Programmazione dinamica e allineamento di sequenze

Laboratorio di Intelligenza Artificiale

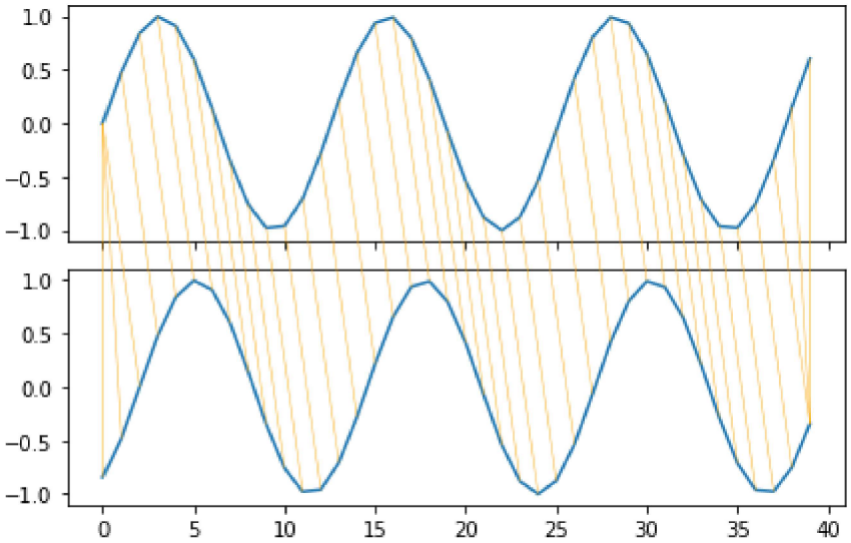
Vincenzo Bonnici
Corso di Laurea Magistrale in Scienze Informatiche
Dipartimento di Scienze Matematiche, Fisiche e Informatiche
Universitá degli Studi di Parma

2025-2026

Dynamic time warping

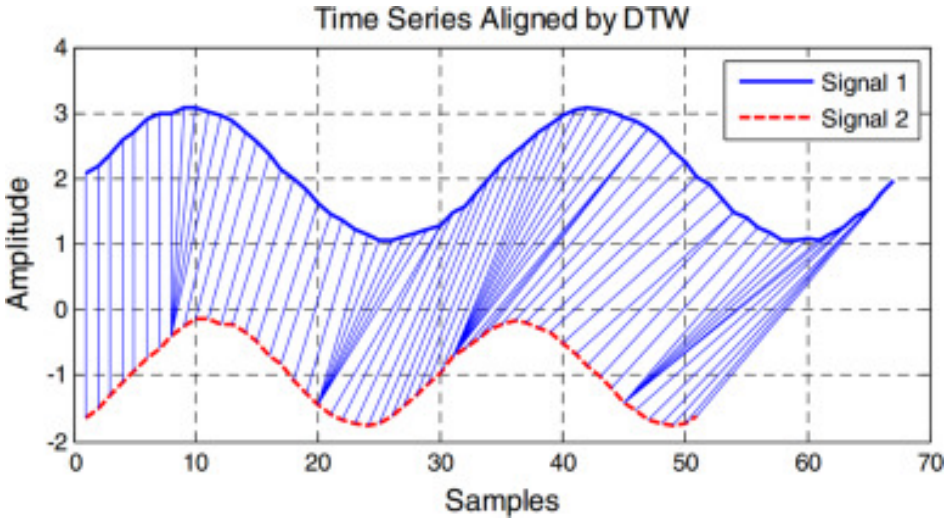
Il **Dynamic Time Warping (DTW)**. è un algoritmo utilizzato per misurare la similarità tra due o più serie temporali.
Permette l'allineamento tra due sequenze, e che può portare ad una misura di distanza tra le due sequenze allineate. Tale algoritmo è particolarmente utile per trattare sequenze in cui singole componenti hanno caratteristiche che variano nel tempo, e per le quali la semplice espansione o compressione lineare delle due sequenze non porta risultati soddisfacenti.

```
x = numpy.arange(0, 20, .5)
s1 = np.sin(x)
s2 = np.sin(x - 1)
```



Dynamic time warping

Prendiamo ad esempio il riconoscimento vocale: una persona potrebbe parlare più velocemente o più lentamente della voce campione preregistrata, mentre il DTW non risentirebbe di questo problema, a differenza di una distanza che venisse applicata "punto per punto".



Il DTW è molto utilizzato in ambito IoT dove gli stream di dati provenienti da video, audio, sensori, possono essere trasformati in una sequenza lineare per essere analizzati con questa tecnica, in particolare è molto efficace con serie ad alta frequenza di campionamento, pensiamo ad esempio al mercato delle utilities o dell'automotive.

Dynamic time warping

L'idea di base è calcolare la distanza tra due vettori/serie storiche indipendentemente dalla loro lunghezza, e per farlo bisogna che i vettori rispettino le seguenti regole:

- ogni indice della prima sequenza deve poter essere confrontato con tutti gli indici delle altre e viceversa;
- il primo indice della prima sequenza deve essere associato al primo indice dell'altra sequenza (ma non deve essere la sua unica corrispondenza);
- l'ultimo indice della prima sequenza deve corrispondere all'ultimo indice dell'altra sequenza (ma non deve essere la sua unica corrispondenza);
- La mappatura degli indici dalla prima sequenza agli indici dell'altra sequenza deve essere monotonicamente crescente. A grandi linee, indipendentemente dalla lunghezza delle serie l'indice iniziale e finale devono coincidere.

Dynamic time warping

Passiamo ora al calcolo della distanza mediante DTW, per farlo ci servirà una matrice di dimensione $n \times m$ dove m ed n sono la lunghezza delle serie confrontate più 1.
Il primo elemento della matrice $DTW[0,0]$ deve essere pari a 0, mentre prima riga e prima colonna pari ad infinito.
Per ogni posizione della matrice verrà calcolata la distanza euclidea di ogni punto di una serie da tutti i punti delle altre partendo dalla posizione $DTW[1,1]$. Ad ogni distanza viene aggiunto il minimo tra:

- il valore precedente sulle righe: **cancellazione**;
- il valore precedente sulle colonne: **inserimento**;
- il valore precedente in diagonale: **corrispondenza**.

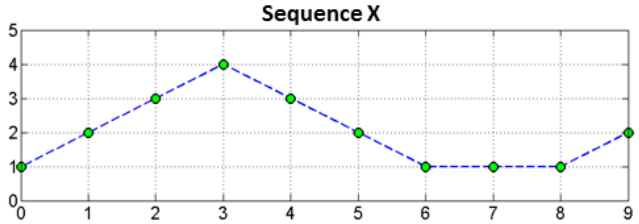
Dynamic time warping

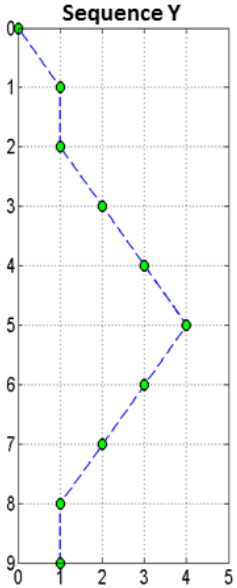
```
int DTWDistance(s: array [1..n], t: array [1..m]) {
    DTW := array [0..n, 0..m]

    for i := 0 to n
        for j := 0 to m
            DTW[i, j] := infinity
    DTW[0, 0] := 0

    for i := 1 to n
        for j := 1 to m
            cost := d(s[i], t[j])
            DTW[i, j] := cost + minimum(DTW[i-1, j  ],    // insertion
                                        DTW[i  , j-1],    // deletion
                                        DTW[i-1, j-1])    // match

    return DTW[n, m]
}
```

Dynamic time warping

L'algoritmo restituisce una matrice con tutte le posizioni valorizzate. Arrivati a questo punto partendo da $DTW[0,0]$ bisogna costruire un percorso che tocchi il valore minimo più vicino a quello nella posizione in cui ci troviamo muovendoci da sinistra verso destra in orizzontale/verticale/obliquo, questo è il **Warping Path**.

- Uno spostamento orizzontale significa che serie_2 è accelerata durante questo intervallo.
- Uno spostamento verticale significa che serie_2 è decelerata durante questo intervallo.
- Il valore finale del path è la distanza tra le serie.



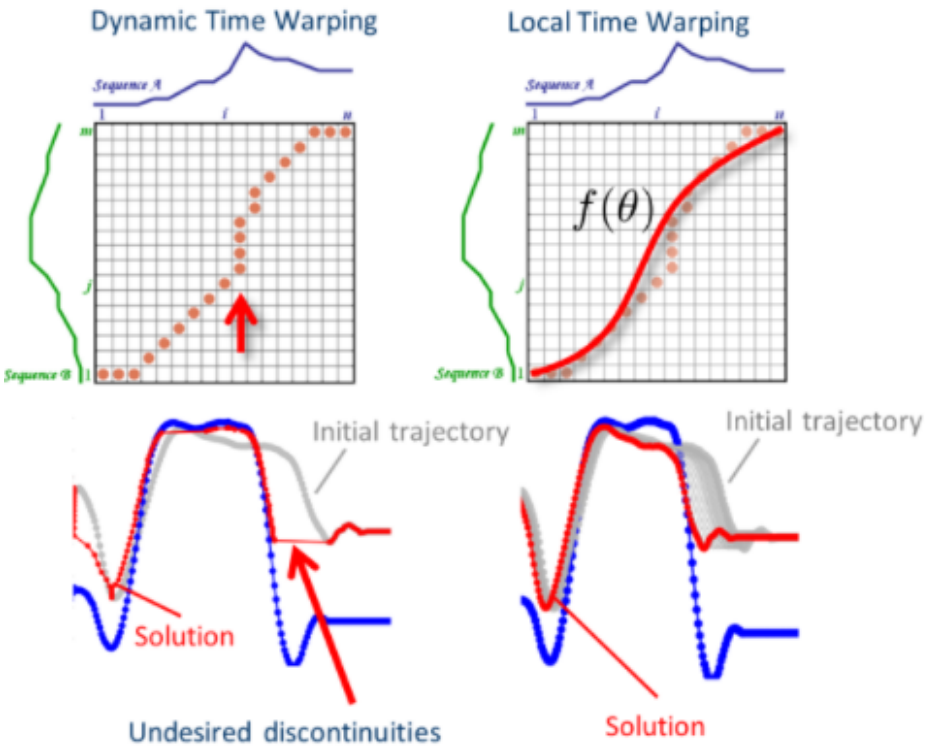


1	5	14	30	39	43	44	45	46	50
1	2	6	15	19	20	20	20	20	21
1	2	6	15	19	20	20	20	20	21
2	1	2	6	7	7	8	9	10	10
6	2	1	2	2	3	7	11	13	11
15	6	2	1	2	6	12	16	20	15
19	7	2	2	1	2	6	10	14	15
20	7	3	6	2	1	2	3	4	4
20	8	7	12	6	2	1	1	1	2
20	9	11	16	10	3	1	1	1	2

Dynamic time warping

La complessità di calcolo del DTW è $O(m \times n)$ dove m ed n rappresentano la lunghezza di ciascuna sequenza.

Se avessimo sequenze molto lunghe, il confronto tra le due risulterebbe complesso, ed anche inutile, perchè alcuni confronti verrebbero effettuati su posizioni lontanissime.
Per questo motivo viene inserita un parametro w, una finestra temporale su cui calcolare le differenze.
La finestra temporale è il valore minimo tra un intero definito dall'utente e la differenza in valore assoluto tra m ed n.



Un'altra alternativa è anche FastWTD.

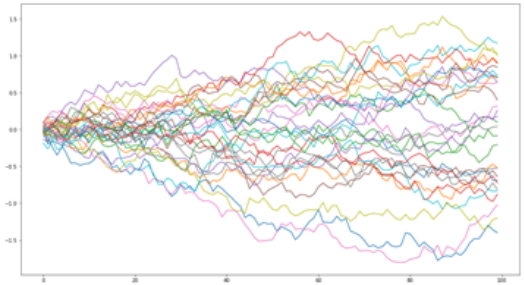
Dynamic time warping e ML

Come può essere/viene usato il DTW nei problemi di Machine Learning?

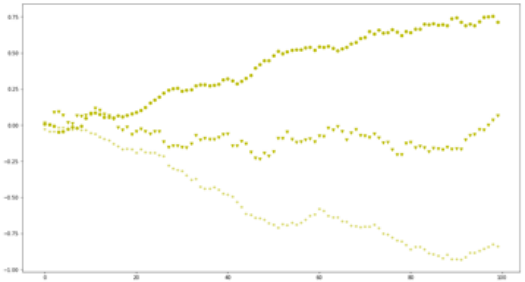
- **Apprendimento supervisionato**, come distanza per algoritmi KNN su Time Series.
- **Apprendimento non supervisionato**, come misura di similarità nel clustering di Time Series.
- **Apprendimento semi-supervisionato**, etichettare le Time Series utilizzando un algoritmo di clustering basato su DTW e successivamente classificare secondo queste assegnazioni.

Dynamic time warping e ML

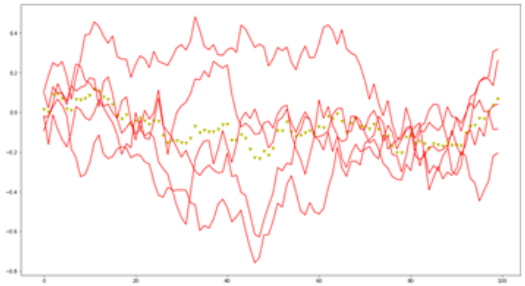
Sequenze originali



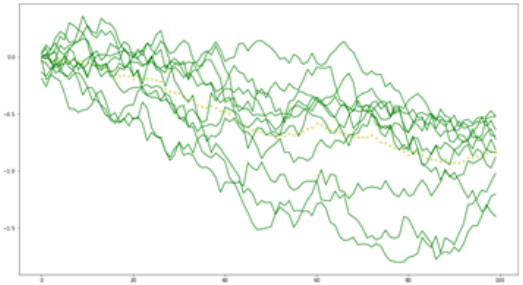
Centroidi



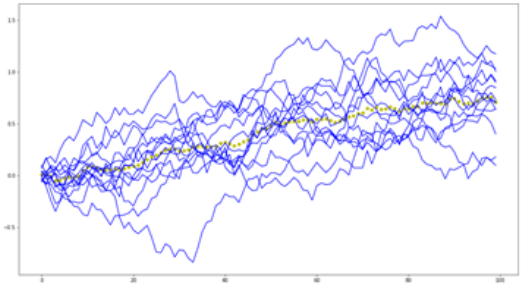
Cluster A: Trend costante



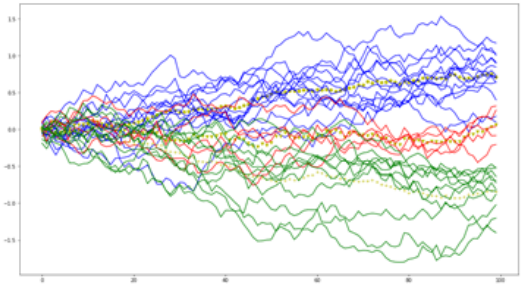
Cluster B: Trend decrescente



Cluster C: Trend crescente



Sequenze etichettate



Altre misure basate sull'allineamento

Sebbene DTW sia probabilmente la misura di somiglianza basata sull'allineamento più nota per le serie temporali, non è l'unica. Alcune altre misure di somiglianza si basano su match locali piuttosto che sull'intera serie. Questo è il caso degli algoritmi **Longest Common Sub-Sequence (LCSS)** e **Longest Common Substring (LCS)**.

