Fabrizio Luccio. Informatica per le Biotecnologie

8 IL PROBLEMA DEL FRAGMENT ASSEMBLY

8.1 Motivazioni e esempi elementari

In questo capitolo considereremo esplicitamente tratti di DNA costituiti come noto da sequenze di *basi* di quattro tipi: A (adenina), G (guanina), C (citosina), T (timina), dove A e G appartengono alla famiglia delle *purine*, C e T a quella delle *pirimidine.* In tali sequenze si riconosce un orientamento dovuto alla struttura della molecola di DNA, sicché possiamo rappresentarle e esaminarle con direzione da sinistra a destra. L'intera molecola di DNA è costituita da due **filamenti** (strands) accoppiati tra loro e che, con un'opportuna interpretazione, recano la stessa informazione: ci riferiamo dunque alla sequenza che rappresenta uno di questi filamenti.

Dal punto di vista algoritmico tutto quello che studieremo potrebbe essere riferito a sequenze costruite su un alfabeto arbitrario e afferenti ad ambienti diversi. Tuttavia la motivazione del capitolo è indicare alcune linee di studio che nascono nella risoluzione di un importantissimo problema di biologia molecolare, consistente nella ricostruzione di un'intera sequenza di DNA sulla base della conoscenza di alcune (in effetti moltissime) sue parti dette **frammenti** (fragnments). Il problema nel complesso è detto **sequenziamento**, e la tecnica è basata sulla composizione dei frammenti. Questa tecnica è motivata dal fatto che gli esperimenti che consentono di *leggere* le basi di filamenti di DNA non sono in grado di operare su un'intera sequenza se questa contiene un numero di basi superiore a qualche migliaio. E anche se lo fosse, è tecnicamente impossibile oggi rendere disponibile all'esame un intero filamento di DNA senza che questo si rompa in un grandissimo numero di frammenti. La tecnologia sta compiendo grandissimi passi avanti, ma anche per organismi la cui intera sequenza di DNA è relativamente breve, non è possibile leggere tale sequenza in una sola passata. Abbiamo dunque il problema:

FRAGMENT ASSEMBLY. Dato un insieme di frammenti di una sequenza, tra loro indipendenti (cioè derivanti da parti distinte della sequenza) o parzialmente sovrapponibili (cioè derivanti da parti parzialmente sovrapposte), ricostruire la sequenza originale con il minimo numero di errori.

Nel caso del DNA i frammenti devono essere moltissimi, ma è possibile ottenerli in breve tempo in quanto una molecola di DNA può essere replicata in laboratorio in un numero di copie che **cresce esponenzialmente nel tempo** con una tecnica detta **PCR** (polymerase chain reaction).

Una semplice riflessione preliminare può convincerci che è l'esistenza di frammenti parzialmente sovrapposti che permette la ricostruzione della sequenza complessiva: se infatti questa fosse

suddivisa in frammenti tutti separati, come avverrebbe se si partisse da un'unica sua copia, non si avrebbe modo di stabilire in che ordine essi debbano essere assemblati.

Iniziamo a esaminare il problema su un esempio elementare, ovviamente costruito su sequenze brevissime per poterlo discutere. Immaginiamo di avere a disposizione i quattro frammenti:

T C C G A C G A C T A A T C A T C C G A

e di sapere che la sequenza complessiva detta **di consenso** deve contenere circa dieci basi, quindi i frammenti devono essere parzialmente sovrapposti perché complessivamente ne contengono venti. In questo caso esiste un allineamento in cui le parti contengono sotto-sequenze identiche che non danno luogo a errori. Tale allineamento, e la corrispondente sequenza di consenso di nove basi, sono i seguenti:

```
- - T C C G A - -
- - - - C G A C T
A A T C - - - - -
- A T C C G A - -
_________________

A A T C C G A C T
```

Questo è un caso ideale perché gli errori sono in genere inevitabili e scaturiscono sia dagli strumenti di lettura dei frammenti che dalla imprecisa replicazione della sequenza originale nella PCR. Vediamo nel nostro esempio come questi errori possano emergere e condizionare l'assemblaggio.

• Se il secondo frammento fosse C C A C T (C anziché G nella seconda posizione) l'allineamento e la sequenza di consenso sarebbero gli stessi di prima, con un errore di sostituzione sul secondo frammento, nella sesta posizione dell'allineamento dove la presenza di due G e una C induce a scegliere G a maggioranza.

