Università degli Studi di Catania
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea Magistrale in Informatica

Giovanni Micale

LGA Gibbs Sampler: un algoritmo per l'allineamento locale di network

Tesi di Laurea

Relatore:
Ch.mo Prof. A. Pulvirenti
Università di Catania

Correlatore:
Ch.mo Prof. A. Ferro
Università di Catania

Luglio 2011

LGA Gibbs Sampler: un algoritmo per l'allineamento locale di network

Giovanni Micale

Ringraziamenti

Ringrazio il prof. Pulvirenti e il prof. Ferro per avermi seguito durante l'ultimo anno in questo progetto e per avermi sempre sostenuto, anche nei momenti più difficili, fornendomi sempre suggerimenti utili al miglioramento del lavoro.

Ringrazio in maniera speciale anche la mia famiglia per il loro continuo supporto e incoraggiamento in questi anni.

Introduzione

LGA Gibbs Sampler è un algoritmo randomizzato, che si propone di risolvere il problema del *Local Graph Alignment* (LGA problem).

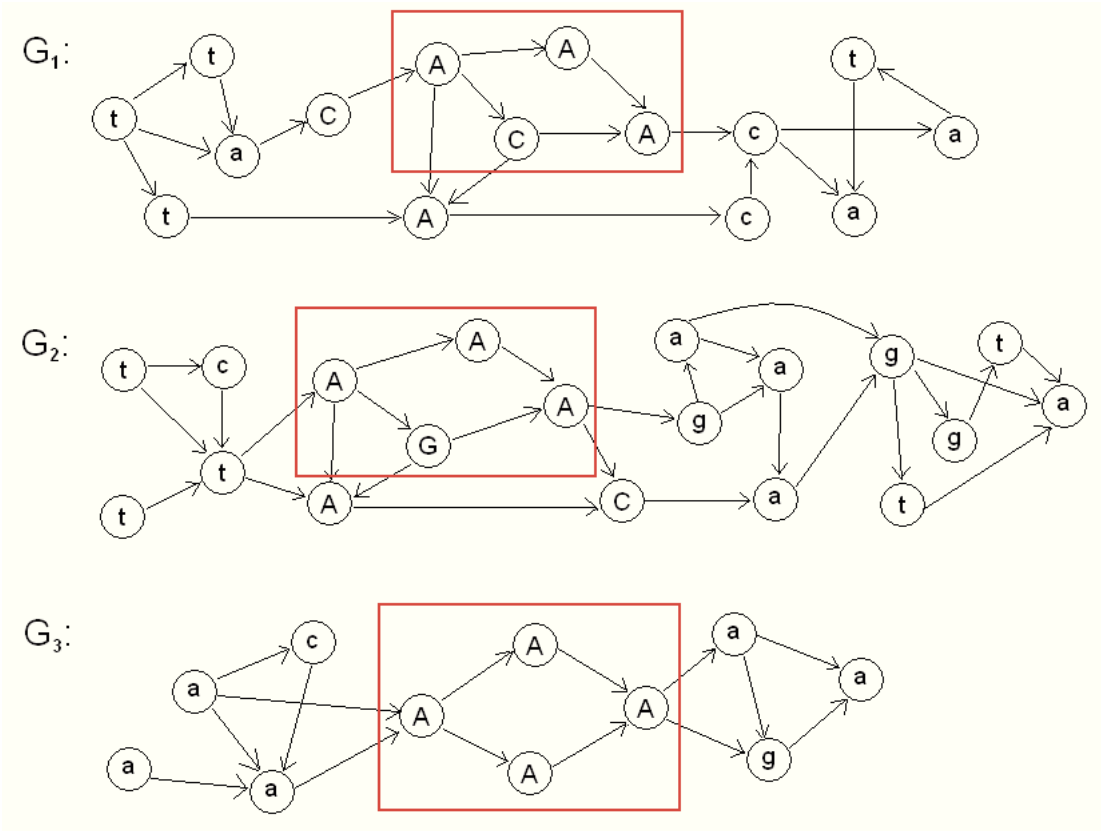


Figura 1: Esempio di allineamento

Dato un insieme di N grafi e un numero positivo W, al più uguale al minor numero di nodi in uno degli N grafi, il problema LGA consiste nel determinare un insieme di N finestre di sottografi di W nodi, una per ogni grafo,

che renda elevata la similarità locale tra i grafi e permetta di allinearli sulla base di questi pattern comuni.

Esempio 1. Siano $N = 3$ e $W = 4$. Consideriamo i grafi G_1, G_2 e G_3 di Fig. 1. Il pattern che massimizza la similarità locale tra G_1, G_2 e G_3 è il sottografo racchiuso nel rettangolo rosso (Fig. 2):

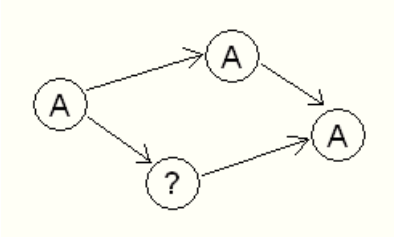


Figura 2: Pattern dell'allineamento

dove il metacarattere ? indica qualsiasi carattere (A, C, G, ecc.).

Nel primo capitolo si accennerà brevemente ai metodi Monte Carlo e si richiamerà il concetto di catena di Markov. Quindi si descriverà un esempio di metodo Monte Carlo per la costruzione di catene di Markov (MCMC), ovvero il Gibbs Sampling, nella sua forma più generale, come algoritmo di campionamento a partire da un insieme di n variabili.

Nel secondo capitolo si descriverà lo schema generale dell'algoritmo, evidenziando il modo in cui viene costruita la catena di Markov per l'allineamento e il modo in cui viene effettuata una transizione da uno stato all'altro della catena, attraverso un meccanismo di campionamento basato sul Gibbs Sampling.

Nel terzo capitolo si descriverà l'algoritmo per il calcolo delle finestre di dimensione fissata W in un grafo G, che costituisce il preprocessing dell'algoritmo.

Nel quarto capitolo si descriverà il calcolo del Likelihood Ratio (LR) di una finestra e, quindi, della probabilità di transizione da uno stato all'altro della catena di Markov.

Nel quinto capitolo si illustrerà il calcolo del Bit Score, ovvero il punteggio che valuta la qualità di un allineamento prodotto, allo scopo di stabilire al termine dell'algoritmo qual è il migliore allineamento ottenuto.

Nel sesto capitolo verrano illustrati i risultati ottenuti dall'algoritmo su un dataset di proteine, rappresentati come grafi in cui i nodi sono gli atomi e gli archi sono i legami molecolari, al variare del numero di reti e della dimensione W delle finestre. Inoltre, saranno descritti i parametri di input e

l'output corrispondente dell'algoritmo e ne sarà analizzato il comportamento nel caso migliore, in quello medio e in quello peggiore.

Nel settimo capitolo verrà illustrata un'ottimizzazione dell'algoritmo, basata sulla costruzione opportuna di un insieme ridotto di finestre di una certa dimensione e sulla ricerca, a partire da questo set ridotto, di finestre di dimensione maggiore. Verranno descritte tutte le procedure necessarie per ottenere questo metodo ottimizzato.

Nell'ottavo capitolo saranno illustrati i risultati di alcuni test effettuati per confrontare l'algoritmo *LGA Gibbs Sampler* con la versione ottimizzata, da cui si deduce che quest'ultima è più veloce della precedente e abbastanza accurata.

Infine, nel nono e ultimo capitolo si accennerà brevemente ad alcune possibili applicazioni del metodo e agli sviluppi futuri del progetto in esame.

Indice

Capitolo 1

Gibbs sampling

LGA Gibbs Sampler si basa sull'applicazione del *Gibbs Sampling* (cf. [1, 2]), un algoritmo di campionamento che appartiene alla classe dei metodi *Markov Chain Monte Carlo* (MCMC).

1.1 Metodo Monte Carlo

Il *metodo Monte Carlo* è un metodo statistico che consiste nel determinare la soluzione di un problema mediante la tecnica del campionamento casuale. Ciò si rivela utile nel caso in cui si ha a che fare con problemi computazionalmente difficili da risolvere in maniera esatta, cioè in modo deterministico, e quindi complessi da risolvere analiticamente.

Il metodo fu formalizzato da John Von Neumann e Stanislav Marcin Ulam verso la metà degli anni '40 (cf. [3]) durante lo studio di complessi problemi nucleari di natura deterministica, e fu ribattezzato 'di Monte Carlo' dallo stesso Von Neumann in riferimento al casinò di Monte Carlo, per sottolineare la natura probabilistica del metodo.

Sia P un problema da risolvere in maniera esatta e I la sua soluzione. Il metodo si articola in tre fasi:

1. Scelta casuale di N campioni rappresentativi nello spazio dei parametri di P, cioè N valori di una variabile casuale X, secondo una certa distribuzione di probabilità (ad es. uniforme);

2. Simulazione: sugli N campioni scelti effettuare misure del parametro di interesse del problema (quello da cui dipende la soluzione I di P);

3. Calcolo del valore medio di tali misure.

Il metodo funziona se il valore medio ottenuto alla fine converge alla soluzione esatta I.

Il metodo Monte Carlo ha applicazione in vari settori, quali l'analisi numerica (calcolo di integrali), la ricerca operativa (problemi di decisione, relativi al comportamento di un sistema), l'economia (stima del rendimento mensile di un titolo azionario) o la programmazione distribuita.

1.2 Catena di Markov

Una *catena di Markov* è una tripla (Q, $P(\pi_1)$, A) dove:

- Q è un insieme finito di k stati (o eventi). Ogni stato rappresenta, in generale, un insieme di n assegnamenti o valori assunti dalle variabili X_1, X_2, ..., X_n di un insieme X;
- π_i per $i = 1, 2, ...$ rappresenta lo stato della catena nel generico istante temporale i. $P(\pi_1)$ è la distribuzione di probabilità iniziale degli stati, cioè l'insieme delle probabilità di trovarsi in un certo stato all'istante di tempo iniziale;
- A è la matrice delle probabilità di transizione da uno stato all'altro della catena. Un generico elemento della matrice, a_{st}, rappresenta la probabilità di passare dallo stato s, nel generico istante di tempo i, allo stato t all'istante di tempo successivo $i + 1$.

Una catena di Markov del primo ordine è una catena di Markov in cui la probabilità di un evento dipende esclusivamente dal precedente. Ciò significa che:

$$\forall\, i = 1, 2, ... \quad P(\pi_{i+1}|\pi_1, ..., \pi_i) = P(\pi_{i+1}|\pi_i)$$

Da questo momento in poi, faremo sempre riferimento a catene di Markov del primo ordine. Dati, quindi, n eventi π_1, π_2, ..., π_n, dall'equazione precedente segue che la probabilità congiunta degli n eventi, ovvero la probabilità di osservare in istanti temporali successivi gli stati π_1, π_2, ..., π_n, è:

$$P(\pi_1, \pi_2, ..., \pi_n) = P(\pi_1) \times \prod_i P(\pi_{i+1}|\pi_1, ..., \pi_i) = P(\pi_1) \times \prod_i P(\pi_{i+1}|\pi_i)$$

Ciò consente, in maniera semplice, di analizzare la distribuzione di probabilità degli stati, ovvero di osservazione degli eventi, in istanti temporali successivi, a partire da una distribuzione iniziale $D_1 = P(\pi_1)$. Si può osservare, infatti, che la distribuzione di probabilità degli eventi:

- all'istante di tempo 1 è semplicemente D_1;
- all'istante di tempo 2 è $D_2 = D_1 \times A$;
- all'istante di tempo 3 è $D_3 = D_2 \times A = D_1 \times A^2$;
- ...;
- all'istante di tempo n è $D_n = D_1 \times A^{n-1}$;
- ...

Come si può osservare dal grafico di Fig. 3, all'aumentare di n la distribuzione di probabilità varia, fino a raggiungere valori stazionari, che non cambiano nel tempo. Si parla in tal caso di distribuzione stazionaria. Questo comportamento si può osservare qualunque sia la distribuzione iniziale D_1.

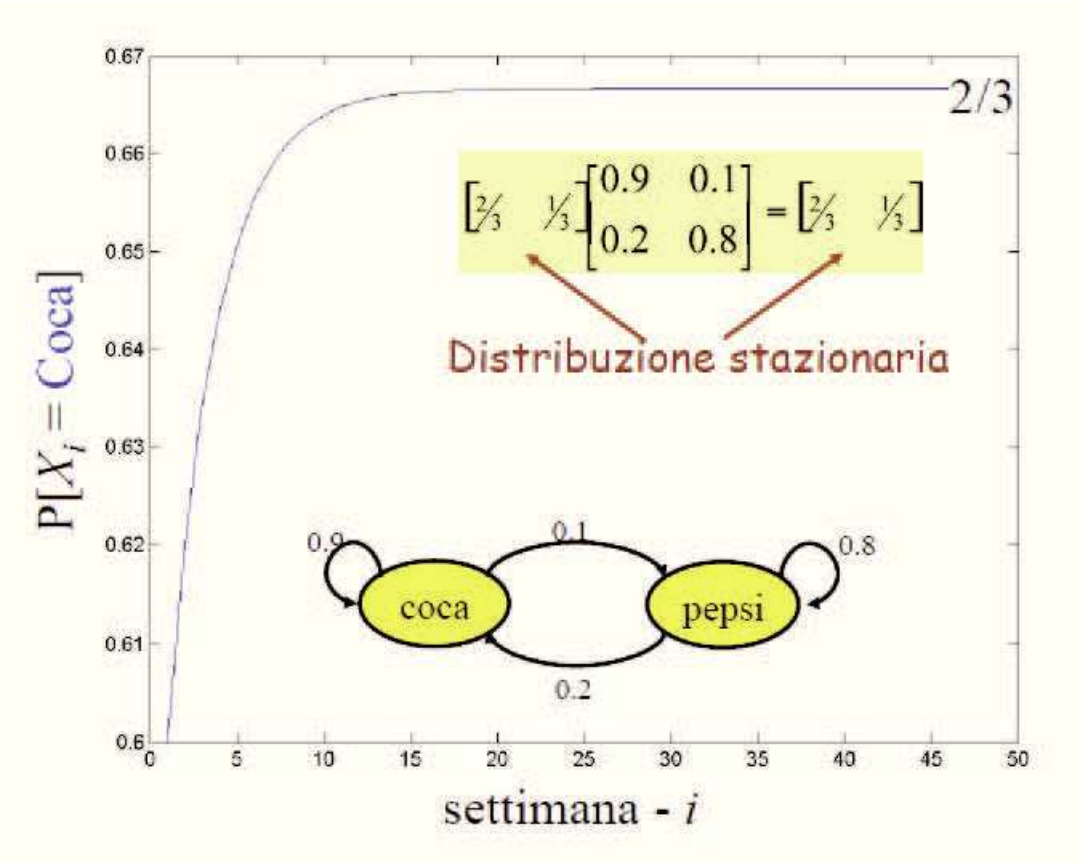


Figura 3: Distribuzione stazionaria

Formalmente, se S è il numero di stati di una catena di Markov, un vettore $\varphi = (\varphi_1, \varphi_2, ..., \varphi_S)$ è una distribuzione stazionaria se:

$$\varphi \times A = \varphi$$

Questa è anche l'equazione, con incognite date dalle componenti di φ, che permette di trovare i valori della distribuzione stazionaria a partire da una catena di Markov nota. Il viceversa, cioè costruire una catena di Markov con S eventi che converga ad una distribuzione stazionaria, è un problema difficile da risolvere.

1.3 Descrizione del Gibbs Sampling

Il *Gibbs Sampling* (cf. [1, 2]) è un metodo appartenente alla classe degli algoritmi MCMC, che permettono di costruire una catena di Markov che converge ad una distribuzione stazionaria: sono veri e propri metodi Monte Carlo, perché approssimano, tramite una serie di campionamenti, la soluzione di un problema difficile da risolvere in maniera esatta. Se la distribuzione stazionaria è rappresentata dal vettore $\varphi = (\varphi_1, \varphi_2, ..., \varphi_S)$, allora la catena di Markov da ricercare avrà S stati.

Il Gibbs Sampling, nella sua forma più generale, è un algoritmo per generare una sequenza di campioni che segua una distribuzione di probabilità congiunta di due o più variabili random.

Sia $X = \{X_1, X_2, ..., X_n\}$ un insieme di n variabili, ciascuna delle quali può assumere diversi valori. Con $P = p(X_1, X_2, ..., X_n)$ indichiamo la distribuzione di probabilità congiunta, definita come prodotto di probabilità condizionali:

$$P(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} P(X_i \,|\, X_1, ..., X_n)$$

Costruire un campione che segua una distribuzione di probabilità congiunta nota a priori è semplice. Ma in molte applicazioni P non è conosciuta e ricavarla usando l'equazione precedente, a partire dalle probabilità condizionali, non è semplice.