Inoltre, il problema dell'allineamento di serie temporali eterogenee (cioè serie temporali i cui elementi non si trovano nello stesso spazio ambientale o non possono essere confrontati direttamente) ha ricevuto una certa attenzione. Una prima misura di somiglianza in questo contesto è **Canonical Time Warping** (Zhou&Torre 2009) che trova sia uno spazio di incorporamento comune per le caratteristiche di entrambe le serie temporali (usando l'analisi di correlazione canonica) sia un percorso di allineamento ottimale à la DTW. Lavori più recenti su questo problema mirano a mappare le caratteristiche di una serie allo **spazio delle caratteristiche** dell'altra serie (Vayer et al. 2020) o ad abbinare matrici di auto-somiglianza (Cohen et al. 2020).

Programmazione dinamica

- La **programmazione dinamica** è un approccio che si può utilizzare in diversi casi per risolvere efficientemente un **problema di ottimizzazione**
  - **Diverse soluzioni** possibili per un problema
  - Ogni soluzione ha un **costo**, il problema è trovare una soluzione con il **costo minimo** o massimo
  - Chiamiamo tale soluzione una **soluzione ottimale**
- Come divide-et-impera, la programmazione dinamica risolve un problema **combinando** le soluzioni di **sottoproblemi**
- Diversamente da divide-et-impera, la programmazione dinamica si applica anche quando **i sottoproblemi non sono indipendenti**
- La programmazione dinamica risolve i problemi in comune **una sola volta**, divide-et-impera più volte

Programmazione dinamica: numeri di Fibonacci

L'n-esimo numero di Fibonacci è definito come:

$$F(n) = \begin{cases} 1 & \text{se } n \leq 1 \\ F(n-1) + F(n-2) & \text{altrimenti} \end{cases}$$

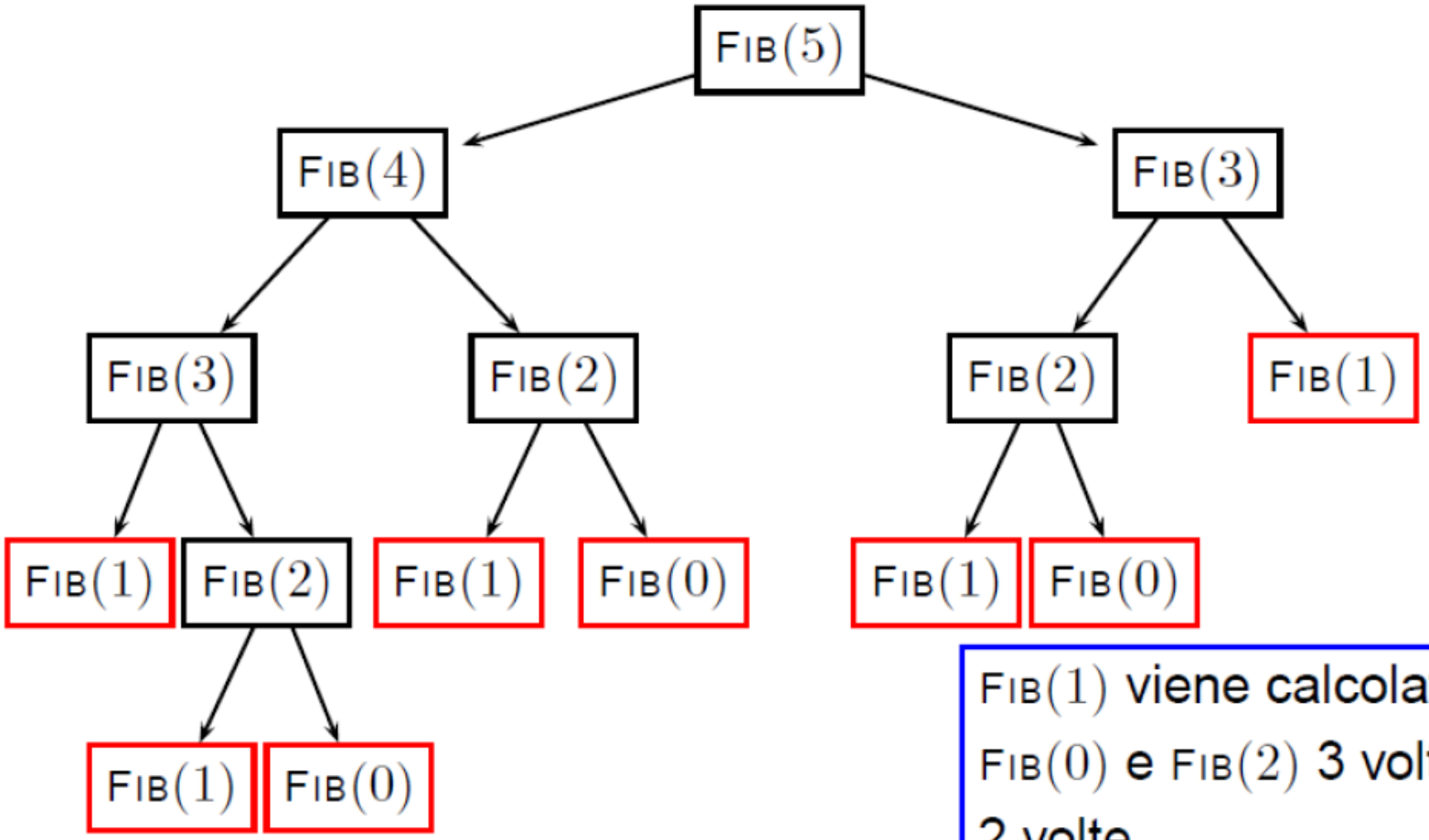
```
def Fib(n):
    if n <= 1:
        return 1
    else:
        return Fib(n-1) + Fib(n-2)
```

Essendo $T(n-1) > T(n-2)$, la complessità dell'algoritmo è

$$T(n) = \begin{cases} \theta(1) & \text{se } n \leq 1 \\ 2 \cdot T(n-2) + \theta(1) & \text{altrimenti} \end{cases}$$

il cui albero di ricorsione è un albero binario completo di altezza $\lfloor \frac{n}{2} \rfloor$ che ha quindi $2^{\lfloor \frac{n}{2} \rfloor + 1} - 1$ nodi. La complessità totale è 2^n.

Programmazione dinamica: numeri di Fibonacci



FIB(1) viene calcolato 5 volte, FIB(0) e FIB(2) 3 volte, FIB(3) 2 volte

Programmazione dinamica: numeri di Fibonacci

Una soluzione più efficiente sarebbe memorizzare i valori di $F(n)$ per n già risolti precedentemente durante il calcolo.
Quindi, se lo specifico valore di n non è stato ancora visto, allora lo si calcola e lo si memorizza in una tabella.
Se invece lo specifico valore n è già stato visto, allora si carica dalla tabella il valore di $F(n)$.

```
table = dict() # chiave n -> valore F(n)

def MemoFib(n):
    if n <= 1:
        return 1
    else:
        if n in table:
            return table[n]
        else:
            return MemoFib(n-1) + MemoFib(n-2)
```

Naturalmente, tale soluzione è efficiente se il salvataggio e il caricamento dei dati dalla tabella è fatto in tempo costante.

Programmazione dinamica: numeri di Fibonacci

Ci accorgiamo però che per calcolare $F(n)$ abbiamo sempre bisogno di calcolare $F(n-1)$ ed $F(n-2)$ e ricorsivamente fino a 0 o 1.
Ovvero, per calcolare $F(n)$ abbiamo bisogno sempre di tutti gli $F(i)$ per $0 \leq i < n$.

Inoltre, per calcolare $F(n)$ dobbiamo sempre aver già risolto tutti gli $F(i)$ per $0 \leq i < n$.
Quindi, perché non applicare un approccio di tipo bottom-up partendo dal calcolo di $F(0)$ e risalendo via via a $F(n)$?

```
def IterFib(n):
    # un semplice array di dimensione n
    F = numpy.empty([10,1], dtype=int)
    F[0] = F[1] = 1
    for i in range(2,n+1):
        F[i] = F[i-1] = F[i-2]
    return F[n]
```

Complessità: $\theta(n)$!!!

Programmazione dinamica

- Un algoritmo di programmazione dinamica può essere sviluppato seguendo quattro **passi**
  - **Caratterizzazione** della struttura di una **soluzione ottima**
  - **Definizione ricorsiva** del **valore** di una soluzione ottima
  - **Calcolo** del **valore** di una soluzione ottima in modo bottom-up, **memorizzando** in tabelle i valori delle sottosoluzioni ottimali
  - **Costruzione** di una **soluzione ottima** dalle informazioni calcolate

- L'ultimo passo non è necessario se è richiesto solo il valore ottimo e non una soluzione ottima

Nota

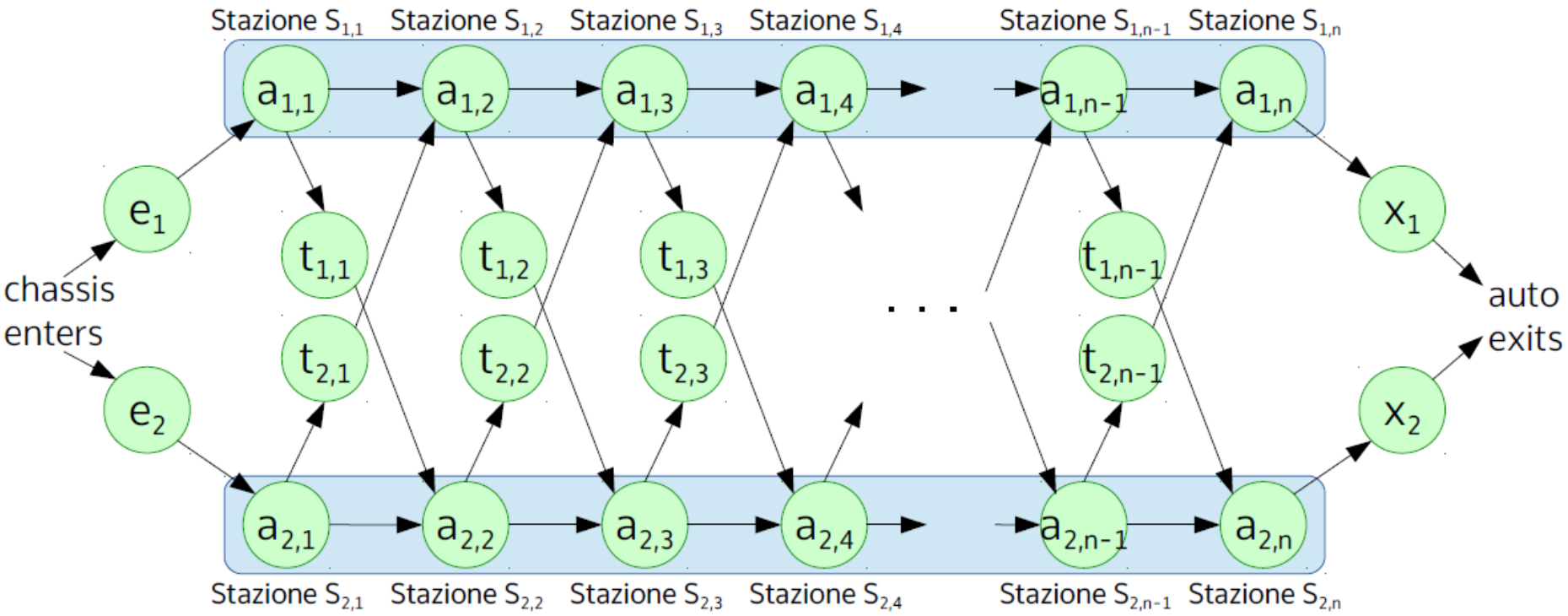
Inventata da Bellmann negli anni '50 per risolvere problemi di schedulazione. La parola dinamica è stata scelta da Bellman per catturare l'aspetto "che varia nel tempo" dei problemi e perché suonava impressionante. La parola programmazione si riferiva all'uso del metodo per trovare un programma ottimale.

Programmazione dinamica

- È conveniente utilizzare la programmazione dinamica quando un algoritmo ricorsivo invocherebbe più volte uno stesso sottoproblema
  - Se ciò non accade, l'approccio divide-et-impera è adeguato

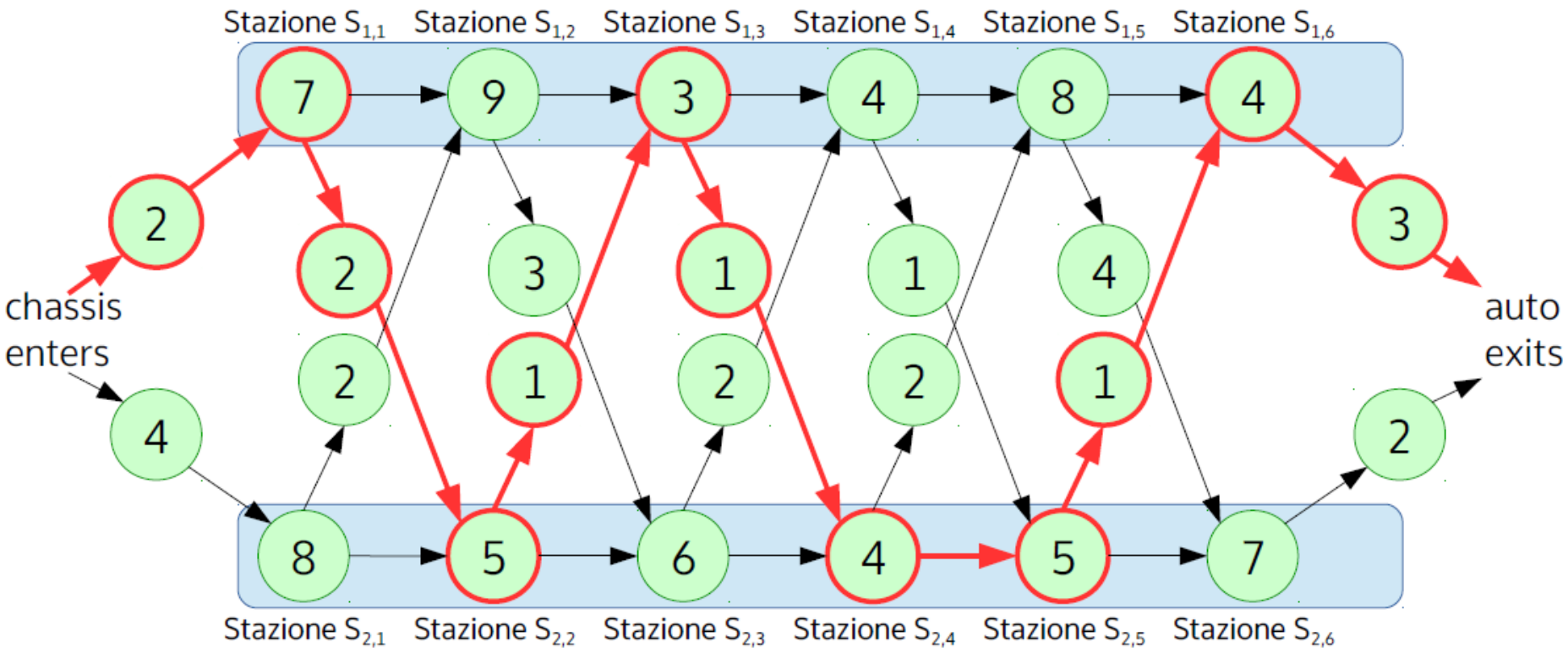
- La programmazione dinamica risolve **ogni sottoproblema una sola volta** e utilizza il valore calcolato quando si ripresenta lo stesso sottoproblema nell'approccio bottom-up
- i sottoproblemi coinvolti devono essere sempre quelli, cioè lo spazio dei sottoproblemi deve essere piccolo
  - il numero dei sottoproblemi distinti deve essere **polinomiale** nella dimensione dell'input
- il metodo divide-et-impera risolverebbe i sottoproblemi inutilmente ogni volta
  - Un algoritmo di programmazione dinamica risolve ogni sottoproblema una sola volta e ne memorizza la soluzione in una tabella (**memoization**), evitando di dover ripetere ogni volta il calcolo della soluzione di un sottoproblema già risolto.

