UNIVERSITÀ DEGLI STUDI DI ROMA TRE

FACOLTÀ DI SCIENZE M.F.N.

Algoritmi euristici per allineamento di sequenze

Sintesi della tesi di Laurea in Matematica

di Giorgio Minelle

Relatore: Pasquale Caianiello

Abstract

Il confronto di tratti di DNA o di una proteina alla ricerca di zone di similarità, noto come *allineamento di sequenze*, viene modellato come il confronto tra stringhe di caratteri che rappresentano gli elementi chimici costituenti il DNA o la proteina. Pertanto la Biologia Molecolare riguarda sequenze: fenomeni biochimici complessi vengono definiti come problemi computazionali su stringhe. La tesi nell'ambito di questo contesto, tratterà il problema della definizione e del calcolo delle omologie tra sequenze, studiando il comportamento di alcuni algoritmi euristici per l'allineamento.

Un aspetto comune a tutte le discipline è l'*analisi comparativa* che mette in luce possibili differenze o eventuali omologie tra gli oggetti studiati. In Biologia è noto a molti come le intuizioni di Darwin siano nate dal confronto della morfologia degli animali presenti sulle isole Galapagos e come tali osservazioni gli permisero di postulare la *teoria dell'Evoluzione*. Concettualmente lo stesso tipo di analisi viene oggi realizzata nel campo della Biochimica e della Bioinformatica per confrontare sequenze di geni o di proteine.

La Biochimicha è quel settore della Chimica che si occupa dello studio della composizione e delle trasformazioni biologiche dei composti essenziali alla vita delle cellule e degli organismi.

La conoscenza dettagliata della struttura architettonica e chimica delle molecole organiche più complesse (come l'acido desossiribonucleico, il DNA), che costituiscono l'organismo vivente, ha messo in luce che alcune anomalie o malattie (*le malattie molecolari*) sono tutte riconducibili ad una alterazione strutturale delle molecole "chiavi" (cioè quelle fondamentali per una determinata funzione) di alcune delle sostanze più importanti.

In una sequenza di DNA si possono verificare, nel corso del tempo, delle mutazioni; se tali errori avvengono in una regione che codifica una proteina, automaticamente questa malformazione strutturale si ripercuote sulla sequenza primaria della proteina stessa e può essere tale da modificarne sia la forma che la funzione biologica; questo meccanismo si può tradurre in una modificazione funzionale di tutto l'organismo, di cui non è possibile prevedere nè l'enitità, nè l'importanza. Nell' Anemia Falciforme, per esempio, in una delle quattro catene peptidiche dell' Emoglobina un residuo di acido Glutammico è sostituito da uno di Valina: basta quindi la modifica di un unico amminoacido (su un totale di 146 !) per avere uno stato patologico.

Partendo in passato dalla convinzione che proteine con strutture primarie simili potessero esplicare funzioni simili, si è cercato di studiare delle metodologie per il loro confronto e si è riscontrato che molti dei risultati ottenuti sono stati di notevole aiuto per la comprensione delle funzioni di proteine di nuova scoperta.

Si è capito ben presto, anche a causa delle inimmaginabili dimensioni del DNA, che l'unica tecnica di analisi attuabile era quella del confronto: cercare nel DNA o all'interno di una proteina un settore simile ad uno già campionato, per il quale si conoscono anche caratteristiche e funzionalità, nella speranza che le sue proprietà restino valide nella sezione di DNA o di proteina sconosciuta.

Una fondamentale assunzione in biologia molecolare è che si possano ottenere risultati biologici significativi considerando il DNA come una stringa monodimensionale (su un alfabeto di 4 simboli) astraendo dalla sua reale natura di molecola tridimensionale. Tale asserto rimane valido anche per

le proteine, in quanto la conformazione dell'avvolgimento tri-dimensionale delle proteine è completamente determinato nella sequenza lineare codificante la proteina (in questo caso l'alfabeto è di 20 simboli). Pertanto la Biologia Molecolare riguarda sequenze: essa riduce fenomeni biochimici complessi alla interazione tra sequenze definite. Non sorprende allora il fatto che problemi fondamentali in Biologia Molecolare siano definiti come problemi computazionali su sequenze.

Di pari passo alla crescita della Biochimica è nata una disciplina che, sfruttando il progresso della tecnologia, si è dedicata totalmente alla gestione e alla interpretazione dei dati di tipo biologico raccolti: questa sezione della biologia si chiama Bioinformatica.

In poche parole la Bioinformatica è l'applicazione degli strumenti informatici al servizio dei problemi di tipo biologico, quindi comprende:

1. strumenti dediti alla gestione e all'utilizzo delle banche dati di sequenze proteiche o di acidi nucleici.

2. tutti gli algoritmi necessari all'analisi delle sequenze e alla predizione del comportamento delle corrispondenti molecole.

3. tutti gli strumenti per la condivisione delle informazioni di natura biologica.

Questa tesi verterà sopratutto sul secondo punto, illustrando nella prima parte concetti e algoritmi già noti e famosi in biologia computazionale e dedicandosi, nella seconda parte, allo sviluppo e alla sperimentazione di algoritmi basati su una strategia *euristica*, per risolvere il problema dell'allineamento tra sequenze. In questa tesi verranno studiati, con particolare attenzione, metodi più efficienti e veloci di allineamento, privilegiando l'utilizzo di algoritmi di tipo approssimato piuttosto che algoritmi di tipo ottimo.