• Se il quarto frammento fosse A T C G A (perdita del terzo o quarto carattere) potremmo costruire l'allineamento:

```
- - T C C G A - -
- - - - C G A C T
A A T C - - - - -
- A T C - G A - -
_________________

A A T C C G A C T
```

che dà luogo alla solita sequenza di consenso con un errore di cancellazione sul quarto frammento, nella quinta posizione

dell'allineamento dove la presenza di due C e uno spazio induce a scegliere C a maggioranza.

• Se il secondo frammento fosse C T G A C T (inserzione di T come secondo carattere) potremmo costruire l'allineamento:

```
- - T C C - G A - -
- - - - C T G A C T
A A T C - - - - - -
- A T C C - G A - -
_____________________

A A T C C - G A C T
```

che dà luogo a una sequenza di consenso di dieci caratteri contenente uno spazio, con un errore di cancellazione sul quarto frammento, nella sesta posizione dell'allineamento dove la presenza di due spazi e una T induce a scegliere lo spazio a maggioranza.

Vi sono poi casi più complicati cui accenneremo brevemente senza dare qui ragguagli sui metodi impiegati per affrontarli.

• Nell'insieme dei frammenti generati è possibile che due di essi, provenienti da parti distanti della sequenza originale e parzialmente sovrapponibili, si connettano tra loro dando luogo a un nuovo frammento detto **chimera**. Chiaramente le chimere non sono parte della sequenza cercata e bisogna escluderle durante la costruzione di questa.

• La sequenza originale può avere tratti (approssimativamente) ripetuti due o più volte, detti **repeat**. Per esempio la sequenza può essere del tipo:

$$T_1\ ABC\ T_2\ ABC\ T_3\ ABC\ T_4$$

ove T_1, T_2, T_3, T_4 sono tratti diversi, e ABC è un repeat composto da tre sotto-sequenze A, B, C. Se tra i frammenti sono presenti:

F_1 = C T_2 A che lega il primo e il secondo repeat con T_2

F_2 = C T_3 A che lega il secondo e il terzo repeat con T_3

la sequenza di consenso può risultare indifferentemente T_1 R T_2 R T_3 R T_4 oppure T_1 R T_3 R T_2 R T_4 di cui solo la prima è corretta.

• Infine una **mancanza di copertura** si verifica quando mancano i frammenti relativi a un tratto della sequenza originale. Ovviamente in questo caso si ricostruiscono solo porzioni disgiunte della sequenza, come è evidente nel risultato. L'unico rimedio è ripetere la frammentazione su un maggior numero di copie della sequenza, tenendo presente che i costi di maggiore duplicazione con la PCR,

di sequenziamento in laboratorio, e di successivo calcolo dell'assemblaggio nel computer, crescono col numero di frammenti; ma che, d'altra parte, oltre il completamento della copertura, migliora l'affidabilità dell'intera sequenza di consenso.

8.2 L'algoritmo di base

Come già detto, l'assemblaggio di sequenze è oggi il problema più importante di algoritmica rivolta alla biologia molecolare. Un'impressionante quantità di risorse di calcolo è dedicata alla sua soluzione su sequenze genomiche di moltissimi organismi incluso l'uomo. Questi studi hanno consentito di ricavare un'enorme mole di dati genomici conservati in diversi Data Base di libero accesso. Una caratteristica molto positiva di questi è che i dati sono presentati in forme sostanzialmente compatibili tra loro rendendone agevole e rapido l'accesso e la fruizione.

La discussione del paragrafo precedente ha mostrato come ricostruire una sequenza da un grandissimo numero di frammenti non può essere un compito semplice data l'assenza di ogni informazione sulle zone della sequenza originale cui i frammenti si riferiscano, la parziale sovrapposizione tra loro e l'inevitabile presenza di errori. Si ricordi in proposito che il progetto per la ricostruzione del genoma umano richiese molti anni e uno sforzo considerevole da parte di qualificatissimi centri che disponevano di grandi risorse; e se da allora le apparecchiature sia di laboratorio che di calcolo hanno fatto enormi progressi, sequenziare un genoma non ancora noto è un compito considerevole.