Programmazione dinamica: catene di montaggio



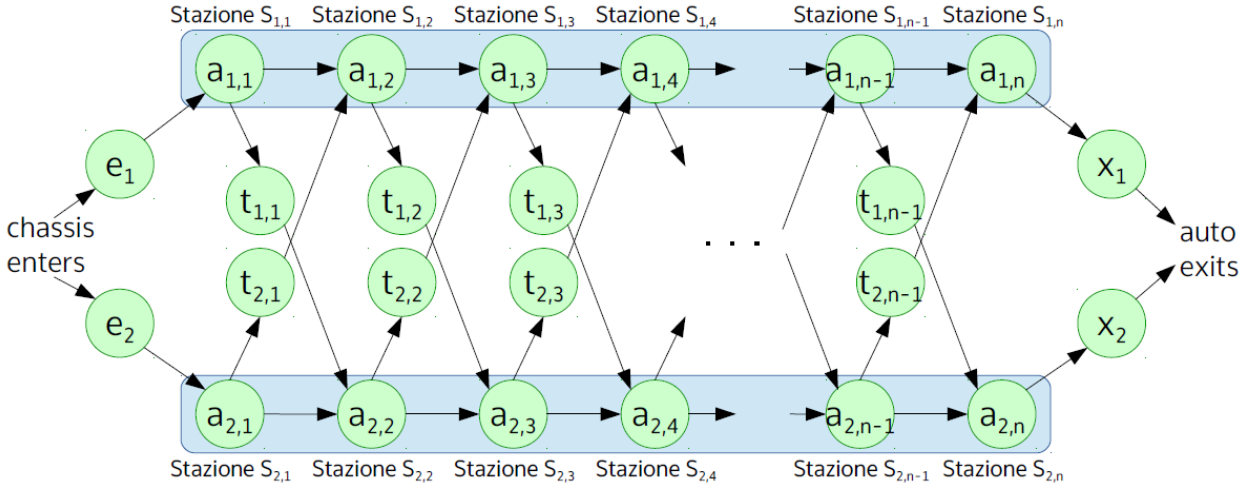
- Uno chassis deve attraversare n **stazioni** su una qualunque delle due **linee**
- I **tempi** di **stazionamento** sono $a_{i,j}$, quelli di **trasferimento** $t_{i,j}$
- Determinare un **percorso** che richiede il **tempo minimo**

Programmazione dinamica: catene di montaggio



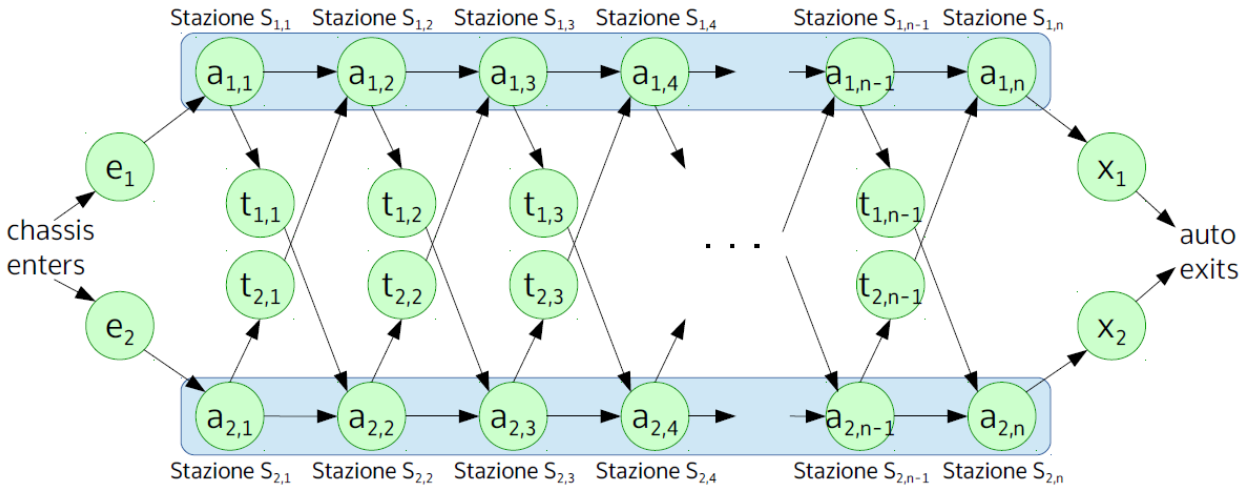
- Il percorso più veloce è evidenziato
- Dato il percorso, calcolarne il tempo richiede $\theta(n)$
- Ma ci sono 2^n possibili percorsi: l'approccio forza bruta non è fattibile!

Programmazione dinamica: catene di montaggio



- Consideriamo il percorso più veloce fino alla stazione $S_{1,j}$
  - $j > 1$, altrimenti è banale (si entra nella linea 1)
  - Lo chassis potrebbe provenire:
    - $S_{1,j-1}$: lo chassis deve aver preso il percorso più veloce fino a $S_{1,j-1}$
    - $S_{2,j-1}$ e poi trasferito su linea 1: lo chassis deve aver preso il percorso più veloce fino a $S_{2,j-1}$
- In ogni caso, la soluzione ottima di un problema contiene al suo interno la soluzione ottima di un sottoproblema
- Il percorso più veloce fino a $S_{1,j}$ è dato da
  - Percorso più veloce fino a $S_{1,j-1}$ e poi direttamente a $S_{1,j}$, oppure
  - Percorso più veloce fino a $S_{2,j-1}$ e poi trasferimento a $S_{1,j}$

Programmazione dinamica: catene di montaggio

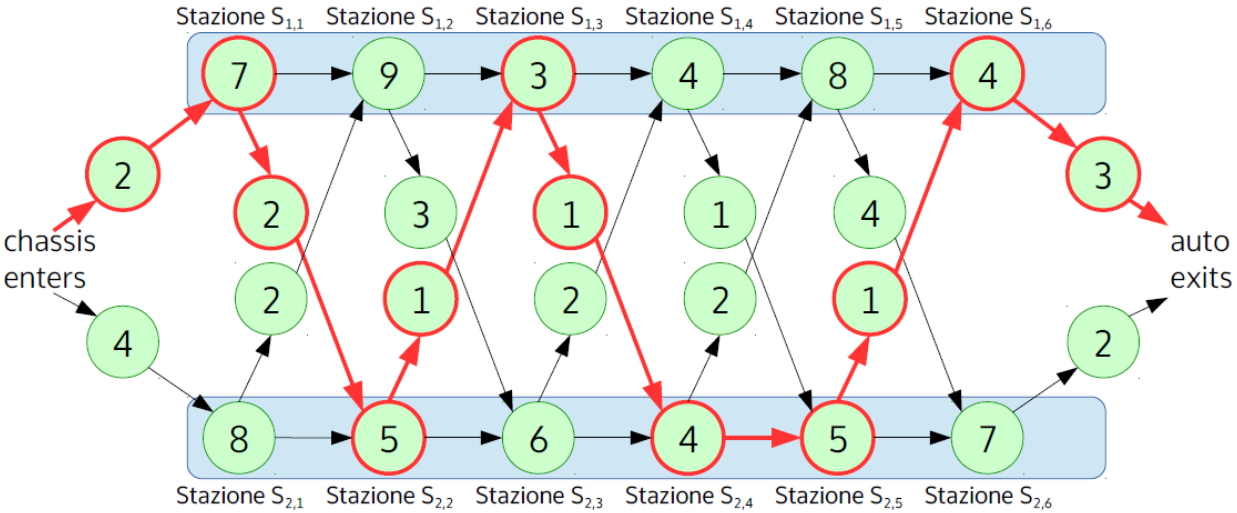


- Si definisce il valore di una soluzione ottima in maniera ricorsiva in funzione delle soluzioni ottime dei sottoproblemi
- Sia $f*$ il tempo minimo di completamento e $f_i[j]$ il tempo minimo per arrivare all'uscita della stazione $S_{i,j}$

$$f* = min(f_1[n] + x_1, f_2[n] + x_2)$$

$f_1[1] = e_1 + a_{1,1}; f_2[1] = e_2 + a_{2,1}$
$f_1[j] = min(f_1[j-1] + a_{1,j}, f_2[j-1] + t_{2,j-1} + a_{1,j})$ per $j > 1$
$f_2[j] = min(f_2[j-1] + a_{2,j}, f_1[j-1] + t_{1,j-1} + a_{2,j})$ per $j > 1$

Programmazione dinamica: catene di montaggio



Per costruire il percorso più veloce, teniamo traccia di:

- $l_i[j]$ la linea (1 o 2) su cui si trova la stazione $(j-1)$ che precede $S_{i,j}$ lungo il percorso più veloce verso $S_{i,j}$
- l^* la linea su cui si trova la stazione n nel percorso più veloce

j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

f*=38

j	2	3	4	5	6
$l_1[j]$	1	2	1	1	2
$l_2[j]$	1	2	1	2	2

l*=1

Usando l^* e $l_i[j]$ si riesce a tracciare il percorso più veloce

Programmazione dinamica: catene di montaggio

```
Fastest-Way (a,t,e,x,n)
f1[1] ← e1+a1,1
f2[1] ← e2+a2,1
for j ← 2 to n
    do if f1[j-1] <= f2[j-1]+t2,j-1
        then f1[j] ← f1[j-1]+a1,j
            l1[j] ← 1
        else f1[j] ← f2[j-1]+t2,j-1+a1,j
            l1[j] ← 2
    if f2[j-1] <= f1[j-1]+t1,j-1
        then f2[j] ← f2[j-1]+a2,j
            l2[j] ← 2
        else f2[j] ← f1[j-1]+t1,j-1+a2,j
            l2[j] ← 1
if f1[n]+x1 <= f2[n]+x2
    then f* = f1[n]+x1
        l* ← 1
    else f* = f2[n]+x2
        l* ← 2
```

```
Print-Stations (l,n)
i ← l*
print "line " i ", station " n
for j ← n downto 2
    do i ← li[j]
    print "line " i ", station " j-1
```

Programmazione dinamica: catene di montaggio

- Sfruttando le equazioni ricorsive ricavate, potremmo scrivere un algoritmo ricorsivo
- Avrebbe un tempo di esecuzione esponenziale!
  - Sia $r_i(j)$ il numero di chiamate ricorsive per calcolare $f_i[j]$
    $r_1(n) = r_2(n) = 1$
    $r_1(j) = r_2(j) = r_1(j+1) + r_2(j+1)$ per $j = 1, 2, \ldots, n-1$
  - Con il metodo di sostituzione si verifica che $r_i(j) = 2^{n-j}$
    $f_1[1]$ viene chiamata 2^{n-1} volte!
  - Si vede che il numero totale di chiamate ricorsive è $\theta(2^n)$
- Calcolando invece $f_1[j]$ e $f_2[j]$ a partire da $j = 1$ riusciamo ad ottenere un tempo di esecuzione $\theta(n)$

Programmazione dinamica

- È conveniente utilizzare la programmazione dinamica quando
  - Il problema esibisce una "sottostruttura" ottima
  - Si presentano sottoproblemi "sovrapposti"

- Il problema esibisce una **sottostruttura ottima** quando **una soluzione ottima del problema contiene al suo interno soluzioni ottime di sottoproblemi**
- Questo è un segno che l'approccio della programmazione dinamica può essere impiegato
  - Ma si potrebbero impiegare anche divide-et-impera e approccio greedy
- La sottostruttura ottima varia da problema a problema in termini
  - del **numero di sottoproblemi** presenti nella soluzione ottima del problema originario
  - del **numero di scelte** che si hanno nel determinare quale/i sottoproblema/i è presente nella soluzione ottima

SAX

Il **Symbolic Aggregate approXimation (SAX)** encoding è un metodo per semplificare le serie storiche.
SAX è un modo di trasformare una serie temporale in una sequenza di simboli.
L'idea di base è che ogni simbolo rappresenta un intervallo.

Questa tecnica permette di condurre una **dimensionality reduction** sulle serie storiche, quindi possiamo considerarlo un metodo non supervisionato.