Gli algoritmi *ottimi* (o *esatti*), presentano il vantaggio di assicurare sempre, prima o poi, il raggiungimento della soluzione, ma anche lo svantaggio dell'anti-economicità: tali strategie richiedono tempo, poichè esigono che siano tentate tutte le possibili soluzioni e vagliate tutte le ipotesi; Si impone cioè la generazione di tutte le possibili combinazioni ("esplosione com-

binatoriale"), che può, in casi di spazio problemico molto esteso, risultare inattuabile.

Gli algoritmi che si basano su strategie di tipo *euristico* invece, pur non garantendo sempre una soluzione ottima, sono in grado di risolvere problemi di ottimizzazione molto complessi, in tempi computazionali ragionevoli; per farlo limitano il numero delle alternative possibili da esaminare a quelle che sembrano aver maggiore possibilità di successo, basandosi sulle caratteristiche strutturali specifiche del problema. In altre parole le ricerche euristiche sono metodi approssimati per risolvere problemi con un costo ragionevole di risorse computazionali: il tempo di computazione viene ridotto a scapito della qualità della soluzione.

Trovare la soluzione di problemi di ottimizzazione, quale l'allineamento globale o locale di sequenze di amminoacidi o di acidi nucleici, può risultare troppo oneroso e inattuabile non appena la dimensione dello spazio problemico diviene significativa: per esempio quando le due proteine da confrontare sono molto estese oppure quando si vuole confrontare una proteina con un intero database proteico. Il ricorso ad algoritmi di approssimazione con complessità polinomiale, diviene quindi indispensabile quando la dimensione del problema cresce.

Lo scopo della tesi é quello di affrontare il problema dell'allineamento tra sequenze attivando delle strategie euristiche: rinunciare alla ricerca della soluzione ottima in cambio di un considerevole risparmio computazionale ed accettare delle soluzioni "buone", che sperabilmente non siano troppo lontane dall'ottimo.

Il lavoro della tesi è strutturato nel seguente modo:

1. Definizioni di base

Nel primo capitolo si compie una breve escursione nel campo della Biologia Molecolare e della Bioinformatica [Wat 95], [BO 98] (primi due paragrafi), illustrando quei concetti chiave e quelle definizioni basilari, così importanti ai fini di una completa conoscenza del problema oggetto di studio.

Definizione 1. *Un* **alfabeto** $\mathcal{A}$ *è un insieme finito di elementi detti simboli, caratteri o lettere.*
Sia $\mathcal{A}^n$ *l'insieme di tutte le parole su* $\mathcal{A}$ *con lunghezza pari ad* n*, si ha che* $\mathcal{A}^* = \bigcup_{n=0}^{\infty} \mathcal{A}^n$ *è l'insieme di tutte le parole finite su* $\mathcal{A}$*.*

Definizione 2. *Una* **stringa** *o* **parola** $\mathbf{s}$ *sull'alfabeto* $\mathcal{A}$ *è una sequenza finita o nulla di simboli di* $\mathcal{A}$*:*

$$\mathbf{s} = (a_1, a_2, ..., a_n), a_i \in \mathcal{A} \tag{1}$$

Definizione 3. *Una parola* $\mathbf{s_1}$ *è un* **fattore sinistro** *o un* **prefisso** *di una parola* $\mathbf{s} \in \mathcal{A}^*$ *se esiste una parola* $\mathbf{s_2} \in \mathcal{A}^*$ *tale che* $\mathbf{s} = \mathbf{s_1 s_2}$*.*
Indichiamo con $\mathbf{s_{[0,i]}}$ *il prefisso di* $\mathbf{s}$ *composto dalle sue prime i lettere.*

Segue poi una presentazione generale (ultimi due paragrafi) dei temi principali nel campo dell'informatica e della ricerca operativa (*algoritmi e complessità, problemi di ottimizzazione, ricerca euristica....*).

2. Confronto fra sequenze

Nel secondo capitolo, si illustrano dettagliatamente i concetti utilizzati in Biologia computazionale per il problema dell'allineamento di sequenze proteiche o di acidi nucleici.

Trovare la similarità tra sequenze di proteine o di acidi nucleici è un problema alla base di molte analisi biologiche e non è affatto così banale come si potrebbe pensare. Già la parola in se di similarità è poco chiara, infatti non esiste nel linguaggio umano un significato univoco e ben preciso per il concetto che essa esprime. Ad esempio basta considerare le parole "cane", "cave" e "ave": sono molto simili nel suono e nel loro *spelling* ma completamente differenti per quanto riguarda il significato. Anche le proteine e i segmenti di DNA possono avere similarità rispetto alla loro struttura, alla loro funzione o alla loro sequenza primaria di amminoacidi o di acidi nucleici. Per fortuna in biologia molecolare la regola generale è che le sequenze primarie determinano la forma della proteina e questa a sua volta ne determina la funzione. Quindi si cerca di studiare la similarità tra sequenze per poter così scoprire o confermare similarità di struttura e di funzione.