L'idea alla base del Gibbs Sampling è la seguente: anziché fare il sampling dalla distribuzione P, si può campionare a partire da una distribuzione di valori che progressivamente si 'avvicini' a P, man mano che generiamo nuovi campioni.

Iterando abbastanza questo processo di sampling, attraverso una serie di sostituzioni di posto tra valori, finiremo per campionare da una distribuzione

molto simile a P. Ciò vale indipendentemente dalla scelta della distribuzione iniziale.

Supponiamo allora di voler ottenere k campioni dall'insieme X. Denotiamo l'i-esimo campione con $X^{(i)} = \{x_1^{(i)}, x_2^{(i)}, ..., x_n^{(i)}\}$, dove $x_1^{(i)}, x_2^{(i)}, ..., x_n^{(i)}$ sono possibili valori assunti dalle variabili $X_1, X_2, ..., X_n$. Il Gibbs Sampling consiste nell'effettuare i seguenti passi:

1. Costruisci in maniera random un campione $X^{(0)}$;
2. Per $i = 1, 2, ..., k$ ripeti le seguenti operazioni:
   - Campiona un valore per $x_1^{(i)}$ secondo la probabilità $p(x_1^{(i)} \mid x_2^{(i-1)}, x_3^{(i-1)}, ..., x_n^{(i-1)})$;
   - Campiona un valore per $x_2^{(i)}$ secondo la probabilità $p(x_2^{(i)} \mid x_1^{(i)}, x_3^{(i-1)}, ..., x_n^{(i-1)})$;
   - ...
   - Campiona un valore per $x_j^{(i)}$ secondo la probabilità $p(x_j^{(i)} \mid x_1^{(i)}, x_2^{(i)}, ..., x_{j-1}^{(i)}, x_{j+1}^{(i-1)}, ..., x_n^{(i-1)})$;
   - ...
   - Campiona un valore per $x_n^{(i)}$ secondo la probabilità $p(x_n^{(i)} \mid x_1^{(i)}, x_2^{(i)}, ..., x_{n-1}^{(i)})$;
   - Costruisci $X^{(i)} = \{x_1^{(i)}, x_2^{(i)}, ..., x_n^{(i)}\}$;
3. Restituisci i k campioni $X^{(1)}, X^{(2)}, ..., X^{(k)}$.

Il Gibbs Sampling, quindi, campiona una variabile per volta a partire dalla probabilità condizionale di quella variabile rispetto alle altre, utilizzando sempre i valori più recenti e aggiornando la variabili col nuovo valore appena campionato.

Possiamo a questo punto caratterizzare il Gibbs Sampling come metodo MCMC, allo scopo di costruire una catena di Markov la cui distribuzione stazionaria corrisponda alla distribuzione di probabilità congiunta P e produrre alla fine dei campioni che seguano la distribuzione stazionaria.

Anzitutto, costruiamo una catena di Markov in cui ogni stato corrisponde ad un possibile assegnamento delle n variabili $X_1, X_2, ..., X_n$. La catena avrà un numero di stati pari al prodotto del numero di valori possibili di ciascuna variabile.

Eseguiamo il Gibbs Sampling con k sufficientemente grande, partendo da un assegnamento iniziale random per le n variabili, cioè da uno stato qualsiasi della catena. La costruzione di un nuovo sample $X^{(i)}$ corrisponde ad un nuovo

assegnamento e quindi ad una transizione da uno stato all'altro della catena, mentre il prodotto delle probabilità condizionali, ovvero:

$$p(x_1^{(i)} \mid x_2^{(i-1)}, x_3^{(i-1)}, ..., x_n^{(i-1)}) \times p(x_2^{(i)} \mid x_1^{(i)}, x_3^{(i-1)}, ..., x_n^{(i-1)}) \times ... \times$$
$$p(x_n^{(i)} \mid x_1^{(i)}, x_2^{(i)}, ..., x_{n-1}^{(i)})$$

corrisponde alla probabilità di transizione da uno stato all'altro.

Quindi, se costruiamo opportunamente la catena per modellare un problema, tramite il Gibbs Sampling possiamo generare alla fine uno o più campioni che rappresentano sequenze di eventi osservati con alta probabilità, e quindi una soluzione probabile al problema stesso.

Capitolo 2

Schema dell'algoritmo

L'algoritmo non deterministico proposto è un'applicazione del Gibbs Sampling come metodo MCMC e si ispira a quello proposto da C. Lawrence (cf. [4]) per l'allineamento locale di sequenze, in cui al posto di sequenze di caratteri vi sono grafi, al posto di caratteri vi sono nodi e al posto di segmenti di W caratteri a partire da un carattere a vi sono sottografi formati da W nodi, chiamate finestre, generati partendo da un nodo, chiamato seme. L'unica differenza importante è che nei grafi l'ordine tra i nodi, in generale, non è univoco, ma dipende dal cammino percorso, mentre nelle sequenze l'ordine tra i caratteri è unico. Ciò implica che non esiste, in generale, un'unica finestra di W nodi a partire da uno stesso seme.

Ogni scelta di una finestra in un grafo corrisponde alla scelta di un sottografo di W nodi. Ogni combinazione di N finestre, una per ogni grafo, corrisponde ad un possibile allineamento. Quanti sono i possibili allineamenti?

Sia G l'insieme dei grafi da allineare, G_i un grafo dell'insieme e N_i il numero di nodi. Sia n_x il numero di finestre a partire da x.

Il grafo G_i, dunque, conterrà $\sum_{j=1}^{N_i} n_j$ finestre di W nodi. Il numero di possibili combinazioni di sottografi, ovvero il numero di possibili allineamenti è:

$$S = \prod_{l=1}^{N} \sum_{j=1}^{N_i} n_j$$

L'algoritmo si basa su un processo stocastico a partire dalla costruzione di una catena di Markov con S stati. Ogni stato corrisponde ad una scelta dei sottografi negli N grafi e ad un possibile allineamento. La Fig. 4 descrive ciò che accade in una possibile transizione da uno stato all'altro della catena.

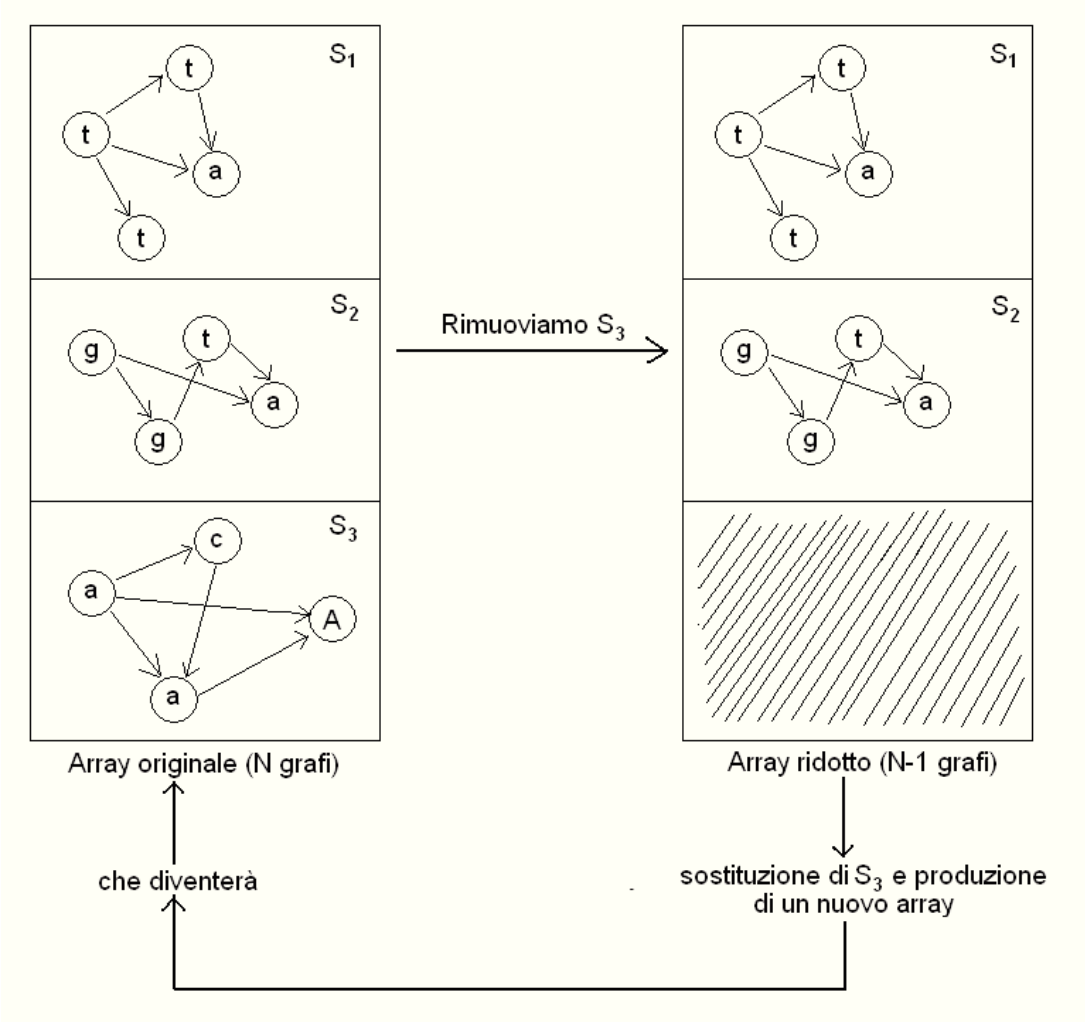


Figura 4: Transizione da uno stato all'altro nella catena

Partiamo da una scelta iniziale casuale dei sottografi, che definirà un allineamento iniziale, ovvero uno stato iniziale, e consideriamo la matrice dei sottografi scelti, formata da $N \times W$ nodi. Chiamiamo *array originale* tale matrice.

Ogni step dell'algoritmo non deterministico consiste nel passare da un allineamento ad un altro, cioè nell'effettuare una transizione da uno stato all'altro della catena. Ciò è ottenuto rimuovendo dall'array originale il sottografo relativo ad uno degli N grafi (ottenendo così una matrice di $N-1$ grafi, chiamata *array ridotto*) e rimpiazzandolo con un altro dello stesso grafo sulla base di una *probabilità di transizione* definita opportunamente a partire da un punteggio, detto *Likelihood Ratio* (LR).

La scelta del sottografo da rimuovere dall'array originale è assolutamente casuale. Ad ogni allineamento è possibile associare un punteggio, chiamato *bit score* o *quantità di informazione*, misurata in bit, che indica il grado di similarità delle finestre dell'allineamento e consente di restituire alla fine il migliore allineamento.

Ciò che segue è lo schema generale di funzionamento del LGA Gibbs Sampler:

LGA_Gibbs_Sampler(G,W)

BEGIN

- Let S be the current alignment and BestAlign the best alignment found.
- 1) Select randomly a window of size W in each graph for the initial alignment.
S := Build_Initial_Alignment();

- Let MAXITER be the maximum number of iterations.
for $indIter = 1$ to $MAXITER$ **do**

- 2) Select randomly a window to be removed from the original array.
i := Random(1,N);
$S := S$ - S_i;

- 3) Let W_i be the set of windows in G_i. Compute the probability of transition P for each window in G_i.
for $j = 0$ to $|Wi|$ **do**
$LR(S_j)$:= Compute_LR_Score(S_j);
$P(S_j)$:= Compute_Prob_Transition(S_j);
end for

- 4) Sample randomly from the distribution of probability P a window in G_i and add it to the original array.
S'_i := Sample(P);
$S := S \cup S'_i$;

- 5) Compute the bit score of the new alignment and update BestAlign if necessary.
$bitScore$:= Compute_Bit_Score(S);

```
    if bitScore > bestScore then
      bestAlign := S;
    end if
  end for
  return bestAlign;

  END
```

Dal momento che si basa su un processo stocastico, l'algoritmo può restituire allineamenti differenti in diverse esecuzioni.

Capitolo 3

Passi preliminari

Per poter eseguire l'algoritmo occorre costruire l'insieme delle finestre dei vari grafi. La seguente procedura descrive come ottenere in un grafo G tutte le finestre di dimensione W a partire da un nodo a:

BuildWindows (G, W, a)

BEGIN

- Let Wind be the set of all windows of size W from seed a.
$Wind := \varnothing$;
- Build the BFS tree from node a.
$BFSTree :=$ BFS(G, a);
if $|BFSTree| \geq W$ **then**
  - Let C be a queue of window, initially empty.
  C.insertTail($\{a\}$);
  while C is not empty **do**
    $currWindow := C$.deleteHead();
    if $|currWindow| \neq W$ **then**
      - 1) Let n be the rightmost deepest node of currWindow in the BFSTree. Extend the current window, adding a brother or a cousin of n in the BFSTree to the right of n (if possible), and then add the new window to the queue.
      for all $x \in$ Brother(n) $\vee$ ($x \in$ Cousin(n) : Father(x) $\in currWindow$) **do**
        C.insertTail($currWindow \cup \{x\}$);
      end for

- 2) Let DeepestNode be the set of the deepest nodes of currWindow in the BFSTree. Extend the current window, adding a child of a deepest node in the BFSTree (if possible), and then add it to the queue.
for all $n \in DeepestNode$ **do**
for all $x \in$ Child(n) **do**
C.insertTail$(currWindow \cup \{x\})$;
end for
end for
else
- The current window has size W, then it can be added to Wind.
$Wind := Wind \cup currWindow$;
end if
end while
end if
return $Wind$;

END

L'algoritmo è incrementale, cioè produce prima tutte le finestre di dimensione 2, poi tutte quelle di dimensione 3, ecc... fino ad arrivare a W. Quindi, tutte le finestre di dimensione W sono ottenute a partire da finestre di dimensione $W-1$, aggiungendo opportunamente un nuovo nodo.

Per fare ciò si parte dall'albero BFS costruito a partire dal nodo a. Esso conterrà tutti i possibili candidati a far parte di una finestra a partire da a, che devono essere nodi raggiungibili da esso. Quindi, se l'albero BFS contiene meno di W nodi, ciò significa che non esistono finestre di dimensione W dal nodo a e l'algoritmo restituisce un insieme vuoto.

In caso contrario, supponiamo di avere già calcolato una finestra F di $W-1$ nodi.

La Fig. 5 mostra un esempio di albero BFS a partire da a in cui i nodi già aggiunti a F sono cerchiati in blu. Quali sono i nodi candidati ad essere aggiunti in F in modo da ottenere una nuova finestra di dimensione W?

Sia x il nodo della finestra più profondo e più a destra nell'albero BFS. I nodi candidati (cerchiati in rosso) vanno cercati o 'al di sotto' di x, tra tutti i figli dei nodi di F più profondi nell'albero, o 'a destra' di x, cioè tra i fratelli o 'cugini', il cui padre è già presente in F, che stanno a destra rispetto a x nell'albero. Questo 'orientamento' verso destra nella costruzione delle finestre permette di generare in maniera esaustiva e senza ripetizioni tutte le finestre a partire dal nodo a.

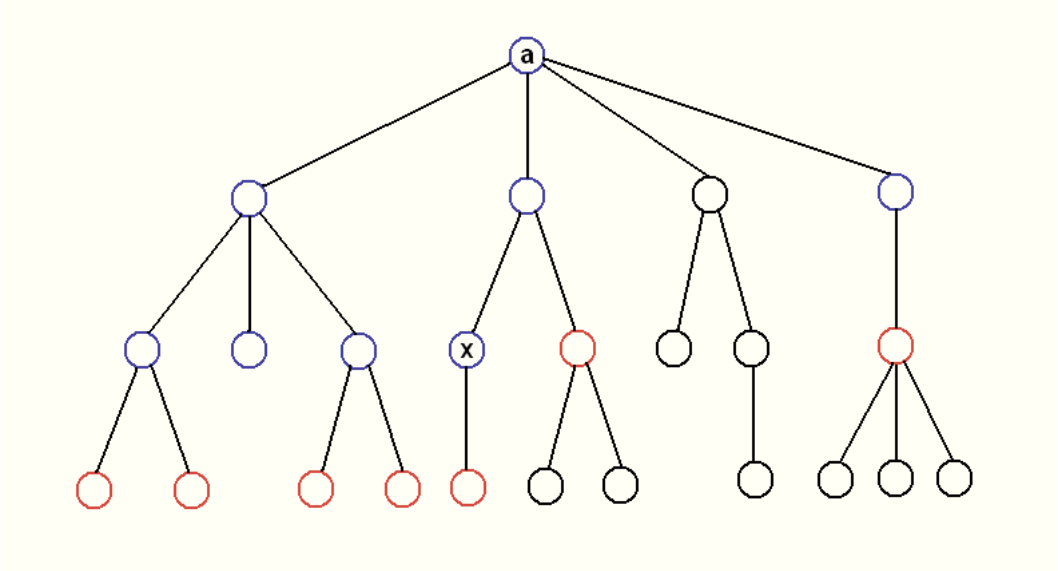


Figura 5: Albero BFS a partire da a

Iterando questo algoritmo su tutti i nodi di G possiamo ottenere tutte le finestre del grafo.