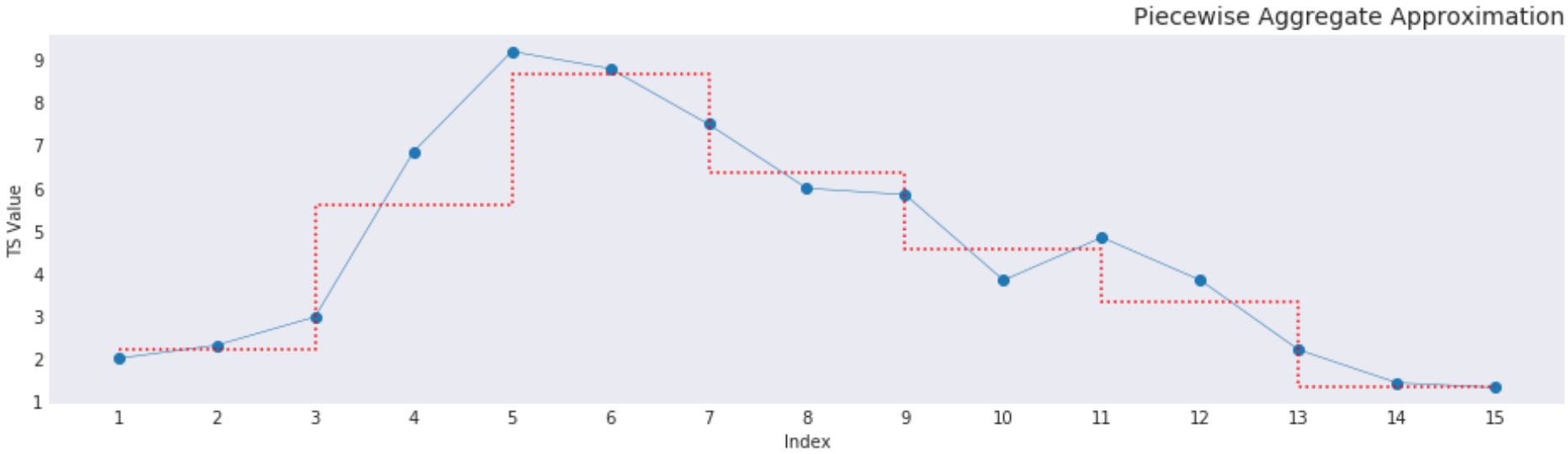
Bisogna evidenziare che il SAX è una tecnica molto robusta ai valori mancanti, e quindi utilizzabile quando si è in presenza di serie di lunghezza diversa.

Le serie devono essere standardizzate così da avere la stessa scala.

$$Z = \frac{x - \mu}{\sigma}$$

SAX - PAA

SAX riduce la dimensionalità della serie e lo fa proprio con la øPiecewise Aggregate Approximation (PAA), l'idea alla base è "riassumere" la sequenza in una serie di segmenti che ne riducano la lunghezza ma ne rappresentino ugualmente l'informazione.



SAX - PAA

Data una time series $Y = [Y_1, Y_2, \ldots, Y_n]$ che può essere ridotta in una sequenza $X = [X_1, X_2, \ldots, X_m]$ con $m \leq n$ l'equazione che sintetizza gli elementi della time series originale per adattarli alla nuova dimensione è la seguente:

$$\bar{X}_i = \frac{m}{n} \cdot \sum_{j=n/N(i-1)+1}^{(n/M)i} x_j$$

Quindi la media per ogni finestra temporale che costituirà la nuova sequenza. Si possono notare subito due casi particolari:

- $m = n$, la nuova sequenza è identica alla serie storica originale
- $m = 1$, la nuova sequenza ha un solo valore equivalente alla media dei valori dell'intera time series

SAX - PAA

Per applicare la P.A.A. avremo bisogno di scegliere la dimensione della finestra temporale su cui calcolare l'informazione, questo influirà anche su quanti segmenti verranno utilizzati per ricostruire la serie storica.

Ridimensionata, ogni serie sarà una sequenza di $lunghezza_serie/w$ periodi, nella quale si prende la media del periodo come nuovo valore.

Attenzione: Bisogna impostare un controllo per non perdere informazioni, quindi se il nuovo numero di periodi non è un intero, esso va sempre arrotondato per eccesso.

SAX

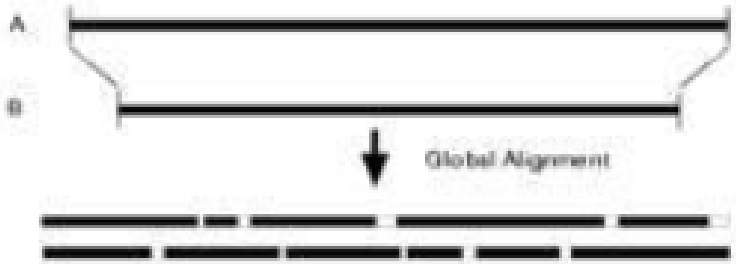
Preparato il dato si può lavorare a ciò che contraddistingue questa tecnica. Adesso il dato numerico verrà convertito in stringa.

La **SAX String** è derivazione del numero di livelli in cui si vuole suddividere la serie, ogni livello avrà un'etichetta, queste etichette verranno successivamente concatenate creando la SAX String.

La scelta dei livelli influisce sul risultato, quindi bisognerebbe sempre farsi affiancare da un esperto di dominio, il consiglio è quello di non creare troppi livelli altrimenti si ha un'eterogeneità troppo alta e non si troverebbero pattern.
I livelli possono essere fissati oppure calcolati ad ogni periodo.

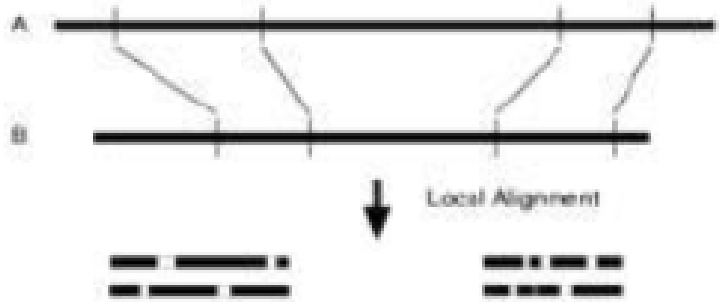
Allineamento di sequenze

L'allineamento di sequenze è il processo di appaiamento delle regioni simili condivise da due sequenze.



Allineamento globale

Le sequenze sono allineate secondo la loro intera lunghezza (inserimento di gap se hanno lunghezze diverse)



Allineamento locale

Non richiede che il confronto di due sequenze sia svolto per la loro intera lunghezza

Allineamento di sequenze

L'allineamento di sequenze nasce per risolvere dei problemi di carattere **bioinformatico**.
In particolare, si voglio confrontare le sequenze di **due proteine**, ognuna appartenete ad una specie diversa ma che si suppone avere un progenitore comune. Con l'allineamento di sequenza riusciamo a farci una idea di come una proteina si stata **modificata** al fine di ottenere l'altra.

Una proteina presente in un organismo può cambiare in seguito a **mutazioni** nella sua sequenza codificante. Le mutazioni possono essere puntiformi (singola base) o più estese.
Nel caso di **mutazioni puntiformi**, distinguiamo:

- **Sostituzione**: una base (carattere ad una determinata posizione della stringa) viene sostituita con un'altra
- **Inserzione**: una o più basi vengono inserite
- **Delezione**: una o più basi vengono tolte
- **Inversione**: un tratto di DNA (stringa della proteina) si inverte

Allineamento di sequenze

Trovare il modo di convertire le sequenze inserendo il minor numero possibile di cambiamenti.

Si assegnano dei criteri per valutare l'importanza dei:

- **MATCH**: similarità positiva
- **MISMATCH**: similarità negativa
- **GAP**: similarità negativa, ma che permette di migliorare l'allineamento globale.

Con soli **mismatch**:

```
ATTCGTAAGCGTCCTCATAGCTGCAGT
|||| |||| ||| |||  |||||||||
ATTCATAAGGGTCGTCAGTGCTGCAGT
```

Con **gap** da **inserimenti** e **cancellazioni**:

```
   IPLMTRWDQEQESDFGHKLP-IYTREWCTRG
   ||||||||| |||||||||| ||||||
CHKIPLMTRWDQ-QESDFGHKLPVIYTREW
```

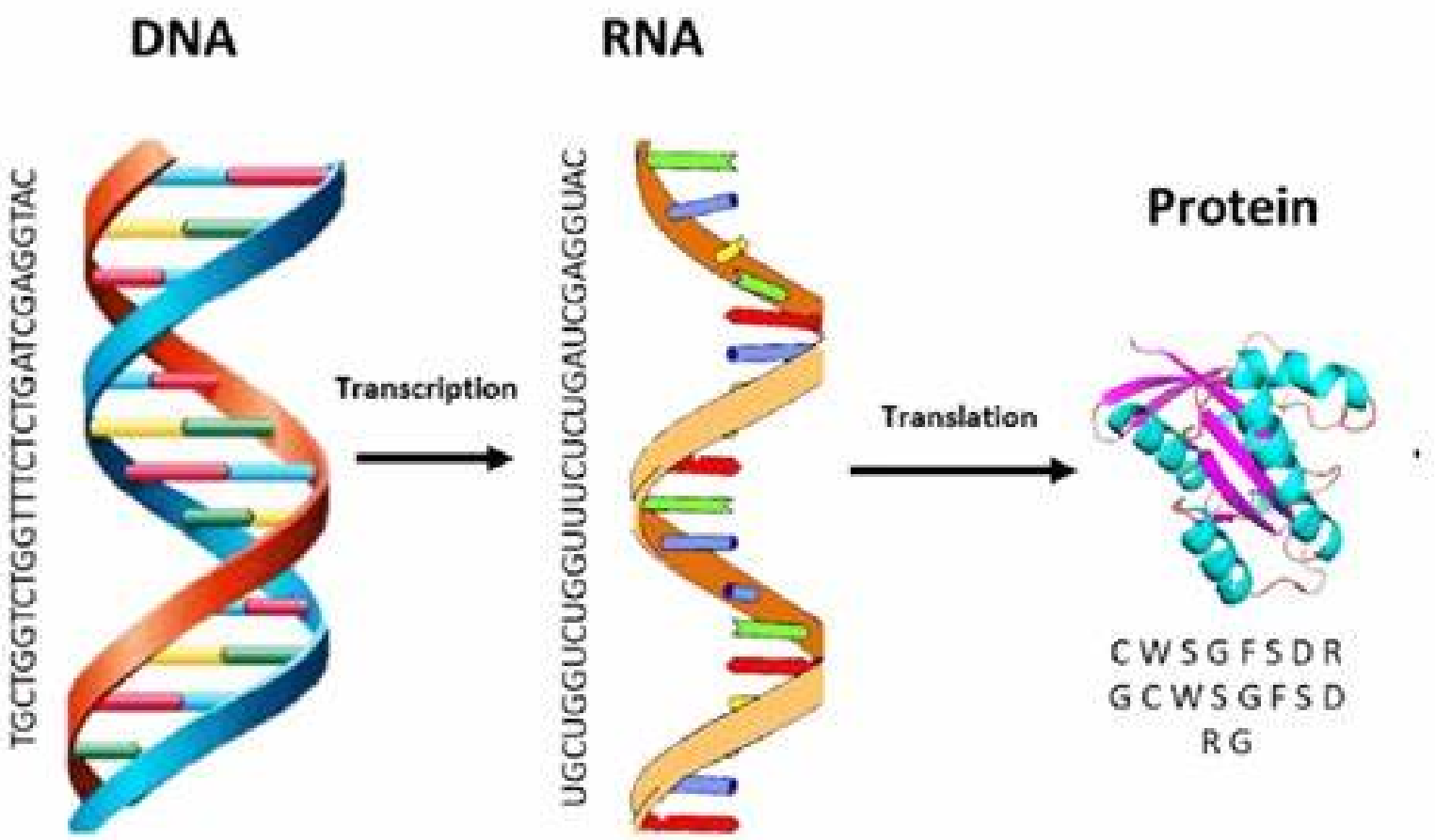
Allineamento di sequenze

Nel caso di allineamento di due sequenze di dati biologici l'obiettivo è:

- Massimizzare il numero di coppie di elementi allineati con un punteggio positivo
- Minimizzare il numero di gap e allineamenti con punteggio negativo

Il problema è decomposto nell'allineare in modo ottimale prima coppie di singoli elementi (sottoproblema di dimensione minore) e poi "sottosequenze" via via più grandi (sottoproblemi di dimensioni maggiori)

Dogma centrale della biologia



Allineamento di sequenze: matrici di sostituzione

Una proteina può essere identificata tramite la sua sequenza nucleotidica (DNA) o aminoacidica. Nel caso degli **aminoacidi** abbiamo una alfabeto di 20 (a volte 21) caratteri.

Confrontando gli amino acidi, è opportuno creare dei criteri di **similarità** che non si basino solo sull'identità assoluta, ma sulle loro **caratteristiche chimico-fisiche**.

Se una coppia S/T si trova appaiata nell'allineamento, lo score dovrebbe essere più alto rispetto ad una coppia S/W.
Inoltre alcuni amino acidi sembrano maggiormente conservati di altri, quindi un allineamento C/C dovrebbe essere più premiato rispetto ad un A/A.

Caratteristiche chimico-fisiche comuni

- **Aromatici**: WYF
- **Idrofobici**: VILMA
- **Polari carichi**: KRDE
- **Polari non carichi**: HCSTQNPG

Allineamento di sequenze: matrici di sostituzione

Una **matrice di sostituzione** è una griglia 20×20 in cui per ogni posizione si riporta il punteggio di quella specifica coppia di amino acidi.