Quando si parla di similarità bisogna introdurre il concetto di allineamento: non posso stabilire quando due sequenze sono simili se non le allineo nè tantomeno posso allinearle senza avere dei criteri per definire cosa vuol dire similarità. L'**Allineamento** è un ordinamento tra due o più sequenze che mostra dove le sequenze sono simili e dove differiscono; mi dà una risposta qualitativa al problema. Per quantificare la similarità tra due sequenze posso usare o la **Misura di similarità** che associa un valore numerico ad una coppia di sequenze, con l'idea che più questo valore è alto e più avrò un maggiore grado di similarità; oppure la funzione **distanza** che associa un valore numerico ad ogni coppia di sequenze con l'idea che più grande sarà la distanza e più piccola risulterà la similarità e viceversa.

Osservazione 1. la nozione di distanza è duale a quella di similarità ed in molti casi si può affermare che siano tra loro intercambiabili: infatti provare a massimizzare la misura di similarità tra sequenze corrisponde a minimizzarne la distanza.

Un esempio di distanza tra due stringhe é la **distanza di editing**, che viene definita come la misura del costo minimo di operazioni "evoluzionarie" necessarie per trasformare una stringa nell'altra.

Le operazioni primitive necessarie per trasformare una sequenza in un altra si chiamano *edit operations*

Definizione 4. *Sia $\mathcal{A}$ un alfabeto e "$-$" il carattere vuoto detto* **gap***, una* **edit operation** *è una coppia $(\mathbf{a}, \mathbf{b}) \neq (-, -)$, dove $\mathbf{a}, \mathbf{b} \in \mathcal{A}$ e $|\mathbf{a}|, |\mathbf{b}| \leq \mathbf{1}$*

Si hanno le sequenti edit operations:

- $(\mathbf{a}, \mathbf{b})$ **substitution** *: denota un cambio di carattere in un altro.* (**match** = *substitution tra due caratteri uguali*).
- $(-, \mathbf{b})$ **insertion** *: denota l'inserimento di un carattere.*
- $(\mathbf{a}, -)$ **deletion** *: denota la cancellazione di un carattere.*

Ad ogni edit operation assegnamo un costo o un peso attraverso una funzione *costo*

Definizione 5. *La funzione* **costo**

$$\begin{aligned} \delta \quad : \quad & (\mathcal{A}^* \cup \{-\})^2 \longrightarrow \mathbb{R}^+ \\ & (a,b) \longrightarrow \delta(a,b) \end{aligned} \tag{2}$$

associa un valore reale e non negativo ad ogni operazione di edit $a \longrightarrow b$*, tra due caratteri. Il costo di un edit operation di tipo match è pari a zero, in simboli* $\delta(a,a) = 0$

Definizione 6. *Il* **costo di un allineamento E** *di due stringhe* $\mathbf{X}$ *e* $\mathbf{Y}$ *è la somma dei costi di tutte le edit operation che mi "trasformano"* $\mathbf{X}$ *in* $\mathbf{Y}$

$$\delta(\mathbf{X} \stackrel{\mathbf{E}}{\Longrightarrow} \mathbf{Y}) = \sum_{(a,b)\in\mathbf{E}} \delta(a,b) \tag{3}$$

Ovviamente esistono diversi allineamenti per ogni coppia di sequenze, noi siamo interessati a trovare quello il cui costo associato sia minimo.

Definizione 7. *Un* **allineamento ottimale** *di due stringhe* $\mathbf{s}, \mathbf{t} \in \mathcal{A}^*$*, è l'allineamento che ha il minimo costo tra tutti gli altri possibili allineamenti*

Definizione 8. *Date due stringhe* $\mathbf{s}, \mathbf{t} \in \mathcal{A}^*$ *e una funzione costo* δ*, la* **distanza di editing** *di* $\mathbf{x}$ *e* $\mathbf{y}$ *è il costo di un allineamento ottimale di* $\mathbf{x}$ *e* $\mathbf{y}$ *rispetto alla funzione costo* δ *e viene indicata con* $\mathbf{d}_\delta(\mathbf{x}, \mathbf{y})$.

Il numero dei possibili allineamenti tra due sequenze può essere enorme, quindi a meno che non si usi un modello dei costi per le edit operation molto semplice, può sembrare abbastanza difficile trovare degli allineamenti ottimali e quindi calcolare l'edit distance. Fortunatamente vi sono dei procedimenti semplici che si basano su algoritmi di tipo ricorsivo che hanno semplificato il problema: Consideriamo due prefissi $\mathbf{s}_{[\mathbf{0},\mathbf{i}]}$ e $\mathbf{t}_{[\mathbf{0},\mathbf{j}]}$, con $i, j \geqslant 1$, l'edit distance viene calcolata attraverso la seguente formula ricorsiva:

$$\begin{aligned} d_\delta(\mathbf{s}_{[\mathbf{0},\mathbf{i}]}, \mathbf{t}_{[\mathbf{0},\mathbf{j}]}) = min\{ & d_\delta(\mathbf{s}_{[\mathbf{0},\mathbf{i-1}]}, \mathbf{t}_{[\mathbf{0},\mathbf{j-1}]}) + \delta(\mathbf{s_i}, \mathbf{t_j}), \\ & d_\delta(\mathbf{s}_{[\mathbf{0},\mathbf{i-1}]}, \mathbf{t}_{[\mathbf{0},\mathbf{j}]}) + \delta(\mathbf{s_i}, -), \\ & d_\delta(\mathbf{s}_{[\mathbf{0},\mathbf{i}]}, \mathbf{t}_{[\mathbf{0},\mathbf{j-1}]}) + \delta(-, \mathbf{t_j})\} \end{aligned} \tag{4}$$

L'algoritmo ricorsivo che illustreremo (in versione pseudo codice) è molto noto in biologia molecolare (Needleman-Wunsch algorithm, [NW 70]) ed è

stato creato utilizzando la tecnica della *programmazione dinamica* [RB 57].