Capitolo 4

Calcolo del LR e della probabilità di transizione

Il *Likelihood Ratio* (LR) di una finestra F è un indice della similarità del sottografo rappresentato da F con gli altri sottografi dell'array ridotto e serve a calcolare la probabilità di transizione per F. Più elevato è il LR, maggiore è la probabilità che la finestra venga selezionata e inserita nell'allineamento.

La similarità viene valutata sulla base di:

a) corrispondenza tra le etichette dei nodi dei sottografi;

b) corrispondenza tra le topologie dei sottografi dell'allineamento.

Le due valutazioni vengono effettutate separatamente, cioè si calcolano due punteggi diversi che moltiplicati danno il LR della finestra.

Per valutare la corrispondenza a) si parte innanzitutto dalla costruzione di una mappa M_n di sequenze di nodi, una per ogni sottografo dell'array ridotto. Ogni sequenza segue l'ordine di visita dei nodi in una BFS a partire dal seme della finestra.

Supponiamo ad esempio che l'array ridotto contenga i tre sottografi, S_1, S_2 e S_3, rappresentati in Fig. 6.

Allora la mappa M_n potrebbe essere formata da:

NCCCHNCC

NCCNCCCH

NCCCNCCC

In generale, la mappa M_n non è unica, ma dipende dall'ordine di visita degli adiacenti di un nodo.

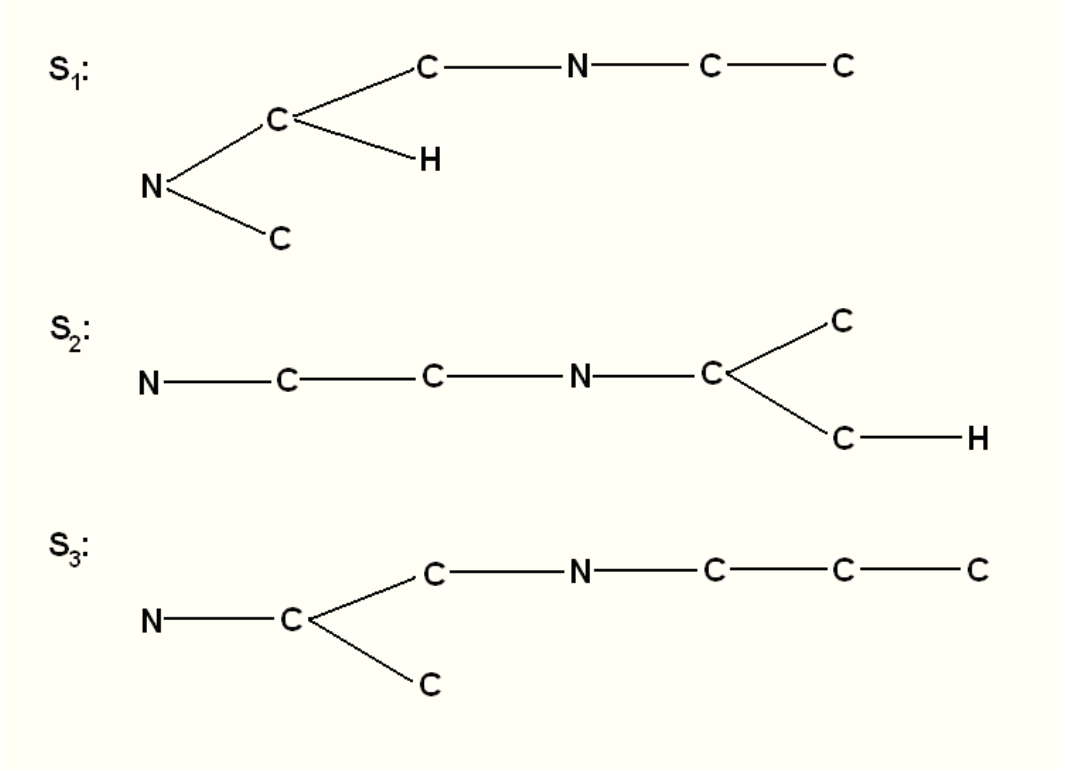


Figura 6: Sottografi dell'array ridotto

A questo punto si considera la sequenza dei nodi di F, ordinata anch'essa in base ai tempi di visita in una BFS a partire dal seme, e si calcola per ogni nodo x_i (in posizione i nella sequenza) la frequenza di x_i alla posizione i nella mappa M_n. Lo score della corrispondenza a) è dato dal prodotto delle frequenze.

Ad esempio, sia F la finestra rappresentata in Fig. 7.

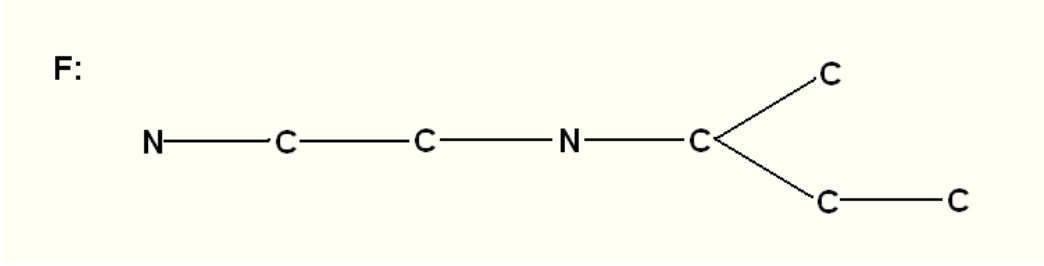


Figura 7: Finestra F candidata

La sequenza ordinata dei nodi di F potrebbe essere:

NCCNCCCC

Lo score della corrispondenza a) rispetto alla mappa M_n precedente è:

$$ScoreA = 1 \times 1 \times 1 \times 0.33 \times 0.33 \times 0.66 \times 1 \times 0.66$$

Per valutare la corrispondenza b) si procede in maniera simile. Si costruisce anzitutto una mappa M_g di sequenze di gradi di nodi, una per ogni sottografo dell'array ridotto. Come in precedenza, ogni sequenza segue l'ordine di visita dei nodi in una BFS a partire dal seme delle finestre.

Successivamente si considera la sequenza dei nodi di F, ordinata secondo i tempi di visita di una BFS a partire dal seme, e si calcola per ogni nodo x_i la frequenza di x_i alla posizione i in M_g. Lo score della corrispondenza b) è dato dal prodotto delle frequenze.

In riferimento all'esempio precedente, M_g è dato da:

23121221

12223121

13212221

La sequenza ordinata dei nodi di F è:

12223121

Lo score della corrispondenza b) rispetto a M_g è:

$$ScoreB = 0.66 \times 0.33 \times 0.66 \times 0.66 \times 0.33 \times 0.33 \times 1 \times 1$$

Il LR della finestra è dato dal prodotto di $ScoreA$ con $ScoreB$.

Se $WindW(G)$ è l'insieme di tutte le finestre di G, la probabilità di transizione di F è:

$$P(F) = \frac{LR(F)}{\sum_{F_i \in WindW(G)} LR(F_i)}$$

Capitolo 5

Calcolo del Bit Score

Il calcolo del *Bit Score* consente di valutare la qualità dell'allineamento ed è una sorta di misura del 'grado di ordine' presente: infatti è duale rispetto all'entropia, cioè è minima quando l'entropia è massima e massima quando l'entropia è nulla.

Il Bit Score consente anche di fissare un altro criterio di terminazione dell'algoritmo basato sulla definizione di un valore soglia: quando il punteggio supera la soglia, anche se non si è raggiunto il numero massimo di iterazioni, l'algoritmo può terminare, restituendo il migliore allineamento trovato.

Per calcolare il Bit Score estendiamo anzitutto le mappe M_n e M_g definite nel paragrafo precedente con l'aggiunta, rispettivamente, delle sequenze dei nodi e dei gradi della finestra appena inserita nell'allineamento dopo il sampling, in modo da rappresentare l'allineamento corrente. Si ottengono così due matrici, ciascuna di dimensione $N \times W$, di simboli appartenenti all'*alfabeto di input* $\sum$, che comprende tutte le possibili etichette di nodi e i numeri 1, 2, ..., $W-1$, che rappresentano tutti i possibili gradi dei nodi delle finestre.

Il Bit Score finale è ottenuto come somma di due Bit Score, relativi, rispettivamente, a M_n, cioè all'allineamento delle etichette, e a M_g, cioè all'allineamento dei gradi.

Consideriamo una delle due mappe, ad esempio M_n. Indichiamo con C_i, con $1 \leq i \leq W$, una delle colonne. Essa rappresenta una sequenza di simboli sull'alfabeto $\sum$. Definiamo *entropia* di C_i la quantità:

$$H(C_i) = \sum_{a \in \sum} \left(P_a \times \log_2 \frac{1}{P_a} \right)$$

dove P_a è la frequenza del simbolo a nella colonna C_i.

La *quantità di informazione* di C_i è definita come:

$$Q(C_i) \;=\; \log_2 \left|\sum\right| - H(C_i)$$

$Q(C_i)$ è duale rispetto a $H(C_i)$: maggiore è la similarità tra gli elementi della colonna, maggiore è $Q(C_i)$ (e minore è l'entropia). $Q(C_i)$ è massimo (e vale $\log_2 |\sum|$) quando gli elementi della colonna sono tutti uguali.

Il bit score relativo a M_n è la somma delle quantità di informazione delle colonne della mappa, ovvero:

$$BitScore(M_n) \;=\; \sum_{i=1}^{W} Q(C_i)$$

Anologo ragionamento si può fare con le colonne della mappa M_g. Segue che il bit score relativo a M_g è:

$$BitScore(M_g) \;=\; \sum_{i=1}^{W} Q(C_i)$$

dove C_i in questo caso rappresenta una generica colonna di M_g. Il Bit Score finale di un allineamento A è, dunque:

$$BitScore(A) \;=\; BitScore(M_n) + BitScore(M_g)$$

In base alle definizioni dei punteggi, il massimo valore che il Bit Score può assumere è:

$$maxScore \;=\; 2 \times W \times \log_2 \left|\sum\right|$$

e ciò avviene quando tutte le sequenze in M_n e tutte le sequenze in M_g sono tra loro uguali.

Il punteggio massimo non dipende, quindi, dal numero N di grafi da allineare.

Capitolo 6

Test su proteine

Per verificare la correttezza e l'efficacia dell'algoritmo, sono stati effettuati una serie di test su un insieme di 30 proteine rappresentate nella loro struttura chimica, cioè come reti in cui i nodi sono gli atomi presenti e gli archi i legami chimici tra gli atomi stessi. L'algoritmo è in grado di individuare, in un numero di iterazioni limitato, strutture molecolari, più o meno grandi, comuni a tutte le proteine.

L'alfabeto di input $\sum$, formato dalle tutte le possibili etichette e da tutti i possibili gradi dei nodi è:

$$\sum = \{C, O, N, H, P, S, Cl, F, Ca, Mg, Mn, Zn, Se, Na, 0, 1, 2, ..., W-1\}$$

Per i test da effettuare, sono stati considerati due dataset differenti, sulla base del numero di finestre di dimensione fissata di una rete:

- *Small Dataset*, formato dalle 24 reti col minor numero di finestre;
- *Large Dataset*, costituito da tutte e 30 le reti.

La Tab. 1 elenca tutte le network utilizzate: d'ora in poi, per semplicità, nei risultati dei test si farà sempre riferimento ad una proteina tramite il suo ID. Le prime 24 reti costituiscono lo Small Dataset. Nella tabella sono indicati anche il numero di nodi, di archi e di finestre di dimensione 8 e 12.

Sono state effettuate tre batterie da 25 test ciascuna per verificare il comportamento dell'algoritmo al variare del numero N di reti usate e della dimensione W della finestra, con parametri:

1. $W = 8$ e $N = 24$;
2. $W = 8$ e $N = 30$;
3. $W = 12$ e $N = 24$;

I test sono stati realizzati su una macchina con processore Intel Core i3-2120 da 3.30 Ghz e memoria RAM da 4 GB.

ID	**NOME**	**Nodi**	**Archi**	**W=8**	**W=12**
HYD_OG	Hydrolase(O-Glycosyl)	2594	1291	13413	85654
KIN	Kinase	8180	4972	257221	5458997
MCD	Mast_Cell_Degranulation	240	240	17076	354540
RIN	Ribonuclease_Inhibitor	2868	1464	15702	102526
HYD1	Hydrolase_1	4408	2000	21540	132525
ALL	Allergen	1988	2005	142704	3193209
HYD2	Hydrolase_2	1907	1942	131605	2778320
TOX	Toxin	411	415	28022	592501
TRR	Transcription_Regulation	1213	1215	84591	1874236
IMM	Immunoreceptor	2103	2127	146583	3160173
ANT	Antibiotic	259	253	16559	351144
ANTC	Antibiotic_Chromoprotein	1487	1498	96634	1985262
CPO	Complex_(Proto-Oncogene /Early_Protein)	1922	1940	131064	2900273
COM	Complement	775	784	49822	1073251
MBP	Metal_Binding_Protein	1197	1212	86013	1931378
PC	Potassium_Channel	466	468	31683	694039
GF	Growth_Factor	2446	2471	167630	3691367
PHO	Phosphorylation	1818	1833	128086	2872206
DNP	De_Novo_Protein	1658	653	6589	43811
TD1	Transferase/DNA_1	6311	3633	60548	581225
HYD_CP	Hydrolase(C-Terminal_Peptidase)	5505	2675	41845	503603
TD2	Transferase/DNA_2	6142	3392	78981	1182948
TD3	Transferase/DNA_3	6184	3455	78667	1135810
TD4	Transferase/DNA_4	6227	3497	80230	1171012
OXI	Oxidoreductase	15593	12010	748475	16133495
REP	Replication	10392	7699	403046	8709456
CAD	Cell_Adhesion	33067	30773	1955778	42942064
ISO	Isomerase	11147	8459	456271	9689123
TRA	Transferase	20030	17263	1029488	21828651
OXI_C	Oxidoreductase (CHOH(D)-NAD+(A))	11010	8280	441082	9422884

Tabella 1: Reti di proteine utilizzate nei test

6.1 Parametri e output dell'algoritmo

L'algoritmo prende in input quattro parametri:

1. N, numero di grafi;
2. W, dimensione della finestra dell'allineamento;
3. MAX_ITER, numero massimo di iterazioni;
4. MAX_SCORE, punteggio massimo che si vuole raggiungere con l'allineamento.

Per questo tipo di test, l'ultimo parametro è sempre impostato al valore di $maxScore$ (vedere Cap.5), che rappresenta il punteggio raggiunto in caso di corrispondenza perfetta. In queste reti, infatti, esistono occorrenze di strutture molecolari presenti in tutti i grafi.

Un ottimo pattern comune, corrispondente ad un grafo con la stessa struttura e con uno o al più due nodi cambiati, può essere ottenuto anche impostando MAX_SCORE ad un valore (per questo tipo di test) pari al 98-99% del punteggio massimo di un allineamento, chiamato soglia di ottimalità, che indichiamo con σ.

Poiché l'algoritmo non sempre converge verso un pattern esatto, impostando la soglia di ottimalità, che di solito è raggiunta in un numero di iterazioni nettamente inferiore a 1000, si può ottenere una soluzione buona in tempi generalmente più brevi. A seconda che le reti da allineare siano più o meno simili, la soglia può essere abbassata o alzata.