PAM250 substitution matrix

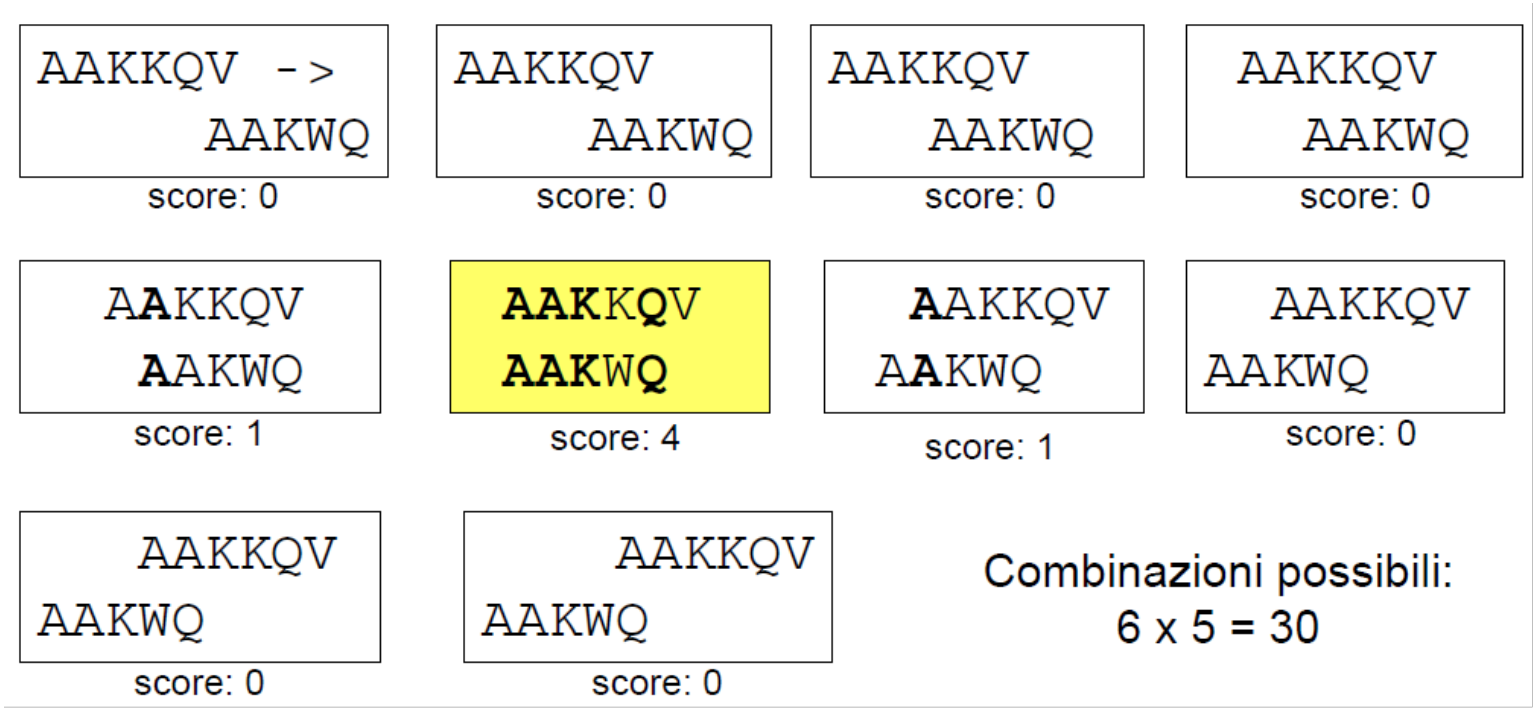
	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	2																			
R	-2	6																		
N	0	0	2																	
D	0	-1	2	4																
C	-2	-4	-4	-5	4															
Q	0	1	1	2	-5	4														
E	0	-1	1	3	-5	2	4													
G	1	-3	0	1	-3	-1	0	5												
H	-1	2	2	1	-3	3	1	-2	6											
I	-1	-2	-2	-2	-2	-2	-2	-3	-2	5										
L	-2	-3	-3	-4	-6	-2	-3	-4	-2	2	6									
K	-1	3	1	0	-5	1	0	-2	0	-2	-3	5								
M	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	6							
F	-4	-4	-4	-6	-4	-5	-5	-5	-2	1	2	-5	0	9						
P	1	0	-1	-1	-3	0	-1	-1	0	-2	-3	-1	-2	-5	6					
S	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	3				
T	1	-1	0	0	-2	-1	0	0	-1	0	-2	0	-1	-2	0	1	3			
W	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-3	-4	0	-6	-2	-5	17		
Y	-3	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-4	-2	7	-5	-3	-3	0	10	
V	0	-2	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	4

BLOSUM45 substitution matrix

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	5																			
R	-2	7																		
N	-1	0	7																	
D	-2	-1	2	7																
C	-1	-3	-2	-3	12															
Q	-1	1	0	0	-3	6														
E	-1	0	0	2	-3	2	6													
G	0	-2	0	-1	-3	-2	-2	7												
H	-2	0	1	0	-3	1	0	-2	10											
I	-1	-3	-2	-4	-3	-2	-3	-4	-3	5										
L	-1	-2	-3	-3	-2	-2	-2	-3	-2	2	5									
K	-1	3	0	0	-3	1	1	-2	-1	-3	-3	5								
M	-1	-1	-2	-3	-2	0	-2	-2	0	2	2	-1	6							
F	-2	-2	-2	-4	-2	-4	-3	-3	-2	0	1	-3	0	8						
P	-1	-2	-2	-1	-4	-1	0	-2	-2	-2	-3	-1	-2	-3	9					
S	1	-1	1	0	-1	0	0	0	-1	-2	-3	-1	-2	-2	-1	4				
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-1	-1	2	5			
W	-2	-2	-4	-4	-5	-2	-3	-2	-3	-2	-2	-3	-2	1	-3	-4	-3	15		
Y	-2	-1	-2	-2	-3	-1	-2	-3	2	0	0	-1	0	3	-3	-2	-1	3	8	
V	0	-2	-3	-3	-1	-3	-3	-3	-3	3	1	-2	1	0	-3	-1	0	-3	-1	5

Allineamento di sequenze: algoritmi esaustivi

Il metodo esaustivo più banale consiste nel far scorrere una sequenza sull'altra, utilizzando un matrice identità a fattore 1 per calcolare gli score.



La complessità è dell'ordine $O(n \cdot m)$ dove n ed m sono le lunghezze delle due stringhe da allineare. Se n è circa m si ha $O(n^2)$. Se volessimo allineare k sequenze la complessità è $O(n^k)$.
Problema: non stiamo ancora considerando i gap! $\binom{n+m}{m} \approx 2^{m+n}$

Allineamento di sequenze: il problema dei gap

Date due sequenze A e B con elementi (a_1, a_2, a_3, a_L) e (b_1, b_2, b_3, b_L).

Definiamo lo **score senza gap** come $\sum_{i=1}^{L} S(a_i, b_i)$, dove s è il punteggio assegnato dalla matrice di sostituzione.

```
 IPLMTRWDQEQESDFGHKLPIYTREWCTRG
 ||||||| |           ||||||
CHKIPLMTRWPQQESDFGHKLPVIYTREW
```

Miglior allineamento senza gap: score 14

```
 IPLMTRWDQEQESDFGHKLP-IYTREWCTRG
 ||||||| | |||||||||| ||||||
CHKIPLMTRWPQ-QESDFGHKLPVIYTREW
```

Miglior allineamento con gap: score 24 – (2 x 5) = 14

γ + δ [len (j) –1] = 5

Allineamento di sequenze: il problema dei gap

Si può assegnare una penalità all'inserimento dei gap e alla loro estensione cosicché l'algoritmo prenda in considerazione solo inserimenti di gap veramente vantaggiosi.

Definiamo lo **score con gap** come

$$\sum_{i=1}^{L} S(a_i, b_i) - \sum_{j=1}^{G} (\gamma + \delta[len(j) - 1])$$

dove:

- G è il numero di gap inseriti
- γ è la penalità per l'apertura di un gap
- δ è la penalità per l'allungamento di un gap

Allineamento pairwise: globale

- L'**allineamento globale** corrisponde all'allineamento con il **massimo numero di elementi** (aminoacidi o nucleotidi) allineati tra le due sequenze, favorendo così gli **allineamenti estesi** a tutta la sequenza (cioè **similarità globali**).
- Le sequenze sono allineate secondo la loro **intera lunghezza** (inserimento di gap se hanno lunghezze diverse)
- Algoritmo di **Needelman & Wunsch** (1970)

Allineamento pairwise globale: Needleman-Wunsch

- **Obiettivo**
  Date sue sequenze A e B, determinare il miglior allineamento (avente il maggior grado di similarità tra le sequenze considerate) che comprenda ogni elemento della sequenza A ed ogni elemento della sequenza B

- **Costrizioni proprie dell'allineamento globale**
  - Lo score di similarità calcolato confrontando A e B, $S(A, B)$, deve essere uguale allo score di similarità che si ottiene confrontando B con A, ovvero $S(A, B) = S(B, A)$
  - lo score di similarità tra due sequenze A e B, $S(A, B)$, deve essere migliore o uguale alla somma degli score di similarità ottenuti confrontando sia A che B con una terza sequenza C:
    $S(A, B) \leq S(A, C) + S(B, C)$

Allineamento pairwise globale: Needleman-Wunsch

Date due sequenze A e B di lunghezza rispettivamente n ed m, viene costruita una matrice $n \times m$.

Nel caso più semplice i valori della matrice vengono inizializzati come segue:

- 1 se i caratteri sono identici
- 0 altrimenti

	A	B	C	N	Y	R	Q	C	L	C	R	P	M
A	1												
Y					1								
C			1					1		1			
Y					1								
N				1									
R						1					1		
C			1					1		1			
K													
C			1					1		1			
R						1					1		
B		1											
P												1	

Allineamento pairwise globale: Needleman-Wunsch

Una volta inizializzata, la matrice viene trasformata a partire dalla cella in basso a destra.

Trasformazione: aggiungere allo score della cella corrente il massimo valore delle celle appartenenti alla coppia riga-colonna avente l'angolo in alto a sinistra posizionato in basso e a destra della cella considerata.

	A	B	C	N	Y	R	Q	C	L	C	R	P	M
A	1												
Y					1								
C			1					1		1			
Y					1								
N				1									
R						1					1		
C			1					1		1			
K													
C			1					1		1			
R						1					2	0	0
B	1	2	1	1	1	1	1	1	1	1	1	0	0
P	0	0	0	0	0	0	0	0	0	0	0	1	0

Allineamento pairwise globale: Needleman-Wunsch

Una volta inizializzata, la matrice viene trasformata a partire dalla cella in basso a destra.

Trasformazione: aggiungere allo score della cella corrente il massimo valore delle celle appartenenti alla coppia riga-colonna avente l'angolo in alto a sinistra posizionato in basso e a destra della cella considerata.

	A	B	C	N	Y	R	Q	C	L	C	R	P	M
A	1												
Y					1								
C			1					1		1			
Y					1								
N				1									
R						5	4	3	3	2	2	0	0
C	3	3	4	3	3	3	3	4	3	3	1	0	0
K	3	3	3	3	3	3	3	3	3	2	1	0	0
C	2	2	3	2	2	2	2	3	2	3	1	0	0
R	2	1	1	1	1	2	1	1	1	1	2	0	0
B	1	2	1	1	1	1	1	1	1	1	1	0	0
P	0	0	0	0	0	0	0	0	0	0	0	1	0

Allineamento pairwise globale: Needleman-Wunsch

Una volta inizializzata, la matrice viene trasformata a partire dalla cella in basso a destra.

Trasformazione: aggiungere allo score della cella corrente il massimo valore delle celle appartenenti alla coppia riga-colonna avente l'angolo in alto a sinistra posizionato in basso e a destra della cella considerata.

	A	B	C	N	Y	R	Q	C	L	C	R	P	M
A	8	7	6	6	5	4	4	3	3	2	1	0	0
Y	7	7	6	6	6	4	4	3	3	2	1	0	0
C	6	6	7	6	5	4	4	4	3	3	1	0	0
Y	6	6	6	5	6	4	4	3	3	2	1	0	0
N	5	5	5	6	5	4	4	3	3	2	1	0	0
R	4	4	4	4	4	5	4	3	3	2	2	0	0
C	3	3	4	3	3	3	3	4	3	3	1	0	0
K	3	3	3	3	3	3	3	3	3	2	1	0	0
C	2	2	3	2	2	2	2	3	2	3	1	0	0
R	2	1	1	1	1	2	1	1	1	1	2	0	0
B	1	2	1	1	1	1	1	1	1	1	1	0	0
P	0	0	0	0	0	0	0	0	0	0	0	1	0

Allineamento pairwise globale: Needleman-Wunsch

Traceback: partendo dal valore massimo si procede verso la cella in basso a destra seguendo il percorso che realizza il punteggio più alto.

	A	B	C	N	Y	R	Q	C	L	C	R	P	M
A	8	7	6	6	5	4	4	3	3	2	1	0	0
Y	7	7	6	6	6	4	4	3	3	2	1	0	0
C	6	6	7	6	5	4	4	4	3	3	1	0	0
Y	6	6	6	5	6	4	4	3	3	2	1	0	0
N	5	5	5	6	5	4	4	3	3	2	1	0	0
R	4	4	4	4	4	5	4	3	3	2	2	0	0
C	3	3	4	3	3	3	3	4	3	3	1	0	0
K	3	3	3	3	3	3	3	3	3	2	1	0	0
C	2	2	3	2	2	2	2	3	2	3	1	0	0
R	2	1	1	1	1	2	1	1	1	1	2	0	0
B	1	2	1	1	1	1	1	1	1	1	1	0	0
P	0	0	0	0	0	0	0	0	0	0	0	1	0

Allineamento pairwise globale: Needleman-Wunsch

La soluzione non è unica, infatti possono esistere più percorsi con lo stesso punteggio.

	A	B	C	N	Y	R	Q	C	L	C	R	P	M
A	8	7	6	6	5	4	4	3	3	2	1	0	0
Y	7	7	6	6	6	4	4	3	3	2	1	0	0
C	6	6	7	6	5	4	4	4	3	3	1	0	0
Y	6	6	6	5	6	4	4	3	3	2	1	0	0
N	5	5	5	6	5	4	4	3	3	2	1	0	0
R	4	4	4	4	4	5	4	3	3	2	2	0	0
C	3	3	4	3	3	3	3	4	3	3	1	0	0
K	3	3	3	3	3	3	3	3	3	2	1	0	0
C	2	2	3	2	2	2	2	3	2	3	1	0	0
R	2	1	1	1	1	2	1	1	1	1	2	0	0
B	1	2	1	1	1	1	1	1	1	1	1	0	0
P	0	0	0	0	0	0	0	0	0	0	0	1	0

```
ABCNY-RQCLCR-PM
| | | | | | | |
AYC-YNR-CKCRBP-
```

	A	B	C	N	Y	R	Q	C	L	C	R	P	M
A	8	7	6	6	5	4	4	3	3	2	1	0	0
Y	7	7	6	6	6	4	4	3	3	2	1	0	0
C	6	6	7	6	5	4	4	4	3	3	1	0	0
Y	6	6	6	5	6	4	4	3	3	2	1	0	0
N	5	5	5	6	5	4	4	3	3	2	1	0	0
R	4	4	4	4	4	5	4	3	3	2	2	0	0
C	3	3	4	3	3	3	3	4	3	3	1	0	0
K	3	3	3	3	3	3	3	3	3	2	1	0	0
C	2	2	3	2	2	2	2	3	2	3	1	0	0
R	2	1	1	1	1	2	1	1	1	1	2	0	0
B	1	2	1	1	1	1	1	1	1	1	1	0	0
P	0	0	0	0	0	0	0	0	0	0	0	1	0

```
ABC-NYRQCLCR-PM
| | | | | | | |
AYCYN-R-CKCRBP-
```

Entrambi i percorsi (allineamenti) realizzano il massimo punteggio.

Allineamento pairwise globale: Needleman-Wunsch

NB: non sono ammessi spostamenti diretti verso destra o verso il basso.

	A	B	C	N	J	R	O	C	L	C	R	P	M
A	8	7	6	6	5	4	4	3	3	2	1	0	0
J	7	7	6	6	6	4	4	3	3	2	1	0	0
C	6	6	7	6	5	4	4	4	3	3	1	0	0
J	6	6	6	5	6	4	4	3	3	2	1	0	0
N	5	5	5	6	5	4	4	3	3	2	1	0	0
R	4	4	4	4	4	5	4	3	3	2	2	0	0
C	3	3	4	3	3	3	3	4	3	3	1	0	0
K	3	3	3	3	3	3	3	3	3	2	1	0	0
C	2	2	3	2	2	2	2	3	2	3	1	0	0
R	2	1	1	1	1	2	1	1	1	1	2	0	0
B	1	2	1	1	1	1	1	1	1	1	1	0	0
P	0	0	0	0	0	0	0	0	0	0	0	1	0

Allineamento pairwise globale: Hirschberg

L'algoritmo di Hirschberg definisce l'allineamento pairwise globale in termini di programmazione dinamica.