PROCEDURA CALCOLA-ALLINEAMENTO (A, B, δ, m, n)

- $A[1, \ldots, m]$ e $B[1, \ldots, n]$ sono due stringhe di lunghezza rispettivamente m e n
- δ è la funzione costo
- D è la matrice distanza con coefficienti: $D[i, j] = d_\delta(A[i], B[j])$

$D[0, 0] = 0$

for $j = 1$ to n **do**

$$D[0, j] = D[0, j-1] + \delta(-, B[j]) \quad \text{("prima riga della matrice")}$$

for $i = 1$ to m **do**

$$D[i, 0] = D[i-1, 0] + \delta(A[i], -) \quad \text{("prima colonna della matrice")}$$

for $i = 1$ to m **do**

for $j = 1$ to n **do**

$$D[i, j] = \min\{D[i-1, j-1] + \delta(A[i], B[j]),$$
$$D[i-1, j] + \delta(A[i], -),$$
$$D[i, j-1] + \delta(-, B[j])\}$$

("formula ricorsiva")

return D[m,n]

Una volta costruita la matrice D, l'Edit Distance è data dall' elemento $D[m, n]$. L'algoritmo di progammazione dinamica possiede una complessità sia temporale che spaziale dell'ordine di $\mathcal{O}(mn)$; quindi si possono allineare due sequenze con un numero di operazioni dell'ordine di grandezza pari al prodotto delle lunghezze delle due stringhe.

In molte applicazioni biologiche due stringhe di DNA possono essere in generale molto differenti pur contenendo delle regioni altamente similari. Quando si confrontano proteine appartenenti a famiglie differenti e quindi globalmente diverse, spesso ci si trova di fronte ad un allineamento globale (cioè tra le intere sequenze) di scarso interesse, pur essendoci al loro interno delle regioni con un alto valore di similarità; accade spesso, cioè che possiedano le stesse subunità strutturali o funzionali. Quindi la similaritá locale tra due stringhe può diventare più significativa e conveniente di quella globale.

Definizione 9. *Date due stringhe* $\mathbf{s}$ *e* $\mathbf{t}$*, il problema della* **similaritá locale** *è definito come il problema di trovare le sottostringhe* α *e* β*, rispettivamente di* $\mathbf{s}$ *e* $\mathbf{t}$*, la cui similarità è massima tra tutte le coppie di sottosequenze.*

Per quantificare la similarità è preferibile utilizzare una misura di similarità, che definiamo in questo modo :

$$\delta(\mathbf{x},\mathbf{y}) = \begin{cases} \geq 0 & \text{se } \mathbf{x},\mathbf{y} \text{ sono similari} \\ < 0 & \text{se non lo sono, o se uno tra } \mathbf{x},\mathbf{y} \text{ è uguale ad un gap} \end{cases}$$

L'algoritmo di programmazione dinamica (Smith-Waterman [SW 81]), che risolve il problema della similarità è molto simile alla procedura utilizzata per l'allineamento globale, l'unica differenza è rappresentata dalla presenza del costo zero che viene introdotta per differenziare i gradi di similarità e non-similarità; quindi anche in questo caso la complessità è pari a $\mathcal{O}(mn)$.

3. Algoritmi euristici per allineamento di sequenze

Gli algoritmi di programmazione dinamica presentati nel capitolo precedente, sono in grado di fornire sempre la soluzione "ottima" del problema, tuttavia, data la loro complessità ($\mathcal{O}(mn)$) possono avere notevoli problemi di memoria e di efficienza computazionale. Quando si confronta una proteina con un intero database proteico o si cercano zone di similarità tra enormi tratti di D.N.A., si rende indispensabile l'utilizzo di strumenti informatici particolarmente efficienti, in grado cioè di calcolare una soluzione ammissibile in tempi ristretti e senza un eccessivo dispendio di memoria. Nel terzo

capitolo, si sviluppano e si implementano dei metodi basati su strategie euristiche, per l'allineamento globale e per la similarità locale tra sequenze di proteine o di acidi nucleici.

Inizialmente abbiamo utilizzato una tecnica di programmazione di tipo greedy [HS 84]; negli algoritmi greedy la soluzione viene determinata attraverso una sequenza di decisioni "localmente ottime", senza mai tornare indietro, modificandole, sulle decisioni prese: si parte cioè da uno stato iniziale, ad ogni passo si effettua la scelta migliore tra quelle possibili e si procede in questo modo fino a raggiungere lo stato finale. Adattando questa strategia al problema dell'allineamento tra sequenze, dobbiamo specificare cosa si intende per decisioni "localmente ottime", stabilire cioè quale sia un buon criterio di approssimazione per giungere ad una soluzione ammissibile; bisogna pertanto definire una *funzione obiettivo* che "stimi" il costo per raggiungere lo stato finale del problema, e poi muoverci verso il minimo di tale funzione.