Di seguito è riportato un esempio di output prodotto dall'algoritmo:

```
NUM_ITERATIONS: 1000 MAX_SCORE: 71.35090589819676
BEST ALIGNMENT:
Iteration: 695, Score: 70.92467724119861

HYDROLASE(O-GLYCOSYL):
Nodes = [(218, C),(219, C),(221, C),(224, N),(223, C),(225, C),(226, C),(232, N)]
Edges = {(218,219) (218,221) (219,224) (221,223) (224,225) (225,226) (226,232)}
KINASE:
Nodes = [(1425, C),(1426, C),(1428, C),(1432, N),(1429, C),(1433, C),(1434, C),(1441, N)]
Edges = {(1425,1426) (1425,1428) (1426,1432) (1428,1429) (1432,1433) (1433,1434) (1434,1441)}
MAST_CELL_DEGRANULATION:
```

```
Nodes = [(36, C),(37, C),(39, C),(59, N),(40, C),(60, C),(61, C),(81, N)]
Edges = {(36,37) (36,39) (37,59) (39,40) (59,60) (60,61) (61,81)}
RIBONUCLEASE_INHIBITOR:
Nodes = [(904, C),(905, C),(907, C),(911, N),(908, C),(912, C),(913, C),(916, N)]
Edges = {(904,905) (904,907) (905,911) (907,908) (911,912) (912,913) (913,916)}
HYDROLASE1:
Nodes = [(1576, C),(1577, C),(1579, C),(1582, N),(1581, C),(1583, C),(1584, C),(1587, N)]
Edges = {(1576,1577) (1576,1579) (1577,1582) (1579,1581) (1582,1583) (1583,1584) (1584,1587)}
ALLERGEN:
Nodes = [(553, C),(554, C),(556, C),(571, N),(563, H),(572, C),(573, C),(586, N)]
Edges = {(553,554) (553,556) (554,571) (556,563) (571,572) (572,573) (573,586)}
HYDROLASE2:
Nodes = [(570, C),(571, C),(573, C),(588, N),(575, C),(589, C),(590, C),(598, N)]
Edges = {(570,571) (570,573) (571,588) (573,575) (588,589) (589,590) (590,598)}
TOXIN:
Nodes = [(158, C),(159, C),(161, C),(174, N),(162, C),(175, C),(176, C),(196, N)]
Edges = {(158,159) (158,161) (159,174) (161,162) (174,175) (175,176) (176,196)}
TRANSCRIPTION_REGULATION:
Nodes = [(904, C),(905, C),(907, C),(922, N),(908, C),(923, C),(924, C),(939, N)]
Edges = {(904,905) (904,907) (905,922) (907,908) (922,923) (923,924) (924,939)}
IMMUNORECEPTOR:
Nodes = [(288, C),(289, C),(291, C),(303, N),(292, C),(304, C),(305, C),(313, N)]
Edges = {(288,289) (288,291) (289,303) (291,292) (303,304) (304,305) (305,313)}
ANTIBIOTIC:
Nodes = [(174, C),(175, C),(177, C),(185, N),(178, C),(186, C),(187, C),(201, N)]
Edges = {(174,175) (174,177) (175,185) (177,178) (185,186) (186,187) (187,201)}
ANTIBIOTIC_CHROMOPROTEIN:
Nodes = [(588, C),(589, C),(591, C),(601, N),(592, C),(602, C),(603, C),(617, N)]
Edges = {(588,589) (588,591) (589,601) (591,592) (601,602) (602,603) (603,617)}
COMPLEX_(PROTO-ONCOGENE/EARLY_PROTEIN):
Nodes = [(383, C),(384, C),(386, C),(402, N),(396, H),(403, C),(404, C),(409, N)]
Edges = {(383,384) (383,386) (384,402) (386,396) (402,403) (403,404) (404,409)}
COMPLEMENT:
Nodes = [(641, C),(642, C),(644, C),(655, N),(645, C),(656, C),(657, C),(667, N)]
Edges = {(641,642) (641,644) (642,655) (644,645) (655,656) (656,657) (657,667)}
METAL_BINDING_PROTEIN:
Nodes = [(615, C),(616, C),(618, C),(633, N),(619, C),(634, C),(635, C),(652, N)]
Edges = {(615,616) (615,618) (616,633) (618,619) (633,634) (634,635) (635,652)}
POTASSIUM_CHANNEL:
Nodes = [(246, C),(247, C),(249, C),(259, N),(250, C),(260, C),(261, C),(281, N)]
Edges = {(246,247) (246,249) (247,259) (249,250) (259,260) (260,261) (261,281)}
```

```
GROWTH_FACTOR:
Nodes = [(133, C),(134, C),(136, C),(146, N),(137, C),(147, C),(148, C),(156, N)]
Edges = {(133,134) (133,136) (134,146) (136,137) (146,147) (147,148) (148,156)}
PHOSPHORYLATION:
Nodes = [(1611, C),(1612, C),(1614, C),(1629, N),(1615, C),(1630, C),(1631, C),(1640, N)]
Edges = {(1611,1612) (1611,1614) (1612,1629) (1614,1615) (1629,1630) (1630,1631) (1631,1640)}
DE_NOVO_PROTEIN:
Nodes = [(108, C),(109, C),(111, C),(115, N),(112, C),(116, C),(117, C),(127, N)]
Edges = {(108,109) (108,111) (109,115) (111,112) (115,116) (116,117) (117,127)}
TRANSFERASE_DNA1:
Nodes = [(3088, C),(3089, C),(3091, C),(3098, N),(3092, C),(3099, C),(3100, C),(3110, N)]
Edges = {(3088,3089) (3088,3091) (3089,3098) (3091,3092) (3098,3099) (3099,3100) (3100,3110)}
HYDROLASE(C-TERMINAL_PEPTIDASE):
Nodes = [(872, C),(873, C),(875, C),(880, N),(876, C),(881, C),(882, C),(888, N)]
Edges = {(872,873) (872,875) (873,880) (875,876) (880,881) (881,882) (882,888)}
TRANSFERASE_DNA2:
Nodes = [(549, C),(550, C),(552, C),(557, N),(553, C),(558, C),(559, C),(569, N)]
Edges = {(549,550) (549,552) (550,557) (552,553) (557,558) (558,559) (559,569)}
TRANSFERASE_DNA3:
Nodes = [(970, C),(971, C),(973, C),(977, N),(974, C),(978, C),(979, C),(985, N)]
Edges = {(970,971) (970,973) (971,977) (973,974) (977,978) (978,979) (979,985)}
TRANSFERASE_DNA4:
Nodes = [(2857, C),(2858, C),(2860, C),(2865, N),(2861, C),(2866, C),(2867, C),(2872, N)]
Edges = {(2857,2858) (2857,2860) (2858,2865) (2860,2861) (2865,2866) (2866,2867) (2867,2872)}

Time elapsed: 3.9736 min
```

In ciascun grafo, ogni nodo è rappresentato come una coppia formata da ID e etichetta e ogni arco come coppia di ID sorgente e ID destinazione.

6.2 Comportamento dell'algoritmo

Le Fig. 8, 9, 10 e 11 mostrano la variazione di punteggio all'aumentare del numero di iterazioni in quattro scenari differenti:

a) Caso migliore: allineamento perfetto raggiunto entro il numero massimo di iterazioni, cioè 1000;

b) Caso medio: convergenza verso un pattern comune (senza corrispondenza esatta tra le etichette dei nodi). In questo caso, solitamente, la soglia di ottimalità è raggiunta prima delle 1000 iterazioni;

c) Caso peggiore (raro): nessuna convergenza ad un pattern comune. Può accadere che:

   1) Emergano due sottostrutture, una comune ad un subset di reti e l'altra comune al resto del dataset;

   2) Emerga un pattern comune a quasi tutti i grafi. In questo caso si è in presenza di uno o più grafi 'outliers';

   In entrambi i casi, si determina un allineamento ottimo per un sottoinsieme più o meno grande di reti e la soglia σ difficilmente viene raggiunta.

In ciascun grafico è riportato il punteggio massimo che si può ottenere in caso di match perfetto (indicato da una linea rossa) e il pattern o i pattern ottenuti alla fine.

Dall'analisi dei grafici si deduce che l'algoritmo, come il Gibbs Sampling sulle sequenze, ha un andamento a 'scala' con rapidi e improvvisi incrementi di punteggio seguiti da fasi di stabilità.

In particolare, si osserva come, all'aumentare del numero di iterazioni, gli incrementi diventano sempre più piccoli e più infrequenti, mentre le fasi di stabilità nel punteggio diventano sempre più lunghe. Ciò non è inaspettato e deriva dal fatto che man mano che si trovano buoni allineamenti, diventa sempre più difficile migliorare lo score ottenuto e trovare un allineamento migliore. La soglia di ottimalità, da un punto di vista grafico, corrisponde più o meno al punteggio a partire dal quale non si registrano più sensibili incrementi di punteggi.

Si osserva anche che, ad eccezione dello scenario c), il punteggio non subisce mai sensibili diminuzioni (solo alcuni piccoli decrementi, subito compensati da successivi incrementi).

Ciò significa che l'algoritmo non fa mai backtracking, nel senso che una volta che è 'instradato' su una possibile soluzione ottima (che può non rivelarsi tale negli scenari b) oppure c)) non torna indietro, il che implicherebbe sensibili diminuzioni di punteggio durante l'esecuzione.

Questo comportamento dell'algoritmo è indipendente dal tipo e dal numero di reti utilizzate e dalla dimensione della finestra. Nello scenario c), e in particolare nel caso 2), a causa della presenza di 'outliers' che creano 'rumore', l'algoritmo può subire delle oscillazioni più o meno sensibili nel punteggio, ma ciò non impedisce comunque di raggiungere soluzioni subottimali (in presenza di due pattern distinti) o quasi ottimali (in presenza di 'outliers'), come evidenziato dai risultati dei test presentati successivamente.

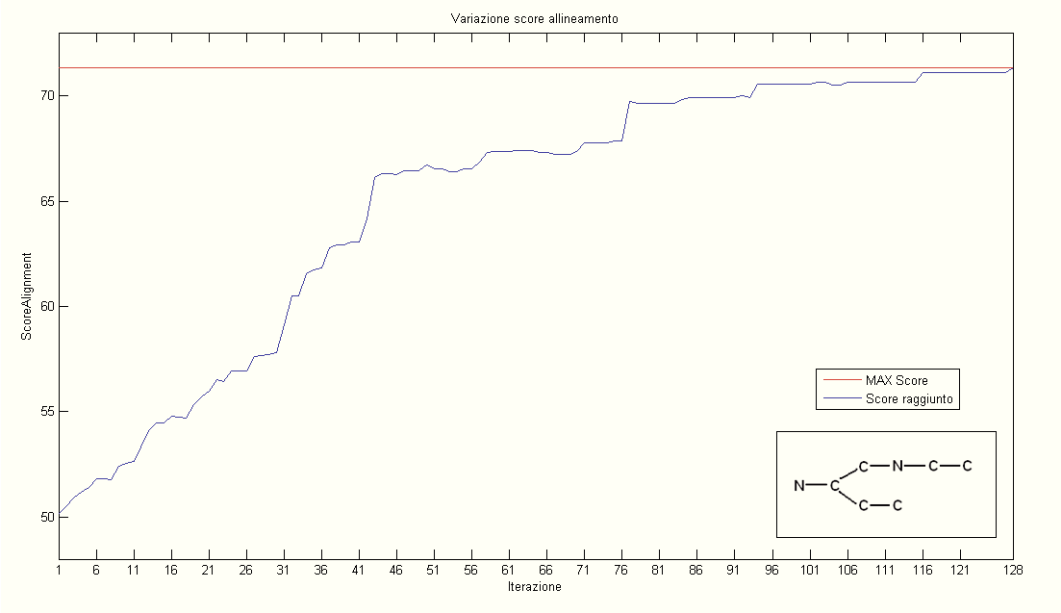


Figura 8: Comportamento nello scenario a)

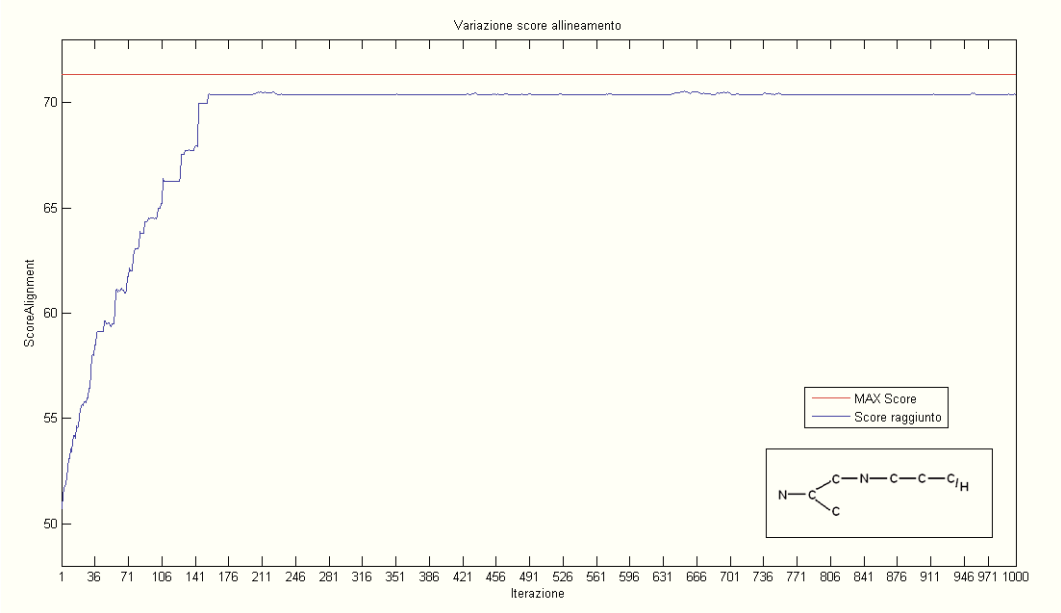


Figura 9: Comportamento nello scenario b)

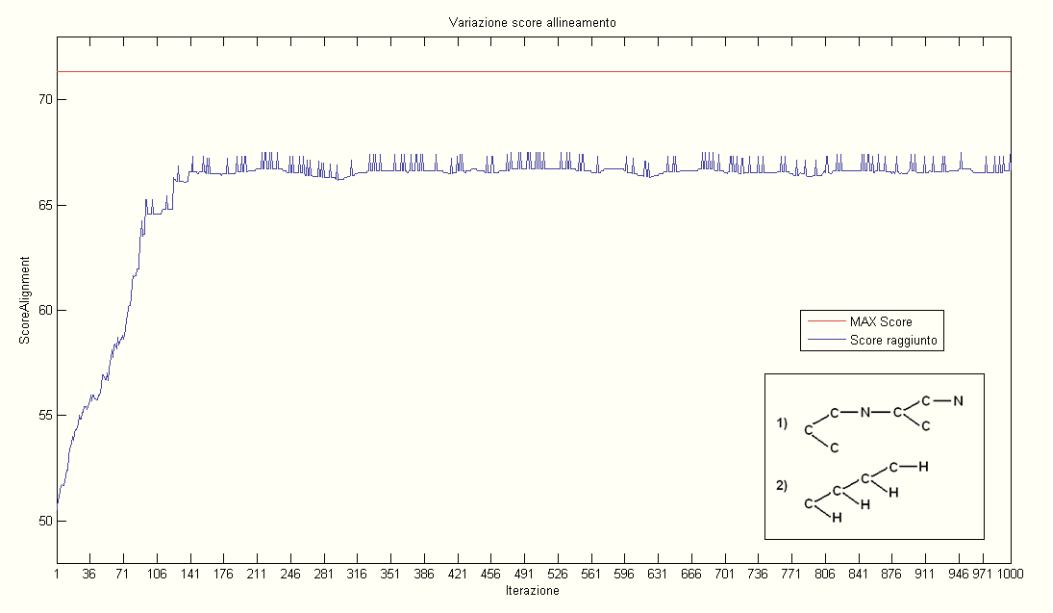


Figura 10: Comportamento nel caso 1) dello scenario c)