Osservazioni:

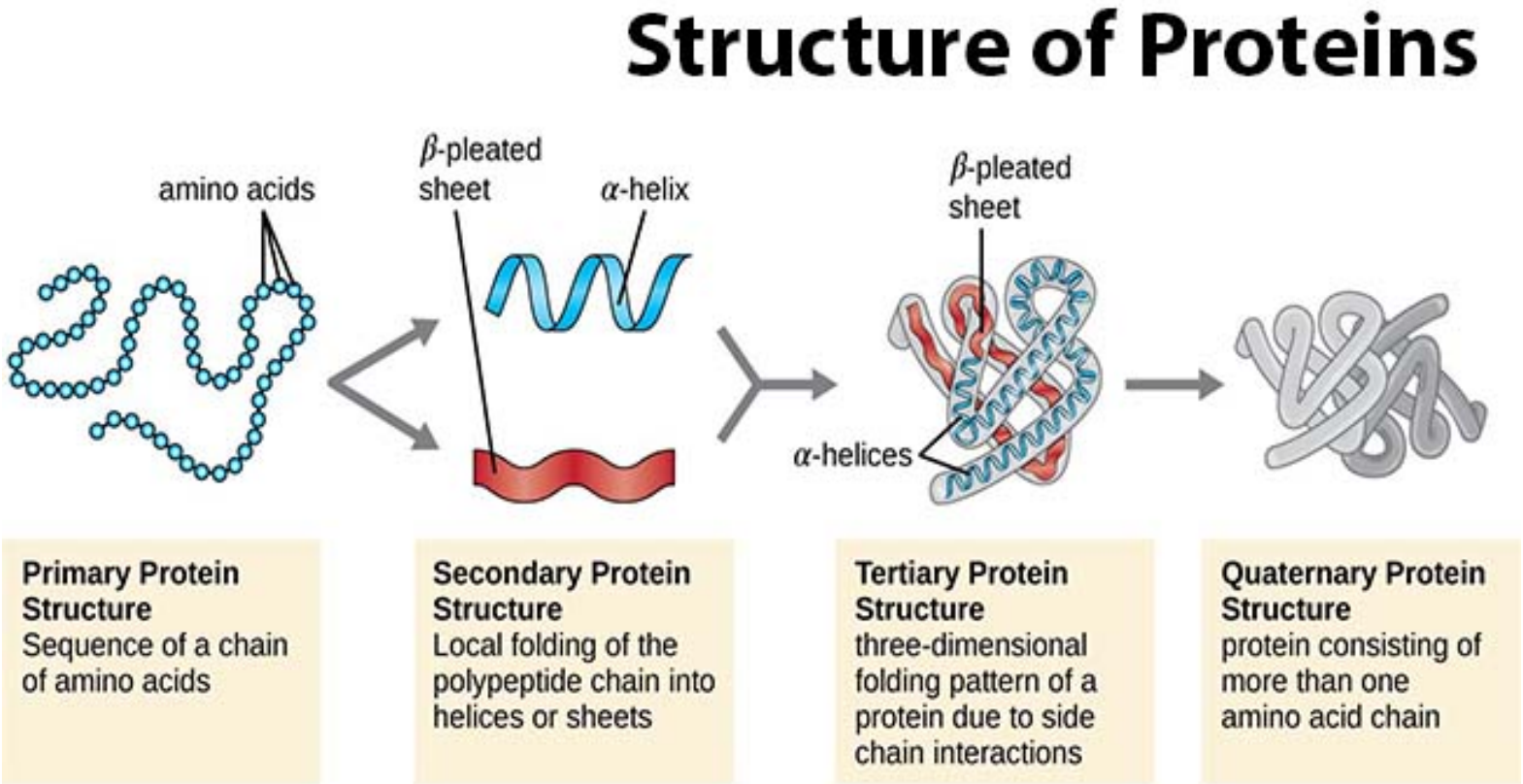
- nell'algoritmo di Needleman-Wunsch, per il calcolo della matrice si ha bisogno di memorizzare solo la riga corrente e quelle al passo precedente
- sia $NW(A, B)$ un allineamento ottimale per Needleman-Wunsch e sia $A = A^l + A^r$ una partizione arbitraria per A, allora esiste una partizione $B = B^l + B^r$ tale che $NW(A, B) = NW(A^l, B^l) + NW(A^r, B^r)$

Definiamo una matrice $F_{i,j}$ di dimensioni (al più) $n \times m$, tale che:

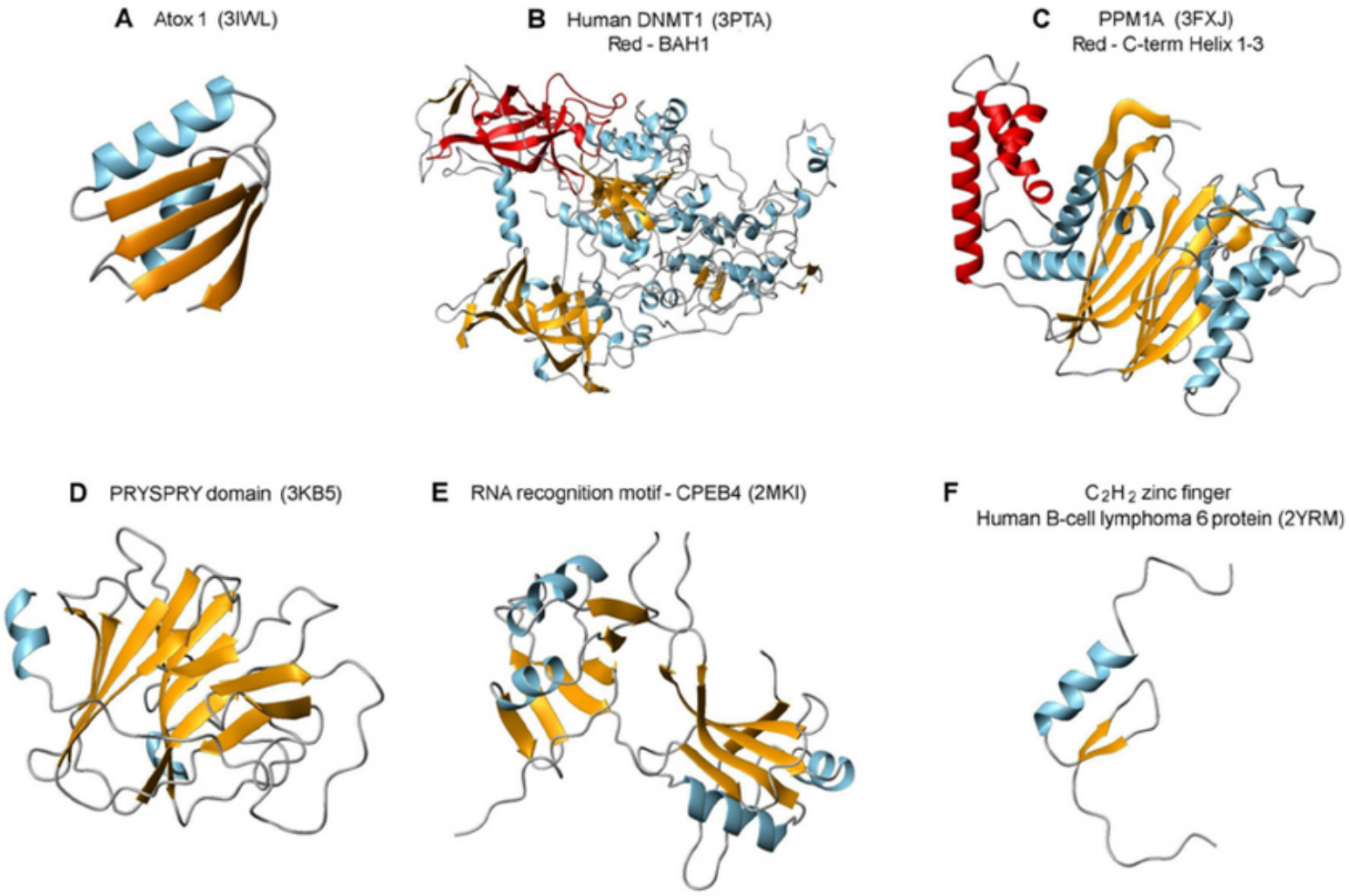
- $F0, j = d \cdot j$
- $F_{i,0} = d \cdot i$
- $F_{i,j} = max(F_{i-1,j-1} + S(A, B)\ ,\ F_{i,j-1} + d\ ,\ F_{i-1,j} + d)$

dove d è il costo dovuto ad un gap.

Struttura delle proteine

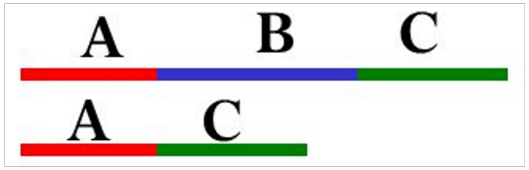


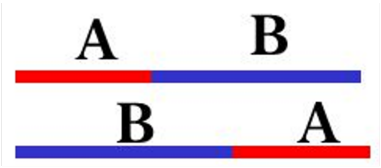
Domini proteici

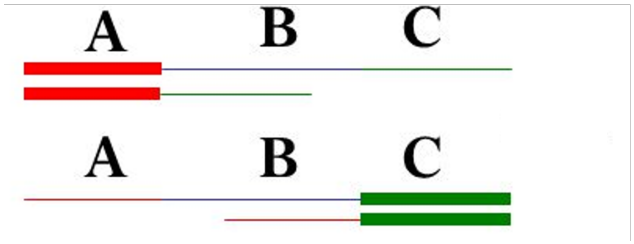


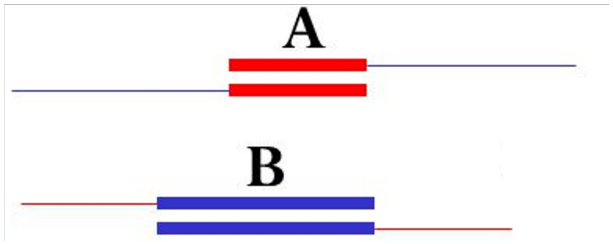
Allineamento pairwise locale

- L'**allineamento locale** considera anche l'**estensione della regione simile**. In questo caso, se abbiamo un dato numero di aminoacidi uguali, daremo un punteggio tanto maggiore quanto più concentrata sarà la regione in cui si estende la similarità. Così, molto spesso, si favoriscono **similarità non estese** a tutta la sequenza, ma localizzate in regioni circoscritte (**similarità locali**).
- Algoritmo di **Smith & Waterman** (1981)









Allineamento pairwise locale: Smith-Waterman

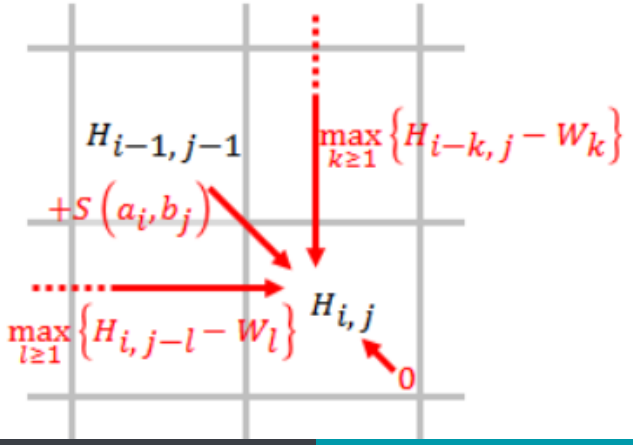
- **Obiettivo**: Date due sequenze relativamente lunghe $S_A = a_1, a_2, \ldots, a_n$ e $S_B = b_1, b_2, \ldots b_m$, determinare la **sottosequenza** della prima sequenza che realizza il **maggior grado di similarità** con una sottosequenza della seconda sequenza.

- **Inizializzazione**
  Costruire una **matrice** $H_{i,j}$ con $n-1$ righe ed $m+1$ colonne. Tale matrice viene inizializzata ponendo: $H_{i,0} = H_{0,j}$ per $0 \leq i \leq n$ e $0 \leq j \leq m$.

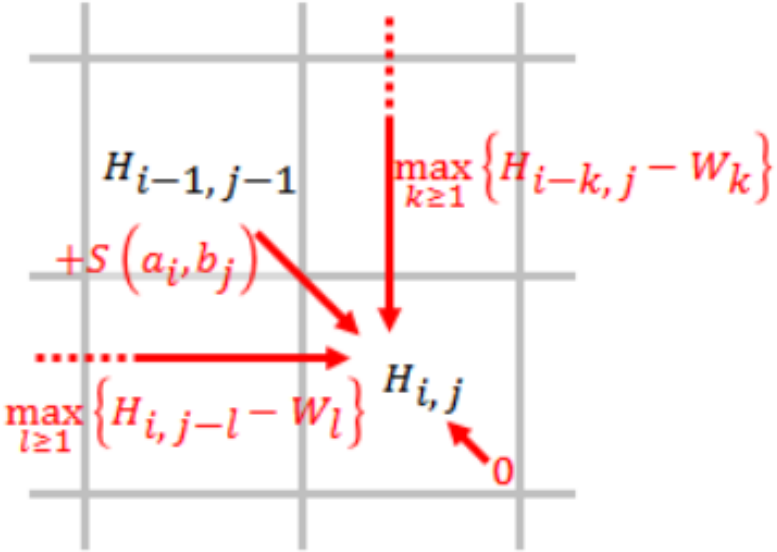
		T	G	T	...
	0	0	0	0	
G	0				
G	0				
⋮					

Allineamento pairwise locale: Smith-Waterman

- **Calcolo**
  Per ciascuna coppia di elementi $a_i \in S_A$ e $b_j \in S_B$ si definisce un **punteggio di similarità** s.
  Per il confronto di sequenze nucleotidiche tale punteggio viene generalmente definito come:
  - $s(a_i, b_j) = \alpha$ se $a_i = b_j$ (usualmente > 0)
  - $s(a_i, b_j) = \beta$ se $a_i \neq b_j$ (usualmente ≤ 0)
  - $W_k = \gamma + \delta(k-1)$ dove
    - γ è il costo fisso di apertura di un gap
    - δ penalità aggiunta per estendere un gap di lunghezza k
- I valori della **matrice** $H_{i,j}$ vengono determinati secondo l'equazione
  $H_{i,j} = max(H_{i-1,j-1} + s(a_i, b_j)\ ,\ H_{i-1,j} - W_k\ ,\ H_{i,j-1} - W_k\ ,\ 0)$



Allineamento pairwise locale: Smith-Waterman



Questa formula **ricorsiva** considera tutte le possibili terminazioni di **segmenti allineati** alle cui estremità si trovano a_i e b_j.