Come funzione obiettivo consideriamo la funzione costo δ (3), che associa un peso ad ogni edit operation effettuata, mentre per minimizzarla sfruttiamo la relazione di dipendenza che si deduce dalla formula ricorsiva (4):

$$\begin{array}{ccc} \mathbf{d_{i-1,j-1}} & & \mathbf{d_{i-1,j}} \\ & \searrow & \downarrow \\ \mathbf{d_{i,j-1}} & \rightarrow & \mathbf{d_{i,j}} \end{array}$$

cioè ogni passo o elemento della matrice distanza, dipende da uno e uno solo di quei tre elementi che lo precedono; pertanto se consideriamo una coppia qualsiasi di caratteri delle due stringhe, per esempio quella individuata dalla coppia di indici (i, j), gli unici movimenti possibili sono:

- uno spostamento verticale ($\downarrow$): indica una operazione di *deletion* ed è individuato dalla seguente coppia di indici $(i + 1, j)$
- uno spostamento orizzontale ($\rightarrow$): indica un operazione di *insertion* ed è individuato dalla seguente coppia di indici $(i, j + 1)$
- uno spostamento diagonale ($\searrow$): indica un operazione di *substitution* o di *match* ed è individuato dalla seguente coppia di indici $(i+1, j+1)$

Per trovare l'allineamento basta quindi, ad ogni iterazione, calcolare il minimo tra i costi delle tre edit operation possibili ed a seconda di quale sia il risultato trovato, effettuare il relativo spostamento; tenendo presente ovviamente che il valore di ogni coppia di caratteri dipende dal valore del passo precedente a cui viene sommato il costo dell'operazione che è stata scelta per arrivare a tale coppia di caratteri; il ciclo iterativo termina non appena sono percorse interamente tutte e due le sequenze. In pratica ci siamo limitati a "ribaltare" la strategia utilizzata negli algoritmi di programmazione dinamica: invece di calcolare l'edit distance di due sequenze e poi determinare, andando a ritroso nella matrice distanza, il loro allineamento, calcoliamo direttamente uno dei possibili cammini (allineamenti) fino ad arrivare a determinare il valore dell'edit distance. In tal modo la complessità dell'algoritmo diventa dell'ordine di $\mathcal{O}(m+n)$.

Nel successivo algoritmo implementato, sempre utilizzando una strategia di tipo greedy, abbiamo introdotto un nuovo criterio di approssimizzazione (*la funzione euristica*), con l'obbiettivo di migliorare la qualità della soluzione, senza però compromettere l'efficienza computazionale. Pertanto la nuova *funzione di valutazione* [HS 84], che si deve minimizzare per giungere ad una soluzione del nostro problema di ottimizzazione, è costituita dalla somma di due funzioni:

$$\mathbf{f(n) = g(n) + h(n)} \tag{5}$$

dove :

- $\mathbf{g(n)}$: il costo che si produce nel raggiungere il nodo n dal nodo iniziale (costo del cammino percorso: *la funzione costo* δ).
- $\mathbf{h(n)}$: la stima del lavoro che si deve svolgere per raggiungere il nodo finale dal nodo corrente (stima della distanza tra il nodo finale ed n: *la funzione euristica*).

Lo schema iterativo della procedura è abbastanza semplice, infatti ad ogni iterazione si considerano i figli del nodo "corrente" (tre figli come le edit operation possibili) e per ognuno di essi viene calcolata la funzione di valutazione; il "figlio" che rende minima tale funzione viene scelto come nuovo

nodo corrente e si procede in questo modo fino a quando non sono lette interamente le due sequenze oggetto del confronto. Per il buon funzionamento di una strategia euristica è indispensabile scegliere delle "buone" funzioni euristiche, infatti una "cattiva" euristica può non solo diminuire l'efficienza computazionale dell'algoritmo ma anche pregiudicare l'ammissibilità della soluzione. Un euristica si rivela efficace quando è in grado di sfruttare le caratteristiche strutturali del problema in esame; i parametri che caratterizzano il problema dell'allineamento di sequenze proteiche, sono essenzialmente due:

- l'amminoacido, presente nella sequenza.
- l'ordine con cui gli amminoacidi si succedono all'interno delle sequenza.

Basandoci quindi, solo sul genere di ogni residuo e sulla posizione, che questi occupano all'interno della sequenza, abbiamo elaborato ed implementato 5 euristiche diverse:

euristica 1: osserva quale tipo di amminoacido è presente nelle due sequenze da confrontare; in un ciclo finale si assegna un costo di penalizzazione ogni qual volta si trova un amminoacido presente in una sola delle due sequenze. Questa euristica è scarsamente informata e di poca efficienza ed ha una complessità di $n + m$.

euristica 2: controlla se le sequenze hanno nella medesima posizione amminoacidi uguali; per farlo scorre le due sequenze simultaneamente e ad ogni coppia di residui diversi (*mismatch*) assegna un punteggio di penalizzazione in modo che sequenze simili abbiano un punteggio basso. La complessità è dell'ordine di $\mathcal{O}(\max(n, m))$.

euristica 3: molto simile all'euristica 2, osserva se gli amminoacidi che si trovano nella stessa posizione sono uguali o appartengono allo stesso gruppo di polarità o non-polarità; in caso di coppie di mismatch o di amminoacidi appartenenti a gruppi diversi viene assegnato un punteggio di penalizzazione. La complessità è dell'ordine di $\mathcal{O}(\max(n, m))$.