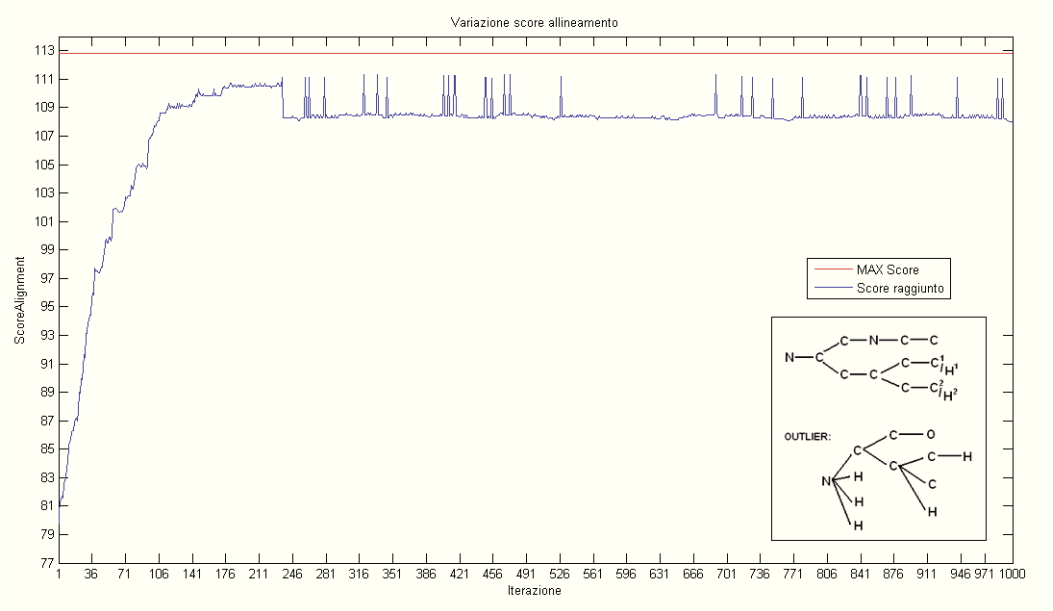


Figura 11: Comportamento nel caso 2) dello scenario c)

6.3 Risultati della prima batteria di test

La Tab. 2 riassume i risultati ottenuti nella prima batteria di 25 test, in cui:

a) $W = 8$;

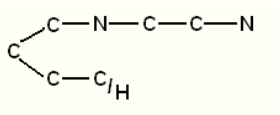
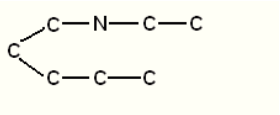
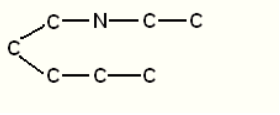
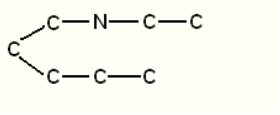
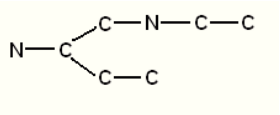
b) $N = 24$;

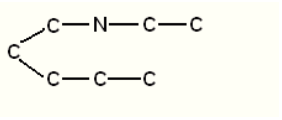
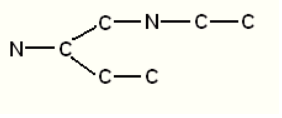
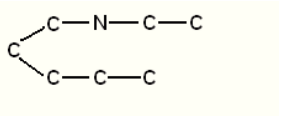
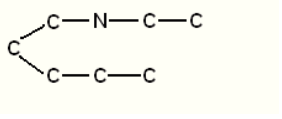
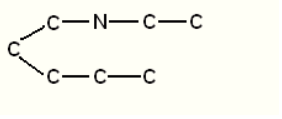
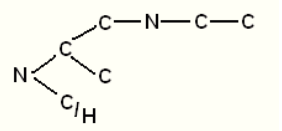
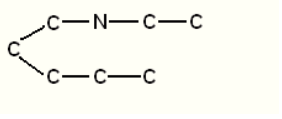
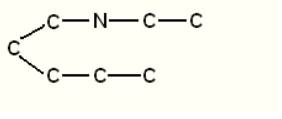
c) $MAX\_ITER = 1000$;

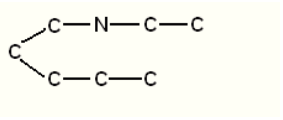
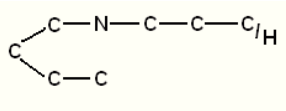
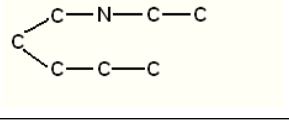
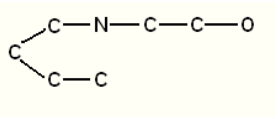
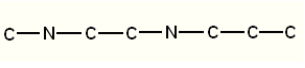
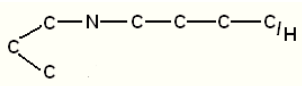
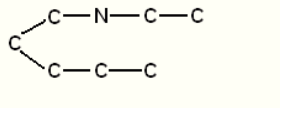
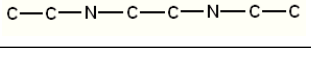
d) $MAX\_SC = maxScore = 2 \times W \times \log_2 |\sum| = 71.35090589819676$;

e) $\sigma = 98.5\%$ di $maxScore \simeq 70.28$

I tempi di esecuzione vanno da 30 secondi a circa 4 minuti.

TEST	PATTERN	ITER	SCORE	σ-ITER	TEMPO
1	 H: 1 grafo (CPO); C: 23 grafi	695	70.9246	477 (70.4643)	$\sim$ 3.97 min
2		194	MAX_SC	135 (70.6843)	$\sim$ 0.71 min
3		192	MAX_SC	146 (70.7922)	$\sim$ 0.73 min
4		147	MAX_SC	134 (70.4086)	$\sim$ 0.62 min
5		128	MAX_SC	94 (70.5342)	$\sim$ 0.49 min

6		235	MAX_SC	126 (71.0928)	$\sim$ 0.88 min
7		105	MAX_SC	62 (70.7922)	$\sim$ 0.41 min
8		488	MAX_SC	116 (70.4263)	$\sim$ 1.80 min
9		173	MAX_SC	103 (70.6666)	$\sim$ 0.63 min
10		196	MAX_SC	138 (70.5768)	$\sim$ 0.65 min
11	 H: 20 grafi; C: 4 grafi (HYD_OG, RIN, HYD1, DNP)	711	70.7922	566 (70.6843)	$\sim$ 3.68 min
12		366	MAX_SC	144 (70.3522)	$\sim$ 1.43 min
13		150	MAX_SC	113 (70.4086)	$\sim$ 0.57 min

14		256	MAX_SC	113 (70.4263)	$\sim$ 1.06 min
15	 H: 4 grafi (MCD, ANTC, COM, GF); C: 20 grafi	872	70.7922	693 (70.4187)	$\sim$ 3.76 min
16		137	MAX_SC	101 (70.4086)	$\sim$ 0.55 min
17		420	MAX_SC	146 (70.5342)	$\sim$ 1.56 min
18		542	MAX_SC	123 (70.3532)	$\sim$ 2.10 min
19	 H: 2 grafi (ALL, TRR); C: 22 grafi	651	70.9246	188 (70.3632)	$\sim$ 3.61 min
20		245	MAX_SC	122 (70.5342)	$\sim$ 0.84 min
21		189	MAX_SC	158 (70.6666)	$\sim$ 0.69 min

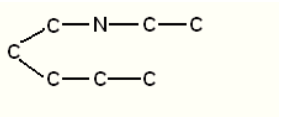
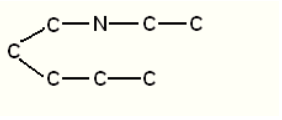
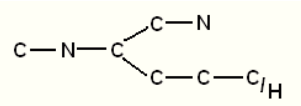
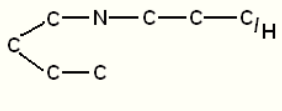
22		499	MAX_SC	104 (70.3522)	$\sim$ 1.90 min
23		219	MAX_SC	131 (70.3852)	$\sim$ 0.85 min
24	 H: 2 grafi (HYD2, ANT); C: 22 grafi	294	70.9246	177 (70.4187)	$\sim$ 3.74 min
25	 H: 6 grafi (MCD, IMM, CPO, COM, PC, GF); C: 18 grafi	847	70.5955	721 (70.3522)	$\sim$ 3.89 min

Tabella 2: Risultati della prima batteria

La tabella elenca, per ogni test, il pattern trovato (la notazione C/H indica la presenza di un nodo etichettato C oppure H nel sottografo), in quale iterazione è stato trovato, lo score dell'allineamento, l'iterazione in cui è stata superata per la prima volta la soglia di ottimalità σ e il tempo di esecuzione dell'algoritmo.

Dall'analisi della tabella possiamo osservare che:

- In 19 test su 25 è stata trovata una corrispondenza esatta (scenario a));
- In 6 test su 25 sono stati trovati pattern che differiscono tra loro su un solo nodo (scenario b));
- In tutti e 25 gli allineamenti trovati, la struttura dei pattern è unica;
- Ad eccezione dei test 1, 11, 15 e 25, la soglia di ottimalità viene raggiunta dopo meno di 200 iterazioni, anche se alla fine l'algoritmo non

converge ad un allineamento perfetto. In tutti i casi, comunque, σ è raggiunta ampiamente entro le 1000 iterazioni;

- Alcuni pattern sono stati trovati più volte in test diversi. Ad esempio, il sottografo in Fig. 12 occorre ben 14 volte.

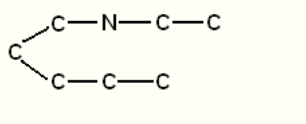


Figura 12: Pattern ripetuto nella prima batteria

Queste ripetizioni, in realtà, rappresentano diverse occorrenze dello stesso sottografo, localizzate in punti diversi di una rete. Nella Tab. 3, ad esempio, è indicata, per ogni test, la sequenza degli ID dei nodi dell'allineamento trovato nella proteina *Hydrolase(O-Glycosyl)*. La tabella mostra che i 25 allineamenti individuati, in realtà, sono da considerare tutti diversi tra loro, anche se alcune finestre si sovrappongono (ad esempio nei test 12 e 23 o nei test 17 e 20).

NUM_TEST	FINESTRA IN HYD_OG
1	[218, 219, 221, 223, 224, 225, 226, 232]
2	[785, 786, 788, 789, 795, 790, 796, 799]
3	[853, 854, 856, 863, 857, 864, 858, 865]
4	[267, 268, 270, 274, 271, 275, 273, 278]
5	[417, 418, 419, 421, 424, 423, 425, 426]
6	[947, 948, 950, 957, 951, 958, 952, 961]
7	[71, 72, 73, 75, 79, 77, 80, 81]
8	[497, 498, 500, 507, 501, 508, 502, 511]
9	[259, 260, 262, 266, 263, 267, 264, 268]
10	[686, 687, 689, 693, 690, 694, 691, 697]
11	[645, 646, 651, 647, 649, 652, 653, 656]
12	[629, 630, 632, 636, 633, 637, 635, 640]
13	[72, 73, 75, 79, 76, 80, 78, 83]
14	[1143, 1144, 1146, 1153, 1147, 1154, 1148, 1155]
15	[533, 534, 536, 539, 538, 540, 543, 544]
16	[489, 490, 492, 496, 493, 497, 494, 500]
17	[893, 894, 896, 903, 897, 904, 905, 906]
18	[46, 52, 53, 54, 60, 61, 64, 65]
19	[707, 708, 710, 713, 714, 717, 718, 719]

20	[893, 894, 896, 903, 897, 904, 898, 907]
21	[607, 608, 614, 615, 616, 619, 620, 621]
22	[714, 715, 717, 724, 718, 725, 719, 728]
23	[629, 630, 632, 636, 633, 637, 635, 638]
24	[779, 784, 785, 786, 788, 795, 789, 790]
25	[391, 392, 394, 398, 395, 399, 402, 404]

Tabella 3: Finestre di Hydrolase(O-Glycosyl) relative ai pattern della prima batteria

6.4 Risultati della seconda batteria di test

Nella seconda batteria si analizza il comportamento dell'algoritmo all'aumentare del numero N di grafi. In questa batteria i parametri utilizzati sono:

a) $W = 8$;

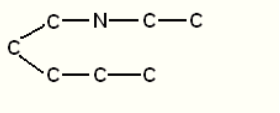
b) $N = 30$;

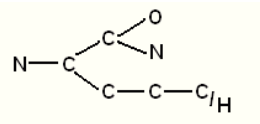
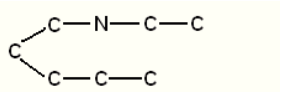
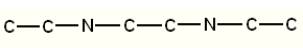
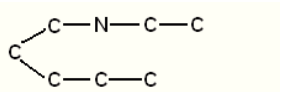
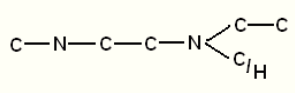
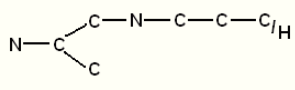
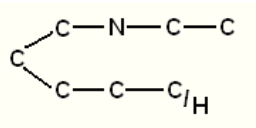
c) $MAX\_ITER = 1000$;

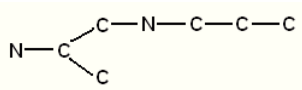
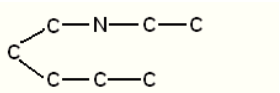
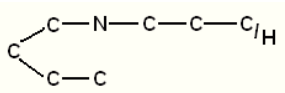
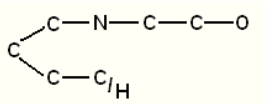
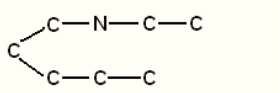
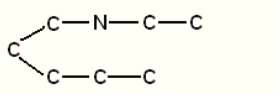
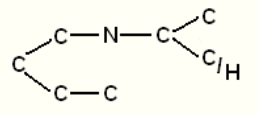
d) $MAX\_SC = maxScore = 2 \times W \times \log_2 |\sum| = 71.35090589819676$;

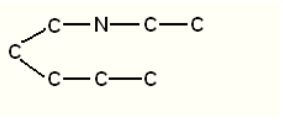
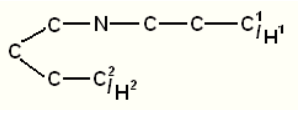
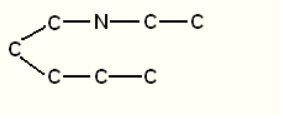
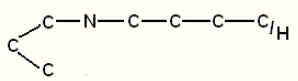
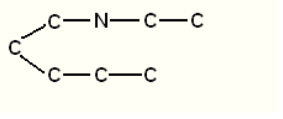
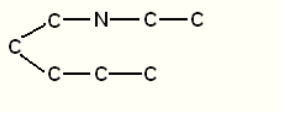
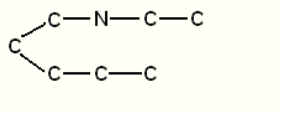
e) $\sigma = 98.5\%$ di $maxScore \simeq 70.28$

I tempi di esecuzione sono più lunghi e vanno dai 2 ai 14 minuti al massimo, ma molte delle caratteristiche evidenziate nella prima batteria si conservano. La Tab. 4 riassume i risultati ottenuti.