- $H_{i-1,j-1} + s(a_i, b_j)$ rappresenta un allungamento dell'allineamento corrente tramite match o mismatch.
- $H_{i-1,j} - W_k$ rappresenta un allungamento tramite gap su S_A
- $H_{i,j-1} - W_k$ rappresenta un allungamento tramite gap su S_B
- 0 significa ce non c'è più similarità tra le due sequenze, quindi terminiamo

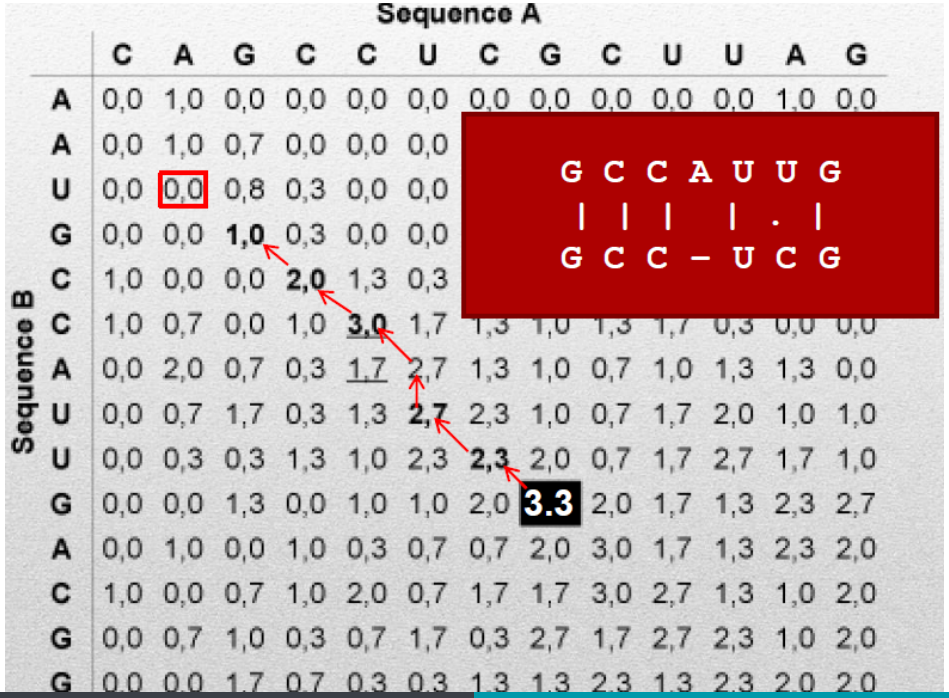
Allineamento pairwise locale: Smith-Waterman

- **Backtracking**

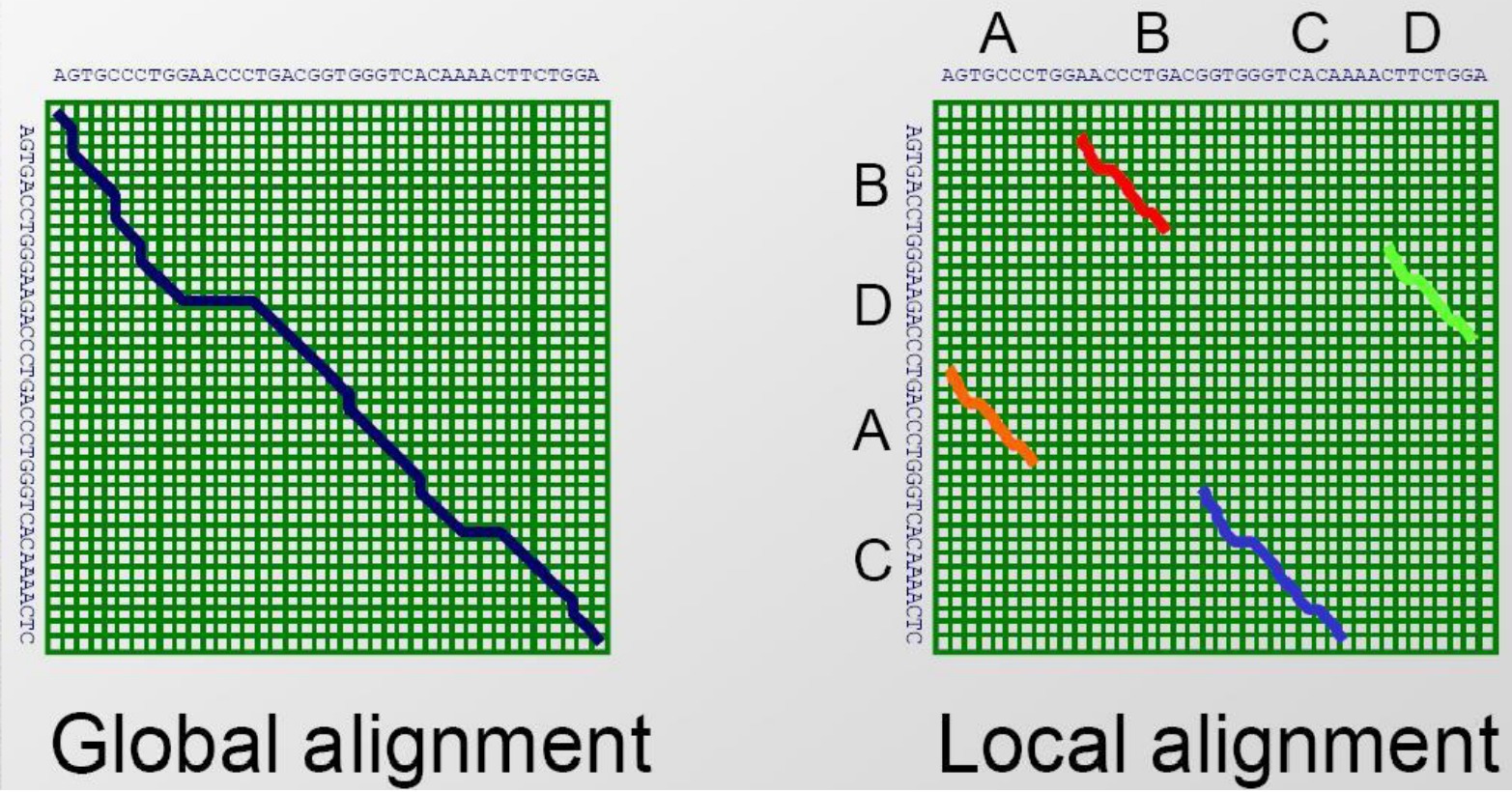
  Una volte riempita la matrice, lo scopo è quello di trovare gli allineamenti locali migliori.

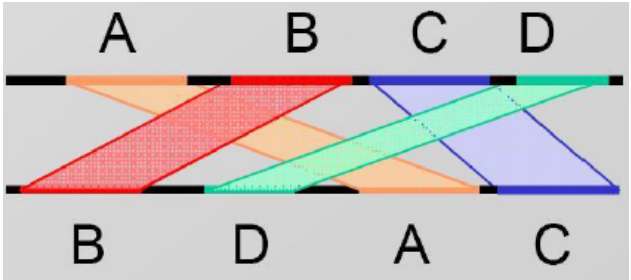
  Iniziamo dall'allineamento di score massimo, ovvero la cella della matrice per cui abbiamo ottenuto il valore massimo.

  Risaliamo quindi la matrice seguendo la cella di score massimo tra $H_{i-1,j-1}$, $H_{i-1,j}$ e $H_{i,j-1}$, fino ad arrivare ad uno 0, che ci indicherà l'inizio dell'allineamento locale

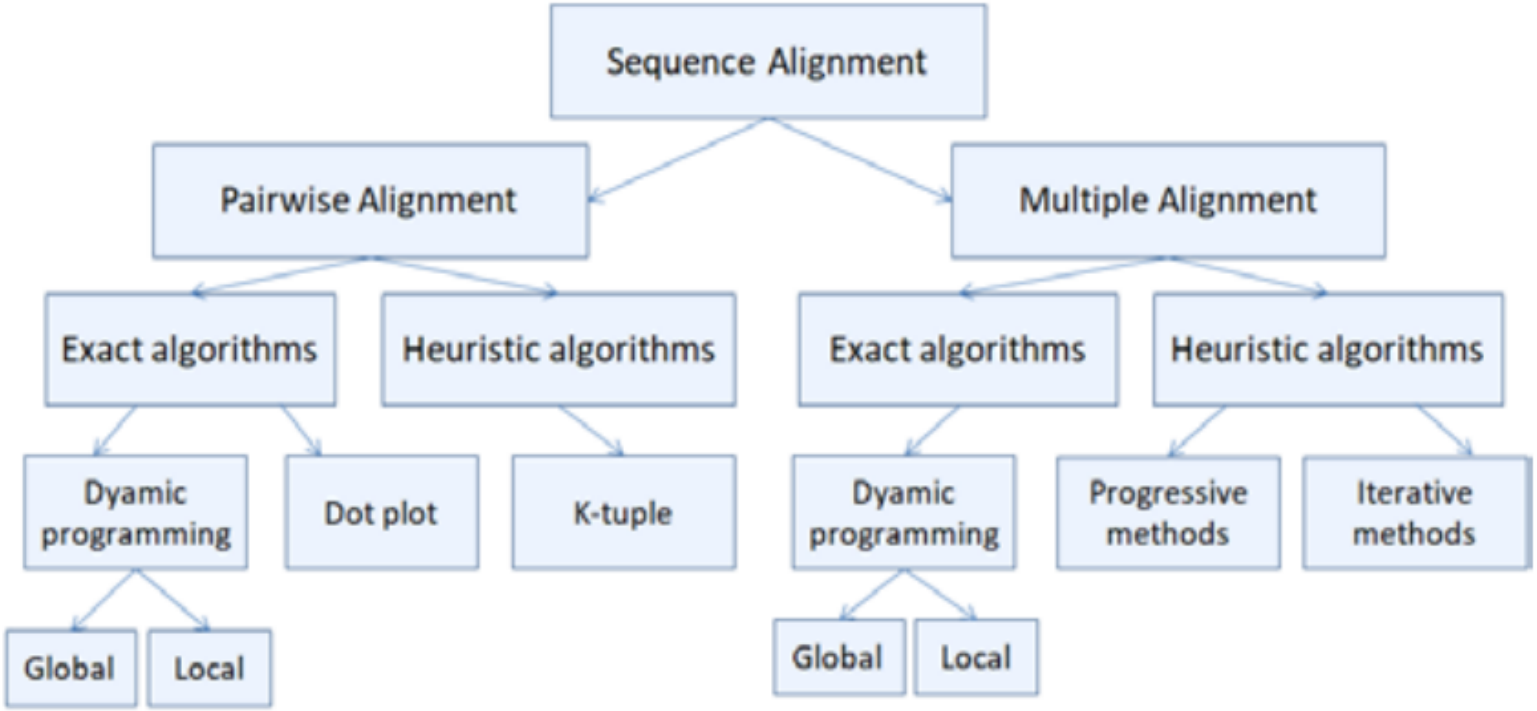


Allineamento pairwise locale: Smith-Waterman



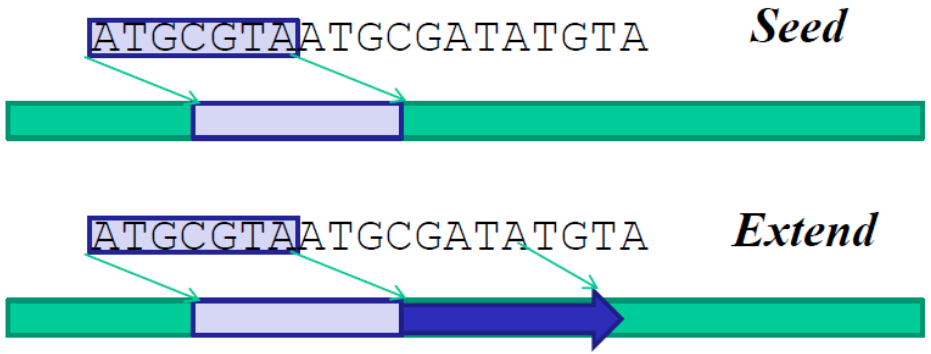


Allineamento di sequenze



BLAST

BLAST (Basic Local Alignment Search Tool) è un algoritmo euristico per l'allineamento locale o per cercare all'interno di una collezione di sequenze target la sequenza target più simile ad usa sequenza query data.
L'idea centrale dell'algoritmo di BLAST è di limitare l'attenzione a coppie di seg-
menti in cui si allineano parole di lunghezza w con un punteggio di almeno T.



BLAST

- Fase 0: si estraggano dalle sequenze del database tutte le parole di lunghezza w e se ne memorizza la posizione in ognuna delle sequenze
- Fase 1: compilare una lista di coppie di parole (di lunghezza w $= 3$) con un punteggio oltre la soglia T
- Fase 2: scansione del database per le voci che corrispondono alla lista compilata (con punteggio maggiore della soglia T).
- Fase 3: quando si riesce a trovare una corrispondenza (abbinamento tra una "parola" con punteggio $>$ T al database), estendere l'allineamento in entrambe le direzioni.
  Dopo aver identificato una corrispondenza del seed con la sequenza subject l'allineamento viene raffinato utilizzando l'algoritmo Smith-Waterman

BLAST score

E 'importante valutare la **significatività statistica** dei risultati della ricerca.
I punteggi seguono una **distribuzione del valore estremo** (EVD), piuttosto che un distribuzione normale, il cui parametro principale è la **costanze di decadimento** λ.
Il **valore atteso** E: è il numero di allineamenti (high scoring segment pairs o HSP) con punteggio maggiore o uguale a un punteggio S che dovrebbero verificarsi per caso in quella ricerca sul database. Si può pensarlo come un indice di **incertezza**.

$$E = KMNe^{-\lambda S}$$

con M lunghezza della query, N lunghezza della sequenza target, S lo score di allineamento, λ parametro che dipende dallo scoring system, K parametro che dipende dal database usato.