euristica 4: come nella prima fase dell'algoritmo FASTA [LP 88], si crea una tabella contenente tutte le posizioni per ciascun tipo di amminoacido presente nelle sequenze oggetto del confronto, in modo da

valutare in unico step la posizione di ciascun residuo. Successivamente viene eseguita la differenza matematica in modulo, dei valori posizionali degli amminoacidi dello stesso tipo tra le sequenze (*offset*); tale operazione comporta nel caso peggiore, una complessità spaziale e temporale dell'ordine di $\mathcal{O}(nm)$.

euristica 5: in questa funzione euristica ci limitiamo a sommare i costi (usando una matrice dei costi) dei k^2- confronti tra gli amminoacidi delle sequenze, dove k è un intero positivo scelto a priori; quindi una volta fissato k, se indichiamo con $X = x_1 \dots x_n$ e $Y = y_1 \dots y_m$ le due sequenze da confrontare e con δ la funzione costo, la funzione calcola:

$$\sum_{i=1}^{k}(\sum_{j=1}^{k}\delta(x_i, y_j))$$

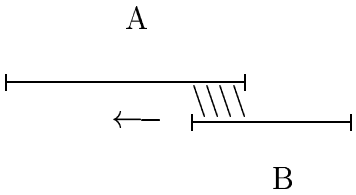
La complessità è pari a $\mathcal{O}(k^2)$

La complessità temporale dell'algoritmo è dell'ordine di $\mathcal{O}((n+m)k(n,m))$ dove $k(n,m)$ è il tempo di calcolo della funzione euristica scelta; la complessità spaziale è pari a $\mathcal{O}(n+m)$.

Il terzo ed ultimo algoritmo implementato si occupa del problema della similarità locale e della misura di similarità (9), quindi rientra in quel settore noto come "pattern matching" che comprende tutti quei problemi in cui bisogna ricercare particolari "stringhe pattern" all'interno di un testo assegnato. Per ridurre i problemi di complessità abbiamo adottato una strategia di tipo euristico; l'algoritmo è composto essenzialmente da tre fasi distinte:

1. Attraverso una tecnica di shifting scorriamo una sequenza sull'altra; in tal modo ci calcoliamo una specie di convoluzione, confrontando tutti gli elementi delle due sequenze.

   Esempio 1. Date due sequenze $A[1 \dots n]$, $B[1 \dots m]$ con $m < n$, in figura mostriamo come viene effettuato lo scorrimento di una sequenza sull'altra:



Utilizzando una matrice di sostituzione δ, assegnamo un costo, *punteggio di similarità*, ad ogni regione del confronto (in figura è quella denotata con "\\\\") che viene chiamata *regione di similarità*. Si prosegue in questo modo fino a che non abbiamo "shiftato" la sequenza B sull'intera sequenza A, cioè fino a quando non confrontiamo l'elemento $A[1]$ con l'elemento $B[m]$; Durante ogni step del ciclo la regione di similarità e il relativo punteggio vengono memorizzati in un vettore (complessità spaziale pari a $\mathcal{O}(n+m)$). la complessità temporale di questa prima fase è dell'ordine di $\mathcal{O}(nm)$.

2. Identifichiamo le k-regioni con il punteggio di similarità più alto (con k intero scelto a priori) e da ognuna di esse ricaviamo la sottoregione con la più alta densità di similarità. Le regole per determinare questi segmenti di similarità sono:

   - il segmento di similarità può avere al suo interno contributi rispetto alla matrice dei costi, sia positivi che negativi, l'importante è che il segmento si apra e si chiuda con un contributo positivo, cioè i due estremi del segmento devono corrispondere ad un confronto di caratteri simili tra loro.
   - Il punteggio di similarità della sottoregione individuata deve essere superiore ad un certo valore soglia (*treshold*) scelto a priori.

   La complessità temporale è dell'ordine di $\mathcal{O}(k(n+m))$, la complessità spaziale risulta pari a $\mathcal{O}(k)$.

3. Dopo aver ordinato le k-sottoregioni, controlliamo se possono essere collegate tra loro (*join*) per ottenere un allineamento approssimato con gap. I vincoli e le regole per la creazione del collegamento sono:

- esclusione di quelle sottoregioni con eventuali aree di overlapping (sovrapposizione) rispetto alla sequenza A e B
- introduzione di gap di costo zero per unire tra loro due o più sottoregioni
- il punteggio di similarità della regione che si viene a creare con l'unione di due o più sottoregioni, viene dato dalla somma dei punteggi di similarità delle singole sottoregioni che la compongono.

In questa fase sia la procedura che ordina le k-sottoregioni, sia quella che controlla la possibilità di eventuali collegamenti, eseguono un numero di operazioni e allocano memoria pari a k. La complessità totale dell'algoritmo è dunque determinata dalla prima fase.