TEST	PATTERN	ITER	SCORE	σ-ITER	TEMPO
1		593	MAX_SC	122 (70.4573)	$\sim$ 6.39 min

2	 H: 4 grafi (TOX, IMM, MBP, ISO); C: 26 grafi	527	70.7721	223 (70.3933)	$\sim$ 10.98 min
3		179	MAX_SC	150 (70.41)	$\sim$ 2.01 min
4		346	MAX_SC	266 (70.41)	$\sim$ 3.82 min
5		352	MAX_SC	178 (70.3393)	$\sim$ 3.79 min
6	 C: 5 grafi (HYD_OG, RIN, HYD1, DNP, TD1); H: 25 grafi	553	70.7721	444 (70.399)	$\sim$ 11.91 min
7	 H: 8 grafi (TOX, TRR, IMM, ANT, MBP, PC, PHO, OXI); C: 22 grafi	653	70.5535	155 (70.3933)	$\sim$ 10.69 min
8	 H: 1 grafo (OXI); C: 29 grafi	577	71.1345	161 (70.4215)	$\sim$ 11.16 min

9		383	MAX_SC	183 (70.4215)	$\sim$ 4.49 min
10		337	MAX_SC	150 (70.6546)	$\sim$ 4.08 min
11	 H: 1 grafo (PC); C: 29 grafi	854	71.1345	187 (70.3517)	$\sim$ 10.77 min
12	 H: 7 grafi (ALL, CPO, COM, MBP, OXI, CAD, OXI_C); C: 23 grafi	241	70.6153	201 (70.41)	$\sim$ 11.23 min
13		314	MAX_SC	116 (70.7721)	$\sim$ 3.72 min
14		234	MAX_SC	234 (71.3509)	$\sim$ 2.41 min
15	 H: 1 grafo (GF); C: 29 grafi	701	71.1345	214 (70.3517)	$\sim$ 10.87 min

16		312	MAX_SC	220 (70.3393)	$\sim$ 3.45 min
17	 H_1: 6 grafi (MCD, HYD2, IMM, OXI, TRA, OXI_C); H_2: 3 grafi (COM, PC, GF); C: 21 grafi	706	70.2078	NON RAGG.	$\sim$ 10.89 min
18		821	MAX_SC	162 (70.3933)	$\sim$ 8.91 min
19	 C: 7 grafi (HYD_OG, RIN, HYD1, DNP, TD1, HYD_CP, TD2); H: 23 grafi	250	70.6153	184 (70.5011)	$\sim$ 11.20 min
20		256	MAX_SC	173 (70.4382)	$\sim$ 3.08 min
21		426	MAX_SC	188 (70.4215)	$\sim$ 5.25 min
22		609	MAX_SC	167 (70.3933)	$\sim$ 6.49 min

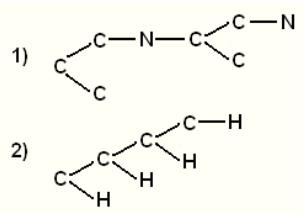
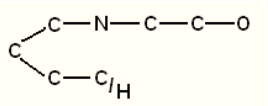
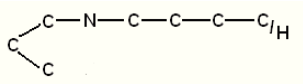
23	 Pattern 1: 4 grafi (HYD_OG, RIN, HYD1, DNP); Pattern 2: 26 grafi	216	67.5122	NON RAGG.	$\sim$ 11.64 min
24	 H: 3 grafi (TRR, OXI, TRA); C: 27 grafi	280	70.871	251 (70.4713)	$\sim$ 12.12 min
25	 C: 7 grafi (HYD_OG, RIN, HYD1, DNP, TD1, TD2, TD4); H: 23 grafi	969	70.6153	318 (70.3724)	$\sim$ 13.50 min

Tabella 4: Risultati della seconda batteria

Dall'analisi della tabella possiamo osservare che:

- In quasi tutti i test, come nella prima batteria, si ricade nello scenario a) oppure b) con una corrispondenza esatta oppure un pattern con un solo nodo cambiato (fa eccezione solo il test 23);
- Il numero di corrispondenze esatte trovate è minore rispetto alla prima batteria (13 su 25 test effettuati), anche se nei test 8, 11 e 15 l'allineamento è quasi perfetto, perché solo in una rete su 30 si ha un pattern

diverso dagli altri e con un solo nodo cambiato. In questi tre casi, probabilmente, si sarebbe potuto ottenere convergenza se si fosse effettuato un numero di iterazioni leggermente superiore a 1000;

- Viceversa, aumenta il numero di corrispondenze non esatte: (12 test su 25);
- Solo nel test 23 la struttura del pattern non è unica: in questo caso, però, vengono trovate due soluzioni esatte ottime, una per un sottoinsieme di 4 reti e una per il resto del dataset (26 reti). Il secondo pattern, in particolare, è la soluzione più interessante perché è comune alla maggior parte dei grafi ed è diverso da quelli trovati negli altri test, per la presenza di ben 4 atomi di idrogeno (H);
- Nel test 17 emergono 3 sottostrutture simili ma diverse, che possono essere ottenute l'una dall'altra sostituendo uno dei due atomi di carbonio (C) terminali con un atomo di idrogeno (H) o viceversa;
- Dal momento che l'algoritmo è non deterministico, è possibile che vengano trovate soluzioni nuove (esatte o no) rispetto alla prima batteria: è quello che accade nei test 2, 6, 7, 8, 9, 12, 15, 17 e 24. Queste sono soluzioni ottime anche per il sottoproblema con $N = 24$ e $W = 8$;
- Ad eccezione dei test 17 e 23, la soglia di ottimalità viene raggiunta e in meno di 300 iterazioni (tranne nei test 6 e 25), anche se alla fine l'algoritmo non converge ad un allineamento perfetto;
- Come nella prima batteria, alcuni pattern vengono trovati più volte in test diversi, anche se rappresentano diverse occorrenze dello stesso sottografo, localizzate in punti diversi di una rete.

6.5 Risultati della terza batteria di test

Nella terza batteria si analizza il comportamento dell'algoritmo all'aumentare della dimensione W delle finestre dell'allineamento. Per questi test i parametri utilizzati sono:

a) $W = 12$;

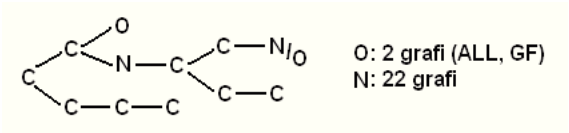
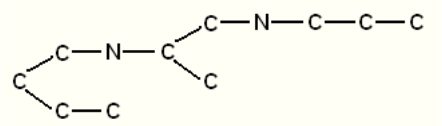
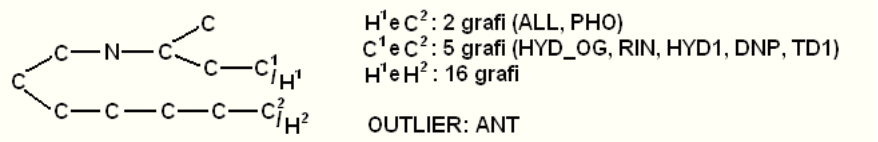
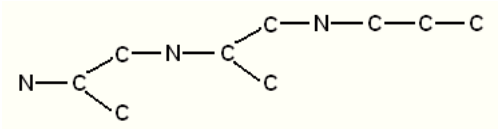
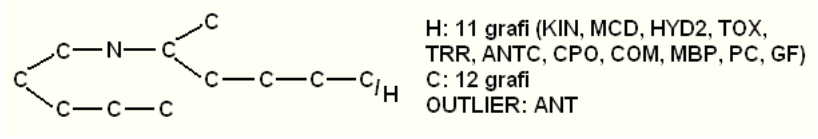
b) $N = 24$;

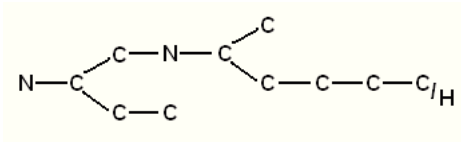
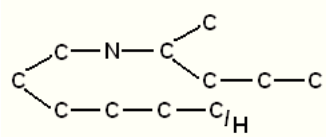
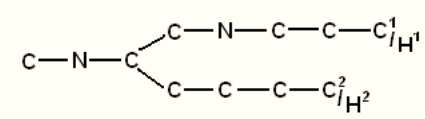
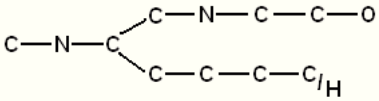
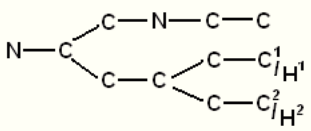
c) $MAX\_ITER = 1000$;

d) $MAX\_SC = maxScore = 2 \times W \times \log_2 |\sum| = 112.81055323538627$;

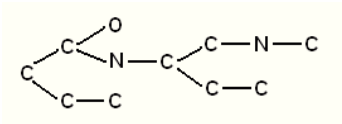
e) $\sigma = 99\%$ di $maxScore \simeq 111.26$

I tempi di esecuzione sono ancora più lunghi e vanno dai 15 ai 116 minuti. La Tab. 5 riassume i risultati ottenuti.

TEST	**PATTERN**			
1				
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	720	112.38432457838812	239 (111.58534550393914)	$\sim$ 116.08 min
2				
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	222	MAX_SC	222 (111.82286072649046)	$\sim$ 92.74 min
3				
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	409	111.16863693000275	NON RAGG.	$\sim$ 106.22 min
4				
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	508	MAX_SC	131 (111.28628772856345)	$\sim$ 55.13 min
5				
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	250	111.81191727122768	193 (111.28628772856345)	$\sim$ 106.07 min

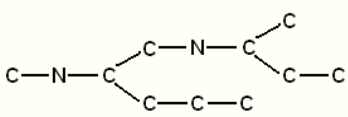
6	 C: 5 grafi (HYD_OG, RIN, HYD1, DNP, TD1) H: 19 grafi			
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	874	112.14397487739134	169 (111.82286072649046)	$\sim$ 105.58 min
7	 C: 7 grafi (HYD_OG, RIN, HYD1, PHO, DNP, TD1, TD4) H: 17 grafi			
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	989	111.98249751000677	674 (111.28628772856345)	$\sim$ 109.80 min
8	 C^1 e C^2: 5 grafi (HYD_OG, RIN, HYD1, DNP, TD1) H^1 e H^2: 19 grafi			
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	337	111.47739651939642	319 (111.38859946482991)	$\sim$ 110.87 min
9	 C: 5 grafi (HYD_OG, RIN, HYD1, DNP, TD1) H: 19 grafi			
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	852	112.05517782282483	117 (111.39663206949231)	$\sim$ 109.63 min
10	 C^1 e C^2: 5 grafi (HYD_OG, RIN, HYD1, DNP, TD1) H^1 e H^2: 18 grafi OUTLIER: TOX			
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	321	111.29980241026341	321 (111.29980241026341)	$\sim$ 108.79 min

11



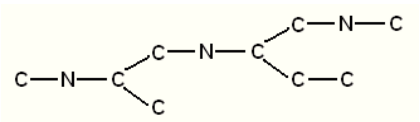
ITER	SCORE	σ-ITER	TEMPO
184	MAX_SC	152 (111.53910404854952)	$\sim$ 19.72 min

12



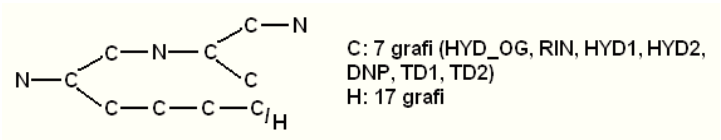
ITER	SCORE	σ-ITER	TEMPO
936	MAX_SC	157 (111.31591915201184)	$\sim$ 106.63 min

13



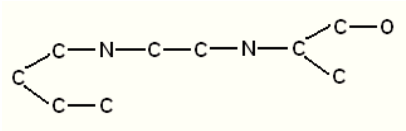
ITER	SCORE	σ-ITER	TEMPO
216	MAX_SC	94 (111.62042299910478)	$\sim$ 22.19 min

14



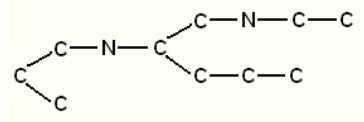
ITER	SCORE	σ-ITER	TEMPO
624	111.98249751000677	369 (111.38235864454276)	$\sim$ 104.13 min

15

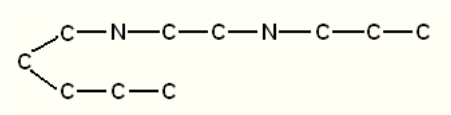
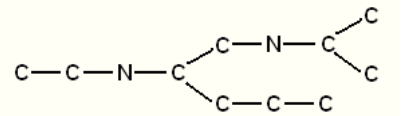
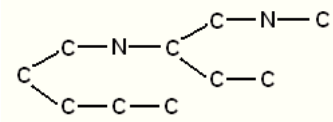
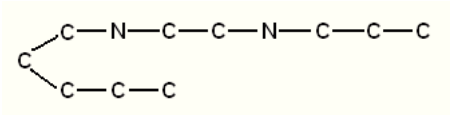
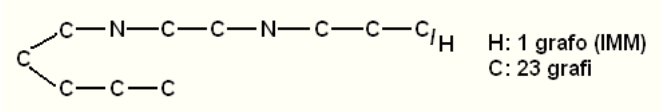
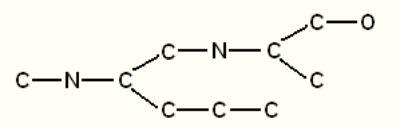


ITER	SCORE	σ-ITER	TEMPO
267	MAX_SC	184 (111.8784416677696)	$\sim$ 29.12 min

16



ITER	SCORE	σ-ITER	TEMPO
846	MAX_SC	128 (111.28112181683038)	$\sim$ 98.38 min

17				
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	308	MAX_SC	152 (111.82286072649046)	$\sim$ 32.14 min
18				
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	203	MAX_SC	148 (111.31591915201184)	$\sim$ 23.61 min
19				
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	410	MAX_SC	148 (111.82286072649046)	$\sim$ 48.93 min
20				
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	660	MAX_SC	147 (111.29587993389805)	$\sim$ 73.74 min
21				
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	783	112.55253456672145	216 (111.41868844501751)	$\sim$ 115.51 min
22				
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	338	MAX_SC	167 (111.4977836855603)	$\sim$ 36.60 min

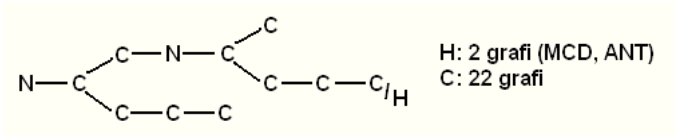
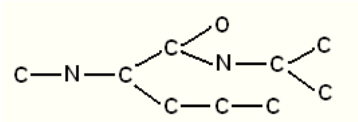
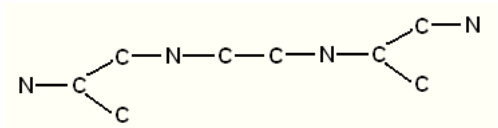
23				
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	628	112.38432457838812	247 (111.31981229431739)	$\sim$ 112.04 min
24				
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	599	MAX_SC	171 (111.41868844501751)	$\sim$ 63.00 min
25				
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	139	MAX_SC	119 (111.44205858406033)	$\sim$ 15.36 min

Tabella 5: Risultati della terza batteria

Dall'analisi della tabella possiamo osservare che:

- Ad eccezione dei test 3,5 e 10, si ricade nello scenario a) oppure b) con una corrispondenza esatta oppure un pattern con uno o due nodi cambiati;
- Il numero di corrispondenze esatte trovate è leggermente superiore rispetto alla seconda batteria (14 su 25 test effettuati), perché è diminuito il numero delle reti (da 30 a 24). Inoltre, nei test 1, 21 e 23 l'allineamento è quasi perfetto, perché solo in una o due reti su 30 si ha un pattern diverso dagli altri e con un solo nodo cambiato. In questi tre casi, probabilmente, si sarebbe potuto ottenere convergenza se si fosse effettuato un numero di iterazioni leggermente superiore a 1000;
- Anche il numero di corrispondenze non esatte non varia rispetto alla seconda batteria (11 test su 25);

- Nei test 3, 5 e 10 si ricade per la prima volta nel caso b) dello scenario c): tutti i grafi, ad eccezione di uno che è 'outlier', si allineano secondo un unico pattern.

  Ciò rappresenta un comportamento strano dell'algoritmo, perché si osserva che il pattern comune è simile a quelli trovati negli altri test, mentre l'outlier trovato è un pattern completamente diverso. Inoltre, gli outlier dei test 3 e 5, relativi allo stesso grafo (*Antibiotic*), sono uguali e sono rappresentati dal sottografo di Fig. 13.

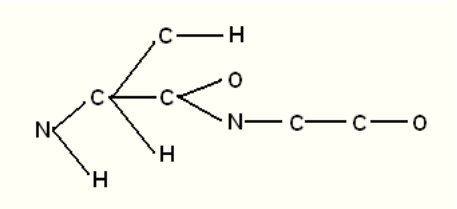


Figura 13: Outlier dei test 3 e 5 della terza batteria

  Nel test 10, invece, l'outlier (relativo alla proteina *Toxin*) è quello rappresentato in Fig. 11;

- Ad eccezione del test 3, la soglia di ottimalità viene raggiunta;

- Ad eccezione dei test 7, 8, 10 e 14, la soglia di ottimalità è raggiunta in meno di 250 iterazioni, anche se alla fine l'algoritmo non converge ad un allineamento perfetto;

- L'algoritmo evidenzia una maggiore varietà di soluzioni: ad eccezione dei test 17 e 20, tutti i pattern trovati sono diversi.

6.6 Conclusioni

Anche se l'algoritmo è non deterministico, dai test si possono trarre alcune considerazioni:

- Aumentando il numero N di reti, occorre aumentare, anche se di poco, il numero di iterazioni per sperare di ottenere in un numero maggiore di casi una corrispondenza esatta, perché in alcuni casi un match perfetto non viene ottenuto;

- Aumentando la dimensione W delle finestre, non muta l'efficacia dell'algoritmo di trovare soluzioni ottimali in un numero fissato di iterazioni;

- In tutti e tre i tipi di test, a parte casi isolati, la soglia di ottimalità è generalmente raggiunta ampiamente entro 1000 iterazioni;
- Solo in 4 casi su 75 non viene trovata una soluzione ottimale. Il caso peggiore, quindi, seppur inevitabile visto il non determinismo, è poco frequente;

La complessità dell'algoritmo, come si può notare dai tempi di esecuzione, dipende principalmente dal numero medio di finestre per ogni grafo: infatti, passando dalla prima alla seconda batteria di test, pur non variando W, i tempi di esecuzione aumentano perché vengono aggiunte al dataset le 6 reti più grandi e/o col maggior numero di finestre.

