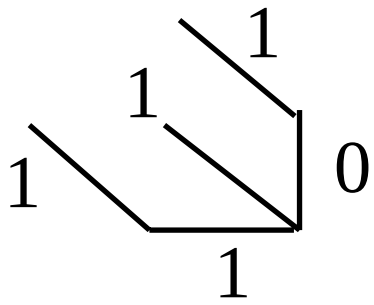
$$C(i,j) = \min \begin{cases} C(i-2, j-1) + w_1 \cdot dist(i-1, j) + \\ w_2 \cdot dist(i, j) \\ C(i-1, j-1) + w_3 \cdot dist(i, j) \\ C(i-1, j-2) + w_4 \cdot dist(i, j-1) + \\ w_5 \cdot dist(i, j) \end{cases}$$

Si possono avere diversi tipi di pesatura. Ecco alcune:

a) pesatura simmetrica:



b) pesatura asimmetrica:



Applicazioni

Solitamente si ha a disposizione una serie di “modelli” (nel nostro esempio, la forma corretta dei comandi) e si utilizza la programmazione dinamica per il confronto.

Si confronta la stringa da analizzare con *tutti* i modelli e si assegna la stringa al modello che ha fornito il miglior indice di accoppiamento.

Usualmente l’assegnamento è condizionato con il superamento di una soglia di accettazione: se la stringa è distante da tutti i modelli, la si rigetta.

(NB: non è detto che questo metodo vada sempre bene: anche qui sarebbe meglio l’ipotesi di “mondo chiuso”!).