4. Sperimentazione

Nel quarto capitolo, utilizzando dei database di sequenze proteiche, si mettono a confronto gli algoritmi da noi sviluppati, con quelli tradizionalmente usati in bioinformatica. Dopo aver selezionato un campione di sequenze da un database ($[SC]$), utilizzando gli algoritmi implementati, ($greedy$ e $greedy + euristiche$) abbiamo messo a confronto i risultati ottenuti con le soluzioni forniteci dall'algoritmo di programmazione dinamica di Needleman-Wunsch [NW 70] (pag viii). Abbiamo selezionato e confrontato proteine che appartengono a tre diverse categorie:

1. molto simili, con un range di similarità pari al $70 - 90\%$
2. medio simili, con un range di similarità pari al $40 - 60\%$
3. poco simili, con un range di similarità pari allo $0 - 30\%$

Osservazione 2. Data una coppia di sequenze, se indichiamo con $len =$ la lunghezza della sequenza più corta e con $Ed =$ la loro edit distance, il range di similarità viene così calcolato :

$$len/Ed$$

Nel grafico, sono state messe a confronto le medie dei rapporti di competitività nei test effettuati, utilizzando un modello dei costi sia unitario sia non unitario (matrice di sostituzione BLOSUM 62); nell'asse delle ascisse sono indicate le fasce di similarità con cui abbiamo classificato le coppie di sequenze utilizzate durante la sperimentazione.

Notazione. *rapporto di competitività*: rapporto tra l'edit distance dell'algoritmo di programmazione dinamica e l'edit distance calcolata con l' algoritmo euristico implementato.

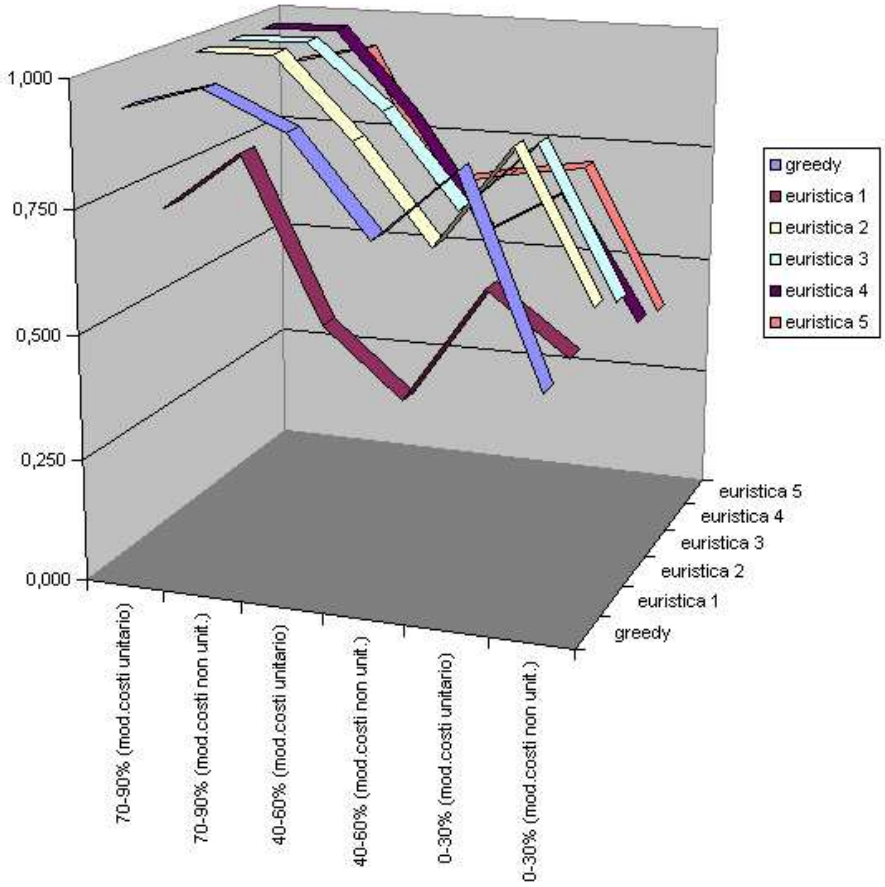


Figura 1: Rapporto di competitività

Nella tabella abbiamo inserito la media dei rapporti tra il tempo di calcolo dell'algoritmo euristico implementato e quello dell'algoritmo di programmazione dinamica:

range di similarità	greedy	eu. 1	eu. 2	eu. 3	eu. 4	eu. 5
$70-90\%$ costi unit.	$0,041$	$0,070$	$0,061$	$0,062$	$0,068$	$0,312$
$70-90\%$ BLOSUM 62	$0,051$	$0,105$	$0,928$	$0,716$	$0,116$	$0,451$
$40-60\%$ costi unit.	$0,041$	$0,085$	$0,083$	$0,079$	$0,095$	$0,352$
$40-60\%$ BLOSUM 62	$0,054$	$0,124$	$0,998$	$0,762$	$0,124$	$0,506$
$0-30\%$ costi unit.	$0,034$	$0,074$	$0,056$	$0,057$	$0,069$	$0,384$
$0-30\%$ BLOSUM 62	$0,040$	$0,077$	$0,994$	$0,729$	$0,104$	$0,360$

Tabella 1: rapporto tra i tempi di calcolo (in secondi)

5. Conclusioni

Dalla figura (1) si evince che l'insieme degli algoritmi implementati mostrano un comportamento differente al variare della classe di similarità, a cui appartengono le proteine oggetto del confronto:

- quando le coppie di sequenze confrontate sono molto simili (range di similartà intorno al $70-90\%$), gli algoritmi euristici, tranne l'euristica 1, forniscono delle soluzioni che rientrano in un range del 10% rispetto alla soluzione ottima. Quindi oltre ad essere "veloci" da un punto di vista computazionale (v. tab. 1), questi algoritmi forniscono delle soluzioni ammissibili poco distanti dall'ottimo.