In tal senso, l'algoritmo è ideale per allineare un numero elevato di grafi piccoli e densi, oppure grandi e sparsi (come le reti di proteine dei test).

Capitolo 7

Ottimizzazioni di LGA Gibbs Sampler

A partire da *LGA Gibbs Sampler* e dall'algoritmo di generazione delle finestre è possibile migliorare, anche sensibilmente, la qualità e la velocità del metodo, sfruttando alcuni risultati parziali già calcolati in esecuzioni precedenti.

7.1 Individuazione di pattern distinti ad ogni esecuzione

Come si può osservare dalle Tab. 2 e 5, alcuni pattern vengono riportati più volte come soluzione ottimale dall'algoritmo, anche se molti di questi, come evidenzia la Tab. 3, rappresentano in realtà diverse occorrenze dello stesso sottografo.

In reti come le proteine utilizzate nei test, in cui occorre valutare la corrispondenza esatta o meno tra le etichette dei nodi, si può utilizzare una procedura che consente di trovare tutte le occorrenze di un pattern restituito da *LGA Gibbs Sampler*, senza dover rieseguire l'algoritmo, e ottenere soluzioni ottimali distinte ad ogni esecuzione dell'algoritmo, aumentando così l'efficacia del metodo.

La procedura *Occurencies* mostra come individuare tutte le occorrenze di un sottografo S di dimensione W all'interno di un grafo G a partire dal suo insieme $Wind$ di finestre di dimensione W.

Occurencies (G, S, Wind)

BEGIN

- Let Occ be the set of occurencies of S in G
$Occ := \varnothing$;
- Let L be the sequence of nodes' labels in S
$L :=$ Build_Sequence_Of_Labels(S);
- Let D be the sequence of nodes' degrees in S
$D :=$ Build_Sequence_Of_Degrees(S);
for $i = 0$ to $|Wind|$ **do**
  - Let L_i be the sequence of nodes' labels of W_i
  $L_i :=$ Build_Sequence_Of_Labels(W_i);
  - Let D_i be the sequence of nodes' degrees of W_i
  $D_i :=$ Build_Sequence_Of_Degrees(W_i);
  if $D = D_i \wedge L = L_i$ **then**
    $Occ := Occ \cup \{Wi\}$;
  end if
end for
return Occ;

END

La procedura si basa sulla costruzione della sequenza di etichette e della sequenza di gradi dei nodi di una finestra, già descritta nel Cap. 4 per il calcolo del LR. Poiché ogni sequenza viene costruita seguendo un ordine ben preciso, quello della BFS eseguita a partire dal seme della finestra, occorrenze di S nel grafo G sono finestre di dimensione W che hanno le stesse sequenze di etichette e di gradi del sottografo S. Al termine dell'esecuzione la procedura restituisce l'insieme di tutte le occorrenze di S in G.

A partire da *Occurencies* si può definire lo schema per ottenere k pattern distinti di dimensione W in k esecuzioni diverse di *LGA Gibbs Sampler* su un insieme G di N grafi.

La procedura *DistinctPatterns*, dopo ogni esecuzione di *LGA Gibbs Sampler*, calcola per ciascun grafo tutte le occorrenze del pattern ottimale trovato, richiamando il metodo *Occurencies*, poi le rimuove dall'insieme delle finestre, in modo che quel pattern non possa più essere individuato rieseguendo il Gibbs Sampler.

DistinctPatterns (G, k, W)

BEGIN

for all G_j in G **do**
- *Let P_j be the set of distinct patterns found in graph G_j*
$Pj := \varnothing$;
end for
for $numEsec = 1$ to k **do**
- *Run LGA Gibbs Sampler using the current set of windows of graphs*
$bestAlign :=$ LGA_Gibbs_Sampler (G, W);
for all G_j in G **do**
- *Let $Wind_j$ be the current set of window of size W of G_j*
$Occ :=$ Occurencies (G_j, $bestAlign$, $Wind_j$);
- *Remove all occurencies found from the set of window of G_j*
$Wind_j := Wind_j - Occ$;
$P_j := P_j \cup Occ$;
end for
end for
return $\{P_1, P_2, ..., P_N\}$

END

Al termine della procedura, gli insiemi P_1, P_2, ..., P_N conterranno tutte le soluzioni ottimali individuate in k esecuzioni di *LGA Gibbs Sampler*, rispettivamente, nei grafi G_1, G_2, ..., G_N.

Se utilizziamo questa procedura, anziché eseguire k volte il Gibbs Sampler classico, si hanno due vantaggi importanti:

a) Otteniamo k pattern tutti diversi tra loro, senza ripetizioni;

b) Poiché *DistinctPatterns* non si limita solo ad eseguire *LGA Gibbs Sampler*, ma trova anche tutte le occorrenze del pattern ottimale (che rappresentano anch'esse soluzioni per l'allineamento), il numero totale di soluzioni ottime individuate alla fine può essere in realtà di gran lunga superiore a k.

7.2 LGA Gibbs Sampler a partire da un insieme ridotto di finestre

Come evidenziato dai test effettuati, il tempo di esecuzione di *LGA Gibbs Sampler* dipende principalmente dal numero medio di finestre in ogni grafo. Se riusciamo ad eseguire il Gibbs Sampling partendo da un insieme molto più piccolo di finestre opportunamente costruito, si può ottenere un Gibbs Sampler più veloce senza perdere in accuratezza.

Si può facilmente osservare che il problema dell'allineamento di grafi gode della proprietà di sottostruttura ottima: se una finestra F di dimensione W è un pattern esatto comune a N grafi, allora anche qualsiasi sottografo di F (di dimensione W') è un pattern esatto comune agli N grafi.

Inoltre, l'algoritmo di generazione delle finestre, *BuildWindows*, illustrato nel Cap. 3, è un algoritmo 'incrementale', nel senso che tutte le finestre di dimensione W possono essere generate per estensione, seguendo gli stessi criteri per la scelta dei nodi da aggiungere alla finestra, a partire da finestre di dimensione $W-1$ o, in generale, W' con $W' < W$.

Si può così definire la seguente variante di *BuildWindows*, che chiamiamo *BuildWindowsFromSeed*, che permette di costruire un insieme di finestre di dimensione W, generate a partire da una finestra F, detta seed, di dimensione W' con $W' < W$, in un grafo G_i.

BuildWindowsFromSeed (F, G_i, W)

BEGIN

- Let a be the first node in F and Wind the set of windows of size W from node a, generated from seed F.
$Wind := \varnothing$;
- Build the BFS tree from node a.
$BFSTree :=$ BFS(G_i, a);
if $|BFSTree| \geq W$ **then**
  - Let C be a queue of window, initially empty.
  C.insertTail(F);
  while C is not empty **do**
    $currWindow := C$.deleteHead();

    if $|currWindow| \neq W$ **then**
      - 1) Let n be the rightmost deepest node of currWindow in the BFSTree. Extend the current window, adding a brother or a cousin

of n in the BFSTree to the right of n (if possible), and then add it to the queue.
for all $x \in \text{Brother}(n) \vee (x \in \text{Cousin}(n)$: $\text{Father}(x) \in currWindow)$ **do**
C.insertTail($currWindow \cup \{x\}$);
end for
- 2) Let DeepestNode be the set of the deepest nodes of currWindow in the BFSTree. Extend the current window, adding a child of a deepest node in the BFSTree (if possible), and then add it to the queue.
for all $n \in DeepestNode$ **do**
for all $x \in \text{Child}(n)$ **do**
C.insertTail($currWindow \cup \{x\}$);
end for
end for
else
- The current window has size W, then it can be added to Wind.
$Wind := Wind \cup currWindow$;
end if

end while
end if
return $Wind$;

END

L'unica differenza sostanziale tra questa procedura e *BuildWindows* è il fatto che si parte da una coda che contiene F e non una finestra di dimensione 1 contenente un solo nodo. Per il resto gli algoritmi sono identici e lavorano allo stesso modo.

Combinando allora le procedure *DistinctPatterns* e *BuildWindowsFromSeed* e sfruttando la proprietà di sottostruttura ottima, è possibile costruire un insieme ridotto di finestre di dimensione W, filtrando quelle che sono soluzioni ottime per il problema dell'allineamento locale, e utilizzare questo set come base di partenza per cercare soluzioni ottimali di dimensioni maggiori.

Supponiamo di partire dagli N insiemi di finestre di dimensione W', uno per ogni grafo dell'insieme G. Vogliamo costruire N insiemi ridotti, uno per ogni grafo, di finestre di dimensione W, con $W > W'$, tali che tutte contengano sottopattern di dimensione W che sono soluzioni ottime per il problema LGA.

Definiamo la seguente procedura *FilteredWindowsSet*.

FilteredWindowsSet (W', G, W, k)

BEGIN

- Let Wind(W') be the set of all windows of size W' and $Wind_i$*(W') the set of windows of size W' in graph* G_i*, for i = 1, 2, ..., N.*
$\{P_1, P_2, ..., P_N\}$:= DistinctPatterns (G, k, W');
for all G_i in G **do**
  - Let $Wind_i(W)$ *be the filtered set of windows of size W in* G_i*.*
  $Wind_i(W) := \varnothing$;
  - Use the elements of P_i *as seeds to generate new windows of size W.*
  for all F_j in P_i **do**
    S := BuildWindowsFromSeed (F_j, G_i, W);
    $Wind_i(W) := Wind_i(W) \cup S$;
  end for
end for
$Wind(W) := \{Wind_1(W), Wind_2(W), ..., Wind_N(W)\}$;
return $Wind(W)$;

END

Il parametro k dell'algoritmo rappresenta di fatto il numero di volte che viene eseguito *LGA Gibbs Sampler* per calcolare i pattern distinti di dimensione W'.

Il valore di k può influenzare molto l'efficacia del metodo, se eseguito successivamente sull'insieme ridotto di finestre di dimensione W restituito da *FilteredWindowsSet*. Aumentando k, aumenta il numero di soluzioni ottime di dimensione W' trovate e di conseguenza anche il numero di finestre di dimensione W, quindi aumenta il numero di potenziali soluzioni ottime di dimensione W per l'allineamento; d'altra parte, aumenta il numero di esecuzioni di *LGA Gibbs Sampler* e il tempo di esecuzione della procedura sale.

Sicuramente il numero delle finestre filtrate è molto più basso del numero totale di finestre calcolate applicando l'algoritmo classico *BuildWindows*, quindi la perdita di tempo legata alle k esecuzioni di *LGA Gibbs Sampler* con parametro W' è ampiamente compensata dal guadagno che si ottiene eseguendo l'algoritmo con parametro W su un insieme di finestre molto più piccolo.

Iterando la procedura *FilteredWindowsSet* possiamo così spingerci nella ricerca verso valori crescenti di W e pattern sempre più grandi senza preoccuparci troppo dell'incremento esponenziale del numero di finestre: l'insieme filtrato restituito dalla procedura può essere utilizzato a sua volta come insieme di partenza per costruire set filtrati più grandi.

Capitolo 8

Confronto tra LGA Gibbs Sampler classico e ottimizzato

Per valutare la bontà del metodo illustrato nel Par. 7.2, utilizziamo lo *Small Dataset* di proteine ($N = 24$) descritto nel Cap. 7. Eseguiamo il metodo *FilteredWindowsSet* con i seguenti parametri:

a) $W' = 8$;

b) $G = SmallDataset$ (24 reti);

c) $W = 12$;

d) $k = 100$;

Dal momento che abbiamo a che fare con reti simili tra loro, in cui esistono pattern esatti, sia di dimensione 8 che di dimensione 12, imponiamo che la procedura *DistinctPatterns* restituisca solo sottografi che siano occorrenze di pattern esatti trovati eseguendo *LGA Gibbs Sampler*, allo scopo di ottenere poi pattern di dimensione 12 quanto più simili tra loro. Pattern non esatti verranno scartati e di essi non verranno calcolate le occorrenze nei vari grafi. Ciò significa che il numero di pattern distinti ottenuti non sarà 100, ma un valore inferiore. In riferimento a questo test, sono stati identificati 35 pattern esatti su 100 esecuzioni.

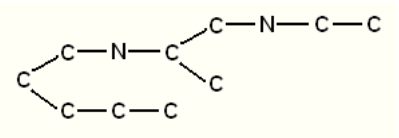
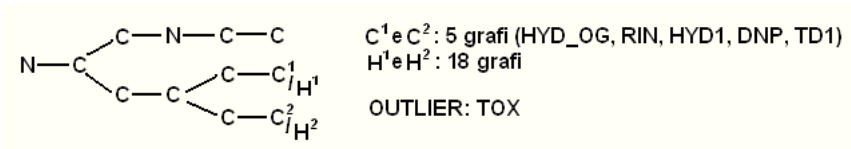
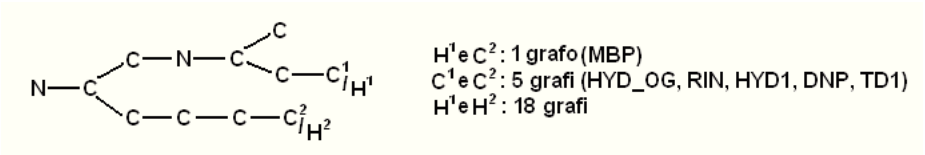
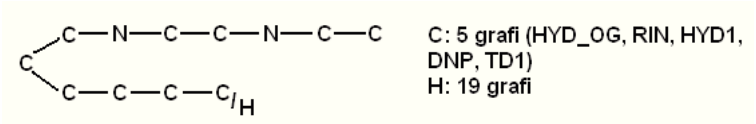
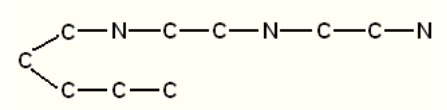
La Tab. 6 indica il numero di finestre ottenute per ogni grafo al termine della procedura, il cui tempo di esecuzione richiesto è di circa 6 ore.

ID	**NOME**	**FINESTRE**
HYD_OG	Hydrolase(O-Glycosyl)	44468
KIN	Kinase	552037
MCD	Mast_Cell_Degranulation	29829
RIN	Ribonuclease_Inhibitor	53772
HYD1	Hydrolase_1	62159
ALL	Allergen	305061
HYD2	Hydrolase_2	224032
TOX	Toxin	51608
TRR	Transcription_Regulation	176961
IMM	Immunoreceptor	271910
ANT	Antibiotic	33586
ANTC	Antibiotic_Chromoprotein	175722
CPO	Complex_(Proto-Oncogene/Early_Protein)	278843
COM	Complement	99679
MBP	Metal_Binding_Protein	189597
PC	Potassium_Channel	58154
GF	Growth_Factor	347448
PHO	Phosphorylation	276794
DNP	De_Novo_Protein	22012
TD1	Transferase/DNA_1	100154
HYD_CP	Hydrolase(C-Terminal_ Peptidase)	115631
TD2	Transferase/DNA_2	166709
TD3	Transferase/DNA_3	159904
TD4	Transferase/DNA_4	162298

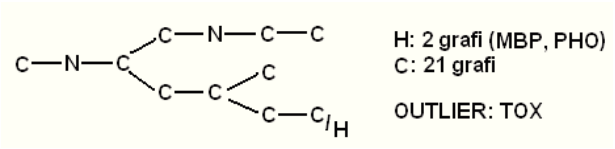
Tabella 6: Numero di finestre nel set ridotto (W=12)

Rispetto ai valori elencati nella Tab. 1, che indica il numero di tutte le finestre di dimensione 12, si può evidenziare una riduzione media significativa (del 90% circa) del numero di finestre generate.

A questo punto, analogamente a quanto fatto nella terza batteria di test (i cui risultati sono stati descritti nel Par. 6.5), effettuiamo 25 esecuzioni di *LGA Gibbs Sampler* utilizzando però questo set ridotto di finestre. La Tab. 7 riassume i risultati ottenuti in questa quarta batteria di test.