BLAST score

- Il valore di E decresce esponenzialmente con l'aumentare S. Valori più elevati di S corrispondono a migliori allineamenti e infatti hanno **E values** più bassi.
- Ottenere un allineamento con E = 1 significa che esiste un altro allineamento con lo stesso score S che è risultato per caso.
- La stessa ricerca, su un database più piccolo o più grande, anche se restituisce lo stesso allineamento deve avere un valore di E diverso (ciò dipende da K)
- I punteggi in **bit** sono paragonabili tra diverse ricerche (es. diverse matrici di sostituzione o diverse banche dati) perché sono **normalizzati**, adatti quindi al paragone anche se originati tramite diverse matrici di punteggio e database di dimensione diversa

$$S' = bitscore = (\lambda S - lnK)/ln2$$

BLAST score

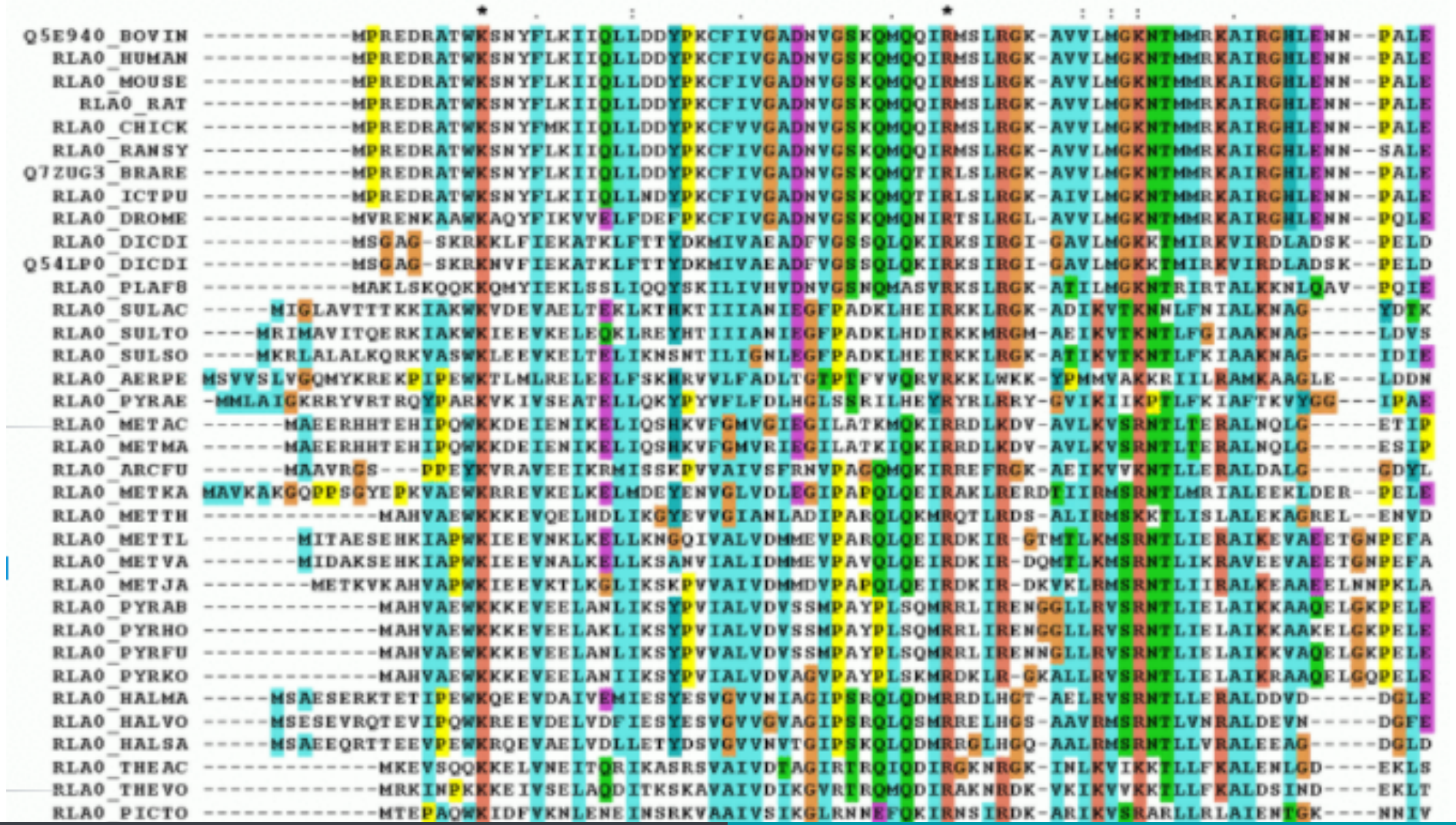
Il valore atteso E è il numero di allineamenti con punteggio maggiore o uguale a un punteggio S che dovrebbero verificarsi per caso in un database di ricerca.
Il **p-value** è un modo diverso (ma equivalente) di rappresentare la significatività di un risultato

$$p = 1 - e^{-E}$$

Per valori molto piccoli E e p sono molto simili. Tuttavia tra 1 e 10 il valore di E è più chiaro (poichè riflette un numero di hits)

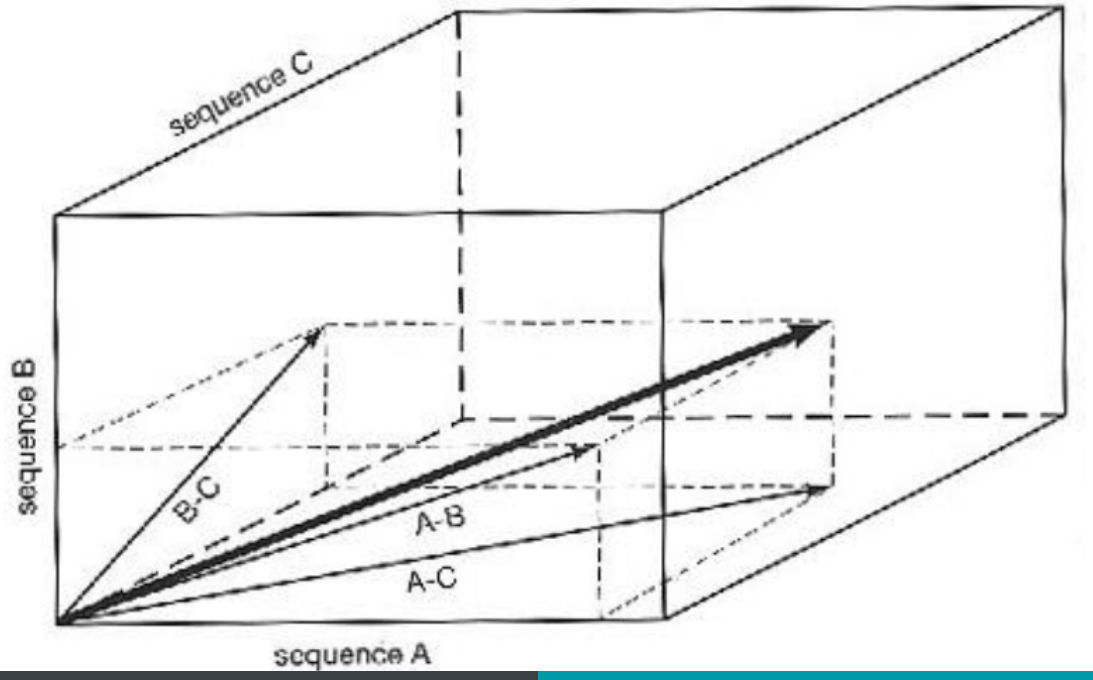
Allineamento multiplo di sequenze

Ammettiamo di aver formato dei cluster di stringhe e di voler trovare una stringa consenso. Per farlo ci serve allineare tutte le stringhe in un unico allineamento.



Allineamento multiplo di sequenze ottimale

Sulla superficie del cubo si hanno le matrici di confronto a coppie tra le sequenze A-B, B-C e A-C. L'allineamento ottimale di tre sequenze (A-B-C) richiede il riempimento del cubo e la valutazione di tutte le possibili mosse all'interno del cubo.
La complessità di questo algoritmo è $O(L^N)$, dove L è la lunghezza e N il numero delle sequenze.
Un algoritmo completo di programmazione dinamica è utilizzabile solo nel caso di tre sequenze.



Allineamento multiplo di sequenze ottimale

Più generalmente, sia $X = \{x_1, \ldots, x_k\}$ un insieme di k sequenze da allineare.
Sia $\alpha(i_1, \ldots, i_k)$ il massimo punteggio di un allineamento delle sottosequenze iniziali $x_1[1 \ldots i_1], \ldots, x_k[1 \ldots i_k]$. Tale punteggio può essere calcolato come:

$$\alpha(i_1, \ldots, i_k) = \max \begin{cases} \alpha(i_1 - 1, i_2 - 1, \ldots, i_k - 1) & + \quad S(m_{1i_1}, m_{2i_2}, \ldots, m_{ki_k}) \\ \alpha(i_1, i_2 - 1, \ldots, i_k - 1) & + \quad S(-, m_{2i_2}, \ldots, m_{ki_k}) \\ \alpha(i_1 - 1, i_2, \ldots, i_k - 1) & + \quad S(m_{1i_1}, -, \ldots, m_{ki_k}) \\ & \vdots \\ \alpha(i_1 - 1, i_2 - 1, \ldots, i_k) & + \quad S(m_{1i_1}, m_{2i_2}, \ldots, -) \\ \alpha(i_1, i_2, i_3 - 1 \ldots, i_k - 1) & + \quad S(-, -, m_{3i_3}, \ldots, m_{ki_k}) \\ & \vdots \\ \alpha(i_1, i_2 - 1, \ldots, i_{k-1} - 1, i_k) & + \quad S(-, m_{2i_2}, \ldots, -) \\ & \vdots \end{cases}$$

Allineamento multiplo di sequenze progressivo

Allineamento di tre sequenze A, B, C per passi successivi.

Allineamento della coppia A-B

```
DGEKFGPPQRSGQRSG   SeqA
|||.|||  ||||.|||
DGERFGP-QRSGNRSG   SeqB
```

Aggiunta della coppia C al precedente allineamento

```
consenso ←→ DHEKFGSSQRSGQRSG   SeqC
```

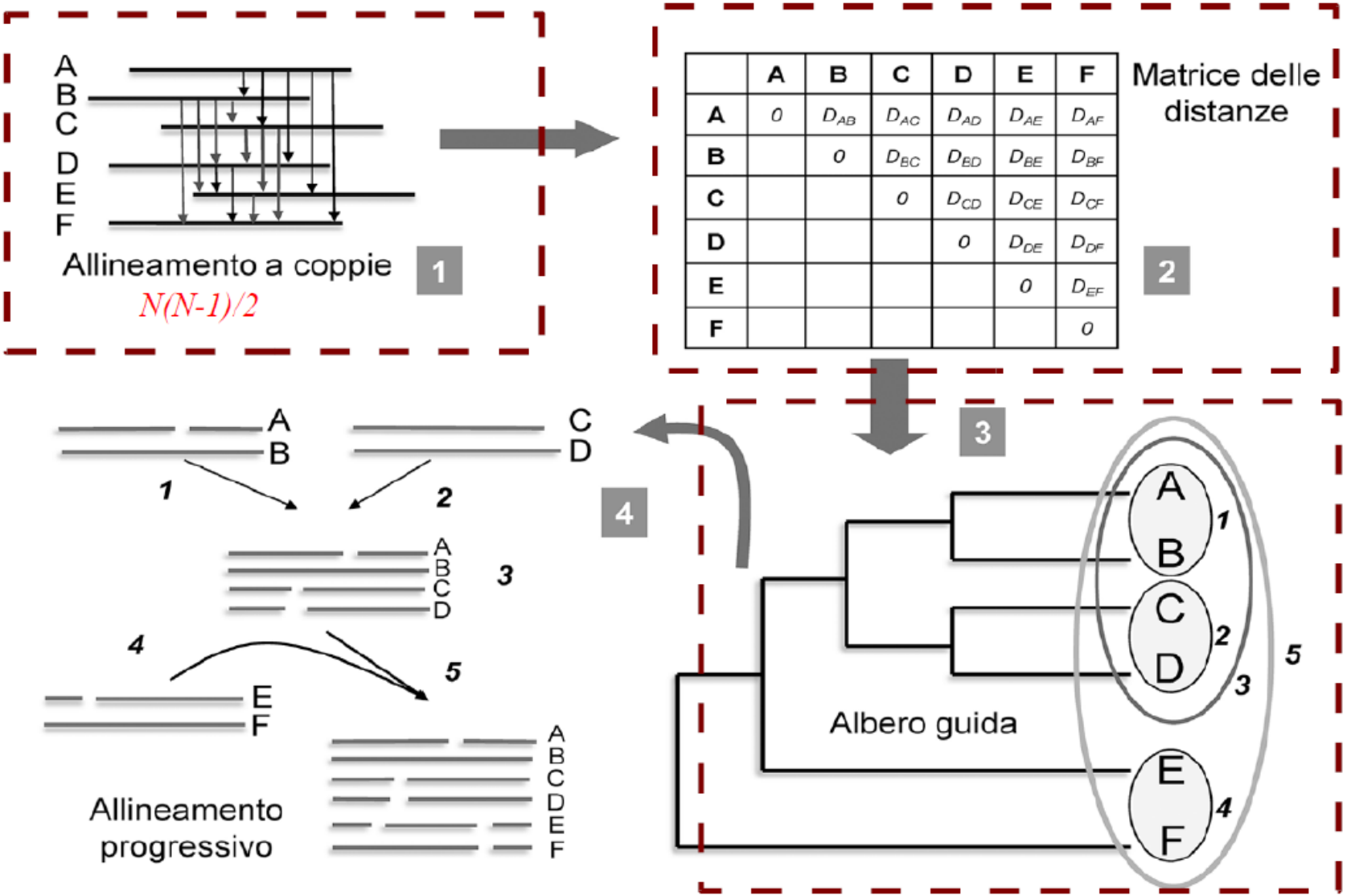
Allineamento multiplo

```
DGEKFGPPQRSGQRSG   SeqA
|||.|||  ||||.|||
DGERFGP-QRSGNRSG   SeqB
|  |.||.  .|||.|||
DHEKFGSSQRSGQRSG   SeqC
```

Allineamento multiplo di sequenze progressivo

- Sia dato un insieme di N sequenze (dataset) disposte a caso, non allineate
- Allineare tutte le proteine con tutte le proteine, a coppie ($N(N-1)/2$ allineamenti)
- Determinare un albero guida basato sui punteggi di similarità di tutte le coppie
  - tramite Neighbor-joining o UPGMA(Unweighted pair group method with arithmentic mean)
- A partire dalla coppia più simile, determinare le colonne conservate, e allineare la coppia successiva mantenendo queste colonne e ricalcolando lo score complessivo
- Si ottiene un output un insieme di N sequenze allineate

Clustal: schema



	A	B	C	D	E	F
A	0	D_{AB}	D_{AC}	D_{AD}	D_{AE}	D_{AF}
B		0	D_{BC}	D_{BD}	D_{BE}	D_{BF}
C			0	D_{CD}	D_{CE}	D_{CF}
D				0	D_{DE}	D_{DF}
E					0	D_{EF}
F						0

Feng-Doolittle

Alla base dell'algoritmo di Clustal vi è la metodologia di Feng-Doolittle.
La motivazione principale è che gli algoritmi per la costruzione dell'albero richiedono una matrice delle distanze mentre gli algoritmi di allineamento forniscono score di similarità.
E' quindi necessario convertire da similarità a distanze.

Sia $S(a, b)$ la similarità tra le sequenze a e b e sia $S_{a,b}^{max}$ la massima similarità possibile.
Sia S_{rand} lo score di similarità atteso per due sequenze random a',b' aventi la stessa lunghezza di a e b.
Definiamo $S_{a,b}^{eff} = \frac{S(a,b)-S_{rand}}{S_{a,b}^{max}-S_{rand}}$

- è sempre tra 0 e 1
- converge lentamente a zero per sequenze evolutivamente sempre più lontane

Quindi, definiamo la distanza lineare come $D(a, b) = -logS_{a,b}^{eff}$