- in presenza di sequenze poco simili o medio simili (range di similarità pari allo $0-30\%$ o al $40-60\%$) gli algoritmi euristici implementati si rivelano efficienti per quanto riguarda la complessità temporale e spaziale ma poco "precisi"; le soluzioni sono infatti molto distanti dall'ottimo (distanza dall'ottimo che oscilla intorno al $20-50\%$).

Per quanto riguarda l'algoritmo euristico implementato che ricerca zone di similarità all'interno di sequenze, possiamo concludere che se confrontato con l'algoritmo di programmazione dinamica, si rivela piuttosto efficiente e innovativo; un limite dell'algoritmo è rappresentato dalla complessità temporale che risulta dell'ordine di $\mathcal{O}(nm)$; Suggeriamo come idea per ridurre la complessità e renderla dell'ordine di $\mathcal{O}(n+m)$, quella di servirsi delle strategie implementate per le problematiche di pattern matching; può ad esempio rivelarsi interessante costruire degli alberi di suffissi di una stringa (*suffix tree* [LA 99]).

Appendice - FASTA

In appendice vengono presentati gli algoritmi, basati su una strategia euristica più comunemente usati in bioinformatica (*FASTA* [LP 88], *BLAST* [Alt 90]).

Bibliografia

[Alt 90] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, D. J. Lipman, *A basic local alignment search tool*, Journal of Molecular Biology **215** pagg 403-410 (1990).

[BO 98] A. D. Baxevanis, B. F. F. Ouellette, *Bioinformatics: A Pratical Guide to the Analysis of Genes and Proteins*, Wiley&Sons, (1998)

[BT 91] C. Branden, J. Tooze, *Introduction to Protein Structure*, Garland Publishing Inc., New York (1991).

[Day 78] M. O. Dahyoff, *Atlas of protein sequence and structure*, National Biomedical Research Fundation Vol. **5** Suppl.3 (1978) pagg 345-352.

[Gus 97] Dan Gusfield, *Algorithms on strings, trees and sequences*, Cambridge University Press , New York (1997).

[GJ] M. R. Garey & D. S. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness*, W.H. Freeman & Company (1979) .

[HH 92] S. Henikoff, J. Henikoff , *Amino acid substitution matrices from protein blocks* , PNAS Classification:Biochemistry **89** (1992) pagg 10915-10919.

[HL 99] F. S. Hillier, G. S. Lieberman *Introduzione alla ricerca operativa*, Franco Angeli, Milano (1999).

[HS 84] E. Horowitz, S. Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press, Inc. cap 4, 5, 8 (1984).

[LA 99] N. J. Larsson, *Structures of String Matching and Data Compression*, Department of Computer Science Lund University, pagg 9-63 (1999)

[LP 85] D. J. Lipman, W. R. Pearson, *Rapid and sensitive protein similarity searches*, Science, **227** pagg 1435-1441 (1985).

[LP 88] D. J. Lipman, W. R. Pearson, *Improved tools for biological sequence comparison*, Proceedings of the National Academy of Science USA, **85** pagg 2444-2448 (1988).

[LW 95] E. S. Lander, M. S. Waterman *Calculating the Secrets of Life*, National Academy Press (1995).

[NW 70] S. B. Needleman, C. D. Wunsch *A general method applicable to the search for similarities in the amino acid sequence of two proteins*, Journal of Molecular Biology **48** pagg. 443-453 (1970).

[Pea 90] W. R. Pearson *Rapid and sensitive sequence comparison with FASTP and FASTA*, Methods in Enzymology, **183** pagg 63-98. Academic Press, San Diego (1990).

[RB 57] R. Bellman, *Dynamic programming*, Princeton University Press, (1957).

[SW 81] T. F. Smith, M. S. Waterman *Identification of common molecular subsequences*, Journal of Molecular Biology, **147(1)** pagg 195-197 (1981).

[TO 89] W. R. Taylor, C. A. Orengo, *Protein structure alignment*, Journal of Molecular Biology, **208** pagg 1-22 (1989).

[Wat 95] M. S. Waterman *Introduction to Computational Biology: Maps, Sequences and Genomes*, Chapman and Hall, (1995).

[WL 83] W. J. Wilbur, D. J. Lipman, *Rapid similarity searches of nucleic acid and protein data banks*, Proc. Natl. Acad. Sci. USA, **80** pagg 726-730 (1983).

Siti Web

- [*KRT*] R. Karp, L. Ruzzo, M. Tompa, *Algorithms in molecular biology*
  www.cs.washington.edu/education/courses/590bi/96w/
- NCBI- National Center for biotechnology information
  www.ncbi.nih.gov
- W.R. Pearson, *FASTA Man Pages*
  http:eos.bio.unipd.it/fasta_man.html
- [*SC*] Structural classification of proteins
  http://scop.mrc-lmb.cam.ac.uk/scop/data/scop.b.html
- [*GW*] R. Giegerich, D. Wheeler, *Pairwise Sequence Alignment*
  www.techfak.uni-bielefeld.de/bcd/Curric/PrwAli/prwali.html
- [*Sog*] L. Soggin, *Fasta*
  www.fog-bio.unipd.it/bioinfor/fasta.html
- [*RS*] Ron Shamir, *Algorithms for molecular biology: course archive*
  www.math.tau.ac.il/ ~rshamir/algmb.html