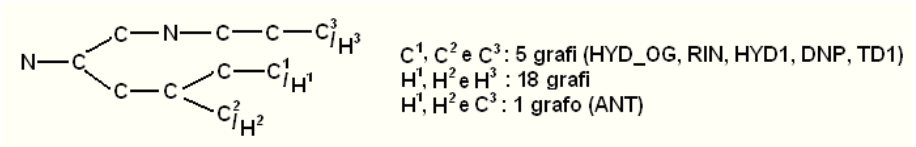
<table>
<tr><th>TEST</th><th colspan="4">PATTERN</th></tr>
<tr><td>1</td><td colspan="4">  </td></tr>
<tr><td></td><td>ITER</td><td>SCORE</td><td>σ-ITER</td><td>TEMPO</td></tr>
<tr><td></td><td>162</td><td>MAX_SC</td><td>103 (112.1263059097233)</td><td>$\sim$ 1.80 min</td></tr>
<tr><td>2</td><td colspan="4"> 
</td></tr>
<tr><td></td><td>ITER</td><td>SCORE</td><td>σ-ITER</td><td>TEMPO</td></tr>
<tr><td></td><td>174</td><td>111.29980241026341</td><td>174 (111.29980241026341)</td><td>$\sim$ 10.99 min</td></tr>
<tr><td>3</td><td colspan="4"> 
</td></tr>
<tr><td></td><td>ITER</td><td>SCORE</td><td>σ-ITER</td><td>TEMPO</td></tr>
<tr><td></td><td>558</td><td>111.38859946482991</td><td>146 (111.31591915201184)</td><td>$\sim$ 11.04 min</td></tr>
<tr><td>4</td><td colspan="4"> 
</td></tr>
<tr><td></td><td>ITER</td><td>SCORE</td><td>σ-ITER</td><td>TEMPO</td></tr>
<tr><td></td><td>644</td><td>112.14397487739134</td><td>119 (111.45221301077144)</td><td>$\sim$ 11.33 min</td></tr>
<tr><td>5</td><td colspan="4">  </td></tr>
<tr><td></td><td>ITER</td><td>SCORE</td><td>σ-ITER</td><td>TEMPO</td></tr>
<tr><td></td><td>662</td><td>MAX_SC</td><td>185 (111.27256912717266)</td><td>$\sim$ 7.14 min</td></tr>
</table>

6



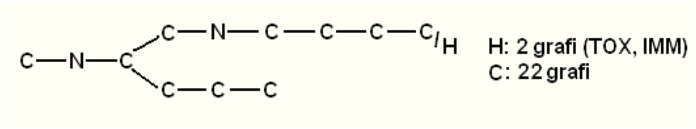
ITER	**SCORE**	**σ-ITER**	**TEMPO**
912	112.38432457838812	89 (111.36240433043996)	$\sim$ 11.38 min

7



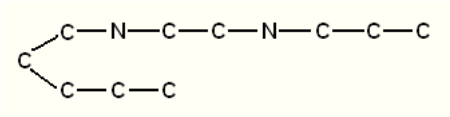
ITER	**SCORE**	**σ-ITER**	**TEMPO**
777	110.72202110683499	NON RAGGIUNTA	$\sim$ 11.56 min

8



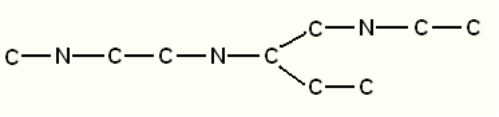
ITER	**SCORE**	**σ-ITER**	**TEMPO**
223	112.38432457838812	131 (112.14397487739134)	$\sim$ 11.43 min

9



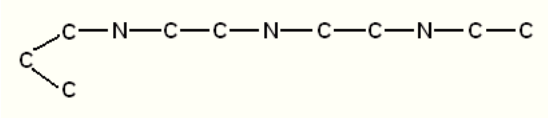
ITER	**SCORE**	**σ-ITER**	**TEMPO**
266	MAX_SC	154 (111.33651151791263)	$\sim$ 2.97 min

10

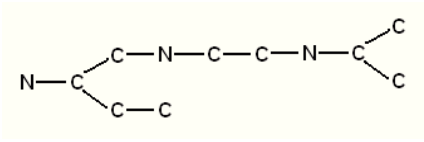
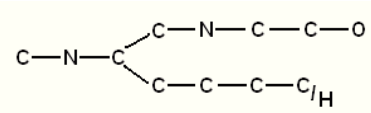
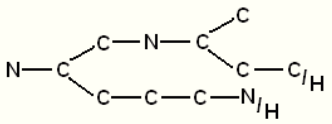
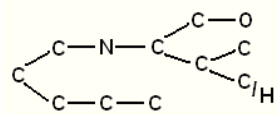
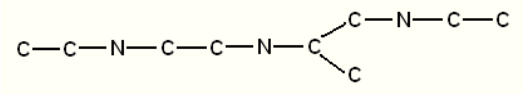
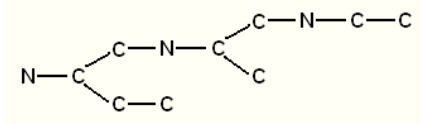


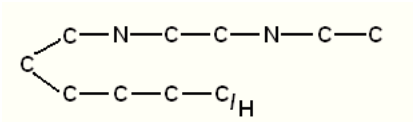
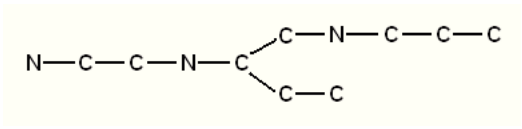
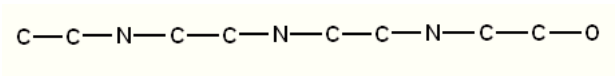
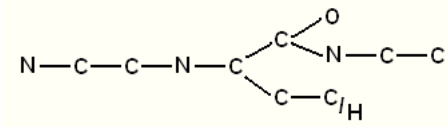
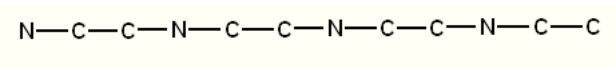
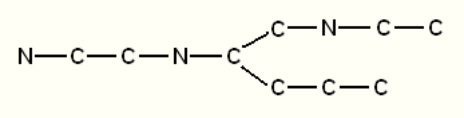
ITER	**SCORE**	**σ-ITER**	**TEMPO**
225	MAX_SC	159 (111.61026857239368)	$\sim$ 2.36 min

11



ITER	**SCORE**	**σ-ITER**	**TEMPO**
122	MAX_SC	115 (112.55253456672145)	$\sim$ 1.49 min

12				
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	756	MAX_SC	151 (111.56484205782564)	$\sim$ 8.26 min
13	 C: 6 grafi (HYD_OG, RIN, HYD1, DNP, TD1, HYD_CP) H: 18 grafi			
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	693	112.05517782282483	467 (111.31981229431739)	$\sim$ 11.06 min
14	 C e N: 5 grafi (HYD_OG, RIN, HYD1, DNP, TD1) H e H: 19 grafi			
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	379	111.47739651939642	379 (111.47739651939642)	$\sim$ 11.72 min
15	 C: 5 grafi (HYD_OG, RIN, HYD1, DNP, TD1) H: 19 grafi			
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	443	112.14397487739134	143 (111.30682338916083)	$\sim$ 10.79 min
16				
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	119	MAX_SC	82 (111.35224990372886)	$\sim$ 1.33 min
17				
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	163	MAX_SC	105 (111.35224990372886)	$\sim$ 1.77 min

18	 C: 6 grafi (HYD_OG, RIN, HYD1, DNP, TD1, TD4) H: 18 grafi			
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	394	112.05517782282483	116 (111.36614163249763)	$\sim$ 11.66 min
19				
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	309	MAX_SC	154 (111.31981229431739)	$\sim$ 3.50 min
20				
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	150	MAX_SC	121 (111.53061558081306)	$\sim$ 1.66 min
21	 H: 2 grafi (MCD, TD2) C: 22 grafi			
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	391	112.38432457838812	76 (111.38568861422955)	$\sim$ 11.47 min
22				
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	113	MAX_SC	97 (111.53186726439182)	$\sim$ 1.15 min
23				
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	284	MAX_SC	97 (111.4778678559396)	$\sim$ 3.13 min

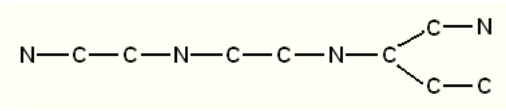
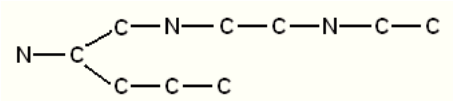
24				
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	152	MAX_SC	94 (111.27384859572702)	$\sim$ 1.65 min
25				
	ITER	**SCORE**	**σ-ITER**	**TEMPO**
	112	MAX_SC	105 (112.29451589805663)	$\sim$ 1.23 min

Tabella 7: Risultati della batteria di test su GS ottimizzato

I tempi di esecuzione vanno da poco più di un minuto a circa 12 minuti, con una riduzione dello stesso fattore (circa 90%) di quella del numero di finestre nel caso del Gibbs Sampler classico, a conferma del fatto che la complessità dipende principalmente e quasi esclusivamente dal numero medio di finestre per ogni grafo.

La Tab. 8 mette a confronto il Gibbs Sampler ottimizzato con quello classico sulla base di alcuni parametri e misurazioni.

	GS classico	**GS ottimizzato**
Percentuale di pattern esatti trovati	56% (14/25)	56% (14/25)
Media delle iterazioni in cui si raggiunge un match esatto	$\sim$ 460	$\sim$ 257
Numero minimo di grafi allineati in modo esatto	12	18
Valore medio di σ-iter (quando raggiunta)	$\sim$ 208	$\sim$ 149
Numero di iterazioni in cui σ-iter non è raggiunta	1	1
Numero di iterazioni in cui si è ottenuto un outlier	3	2
Tempo medio di esecuzione di un test (su 1000 iter.)	$\sim$ 76.88 min	$\sim$ 6.55 min

Tempo totale richiesto per i 25 test (su 1000 iter.)	$\sim$ 32.03 h	$\sim$ 2.73 h
Tempo richiesto per il pre-processing	$\sim$ 4.53 h (calcolo finestre con W=12)	$\sim$ 0.16 h (calcolo finestre con W'=8) + $\sim$ 6 h (*Filtered-WindowsSet*)
Tempo totale richiesto (pre-processing + test)	$\sim$ 36.56 h	$\sim$ 8.89 h

Tabella 8: Confronto tra GS classico e ottimizzato

Per il confronto tra i due metodi sono stati considerati i seguenti parametri:

1. La percentuale, in rapporto al numero totale di test, di pattern esatti trovati. Pur considerando la natura non deterministica dei due metodi, in questo caso possiamo dire che i due algoritmi si comportano sostanzialmente allo stesso modo.

2. Media aritmetica delle iterazioni in cui si ottiene un pattern esatto, considerando, quindi, solo i 14 casi in cui è stato raggiunto il punteggio massimo. Per il Gibbs Sampler ottimizzato questo valore è molto più basso: ciò significa che converge più velocemente.

3. Numero minimo di grafi allineati in maniera esatta in una esecuzione del Gibbs Sampler: dall'analisi delle Tab. 5 e 7 questo valore è, rispettivamente, 12 per il GS classico (test 5) e 18 per il GS ottimizzato (test 2, 3, 7, 13 e 18). Il metodo ottimizzato sembra rivelarsi, quindi, leggermente più accurato.

4. Media aritmetica delle σ-iter, cioè delle iterazioni a partire dalle quali la soglia di ottimalità viene superata, nei 24 casi su 25 in cui σ è stata raggiunta. Come per la misura 2., il valore ottenuto per GS ottimizzato è molto più basso, a conferma del fatto che quest'ultimo metodo converge più velocemente. E questo risultato assume ancora più importanza se si tiene conto del fatto che spesso, ad esempio nelle network biologiche, non esiste una corrispondenza esatta ed è importante cercare una soluzione ottima approssimata.

5. Numero di iterazioni in cui la soglia di ottimalità non viene raggiunta: in questo caso i due metodi registrano un comportamento simile.

6. Numero di iterazioni in cui è stato ottenuto un outlier, cioè un grafo che non si allinea con gli altri: contrariamente a quanto ci si può aspettare, anche GS ottimizzato può far emergere un outlier (con un pattern topologicamente diverso dagli altri). In realtà, la presenza di outlier per il metodo ottimizzato è da legare al fatto che è stato effettuato il filtering passando direttamente da finestre di una certa dimensione ($W' = 8$) a finestre molto più grandi ($W = 12$).

7. Tempo medio di esecuzione di un test e tempo totale di esecuzione dei 25 test: in questo caso lo scarto tra i valori dei metodi è molto netto, con un fattore 10 o poco più di differenza e una riduzione dei tempi del 90% o poco più.

8. Tempo di preprocessing richiesto. Nel caso del GS classico il preprocessing consiste nel calcolo dell'intero set di finestre con $W = 12$, mentre per il GS ottimizzato consiste nel calcolo dell'intero set di finestre di dimensione $W' = 8$ e nell'applicazione della procedura *FilteredWindowSet*. In questo caso la versione classica impiega meno tempo di quella ottimizzata, perché l'efficienza di quest'ultima dipende anche dal fatto che si parte da un valore di W' abbastanza elevato e dalla necessità di avere ad ogni esecuzione del Gibbs Sampling pattern distinti.

9. Tempo totale (preprocessing + 25 test), nettamente a favore del GS ottimizzato.

Riassumendo, il metodo ottimizzato è sicuramente molto più veloce di quello classico, converge più velocemente ad una soluzione ottimale e sembra essere accurato al pari (se non di più) del metodo tradizionale, se il filtering delle finestre viene effettuato eseguendo un numero k sufficiente di volte (nell'esempio $k = 100$) il Gibbs Sampling, in modo da catturare quante più soluzioni ottimali possibili.

Capitolo 9

Possibili applicazioni e sviluppi futuri

L'applicazione più importante in cui potrebbe essere impiegato l'algoritmo, che è anche quella che ha motivato la progettazione di questo metodo, sulla scia del Gibbs Sampler già sperimentato con successo sulle sequenze proteiche (cf. [4]), è l'allineamento multiplo di network biologiche.

Nelle network di interazioni molecolari i nodi rappresentano geni o proteine e gli archi le possibili interazioni. L'obbiettivo dell'allineamento è quello di individuare pathway (cioè cammini) o cluster (cioè sottostrutture) di proteine o geni che regolano processi biologici complessi e che interagiscono in modo simile e svolgono funzioni equivalenti in organismi viventi diversi tra loro. Ciò consente di avere una conoscenza più completa di un sistema complesso (come un organismo) in cui tante parti interagiscono tra loro (geni e proteine) e delle cause che portano alla nascita di alcune malattie, ma anche di predire nuove interazioni genetiche o scoprire le funzionalità di geni poco conosciuti.

Apportando qualche piccola modifica, l'algoritmo si potrebbe applicare alle network biologiche, considerando gli score di omologia tra le proteine come parametro per valutare il grado di corrispondenza tra le etichette dei nodi della rete.

In futuro, l'obbiettivo è proprio quello di testare l'efficacia dell'algoritmo su network di interazioni proteina-proteina (PPI) eucariotiche e microbiali e confrontarlo con altri algoritmi di allineamento già esistenti, come *NetworkBlast-M* (cf. [5, 6]) e *Graemlin* (cf. [7]), che sono metodi sviluppati ad hoc per le network biologiche e la cui complessità computazionale dipende fortemente dal numero di reti considerate.

LGA Gibbs Sampler, comunque, è un metodo di allineamento generale, che può essere eseguito su qualsiasi tipo di rete, quindi si potrebbe prestare,

con qualche variazione, a molte applicazioni nel campo del graph mining in generale, come ad esempio la ricerca di occorrenze di una query in un database di grafi.

Bibliografia

[1] S. Geman and D. Geman, *Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images*, IEEE Transactions on Pattern Analysis and Machine Intelligence (1984), **12**, 609-628.

[2] J. Liu, *The collapsed Gibbs Sampler and other issues: with applications to a protein binding problem*, Research Report R-426 (1992), Dept. Statistics, Harvard Univ.

[3] N. Metropolis and S. Ulam, *The Monte Carlo method*, J. Am. Statist. Assoc. (1949), **44**, 335-341.

[4] Charles E. Lawrence, Stephen F. Altschul, Mark S. Boguski, Jun S. Liu, Andrew F. Neuwald, John C. Wootton, *Detecting Subtle Sequence Signals: A Gibbs Sampling Strategy for Multiple Alignment*, Science (1993), **262**, 208-214.

[5] Roded Sharan, Silpa Suthram, Ryan M. Kelley, Tanja Kuhn, Scott McCuine, Peter Uetz, Taylor Sittler, Richard M. Karp, Trey Ideker, *Conserved patterns of protein interaction in multiple species*, PNAS (2005), **6**, Vol. 102, 1974-1979.

[6] Maxim Kalaev, Vineet Bafna, Roded Sharan, *Fast And Accurate Alignment of Multiple Protein Networks*, Journal of Computational Biology (2009), **16**, 989-999.

[7] Jason Flannick, Antal Novak, Balaji S. Srinivasan et al., *Graemlin: General and robust alignment of multiple large interaction networks*, Genome Res. (2006), **16**, 1169-1181.