In queste note dobbiamo limitarci a presentare un algoritmo di base, cuore dell'intero processo: si tratta di un'estensione degli algoritmi di GLOBAL COMPARISON e LOCAL COMPARISON riportati nella dispensa 7, anch'essi di grande rilevanza nella biologia, che impiegano il concetto di similarità. A scopo di presentazione adotteremo di nuovo i pesi +1, -1, e -1 rispettivamente per match, mismatch e carattere-spazio rimandando al prossimo paragrafo per qualche considerazione in merito. Poniamo:

SUFFIX-PREFIX COMPARISON. Date due sequenze X,Y trovare un suffisso S_X di X e un prefisso P_y di Y che presentino la **massima similarità.**

Dovrebbe essere immediatamente chiaro come questo problema intervenga nel Fragment Assembly, perché due frammenti si compongono in un unico frammento più grande sovrapponendo una parte finale (suffisso) di uno a una parte iniziale (prefisso) del secondo. Nel caso presente non si conosce a priori la lunghezza delle sotto-sequenze S_X e P_y cercate, che diverranno note solo come risultato dell'algoritmo. In particolare non si conosce il punto dove inizierà S_X in X né il numero di caratteri che conterrà la zona di sovrapposizione perché vi potranno essere spazi inseriti in

S_X e P_y. Ancora una volta impiegheremo la programmazione dinamica operando sulla matrice M con qualche variazione.

L'inizializzazione della riga e della colonna zero sarà:

M[Ø,j]=-j, per Ø≤j≤m; M[i,Ø]=Ø, per Ø≤i≤n

ove i valori M[Ø,j]=-j nella prima riga hanno il consueto significato e i valori M[i,Ø]=Ø nella prima colonna sono dovuti al fatto che S_X può iniziare da qualunque posizione di X. La formula ricorsiva che guida il calcolo di M[i,j] sarà la consueta:

M[i,j] = max{M[i,j-1]-1, M[i-1,j]-1, M[i-1,j-1]+p(i,j)}

con p(i,j) uguale a +1 (match) o -1 (mismatch), mentre il valore contenuto nella cella M[i,j] indica la massima similarità tra due sotto-sequenze di X, Y, in particolare $x_h \ldots x_i$ di X per un opportuno valore $1 \le h \le i$, e $y_1 \ldots y_j$ di Y, che terminano in i,j e che nell'allineamento possono contenere degli spazi.

I risultati dovranno ora essere cercati nell'ultima riga della matrice, ove in una cella M[n,j] l'esame di un suffisso S_X è stato completato paragonandolo al prefisso P_y fino alla colonna j. L'interpretazione di questi risultati è nuova rispetto a quanto visto fin qui, come spiegheremo ora riferendoci al seguente esempio.

	Ø	T	A	A	C	G	T	G	A	A	C
Ø	Ø	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10
G	Ø	-1	-2	-3	-4	-3	-4	-5	-6	-7	-8
A	Ø	-1	Ø	-1	-2	-3	-4	-5	-4	-5	-6
T	Ø	+1	Ø	-1	-2	-3	-2	-3	-4	-5	-6
C	Ø	Ø	Ø	-1	Ø	-1	-2	-3	-4	-5	-4
A	Ø	-1	+1	+1	Ø	-1	-2	-3	-2	-3	-4
A	Ø	-1	Ø	+2	+1	Ø	-1	-2	-2	-1	-2
G	Ø	-1	-1	+1	+1	+2	+1	Ø	-1	-2	-2
C	Ø	-1	-2	Ø	+2	+1	+1	Ø	-1	-2	-1
T	Ø	+1	Ø	-1	+1	+1	+2	+1	Ø	-1	-2
G	Ø	Ø	Ø	-1	Ø	+2	+1	+3	+2	+1	Ø

Come detto, ogni valore M[n,j] nell'ultima riga della matrice corrisponde a uno o più allineamenti di massima similarità tra il prefisso di Y che termina in y_j e un opportuno suffisso di X. Per conoscere l'elemento x_h in cui comincia questo suffisso si deve costruire l'allineamento con l' algoritmo noto sulla matrice M, risalendo in senso inverso nel percorso di calcolo fino a raggiungere la colonna zero nella riga ove il suffisso ha inizio (con eventuali spazi iniziali) e l'algoritmo termina. Nel nostro esempio il valore di similarità M[10,7] = +3 corrisponde ai due allineamenti:

```
Y:     T - A A C G - T G A A C
X: G A T C A A - G C T G

Y:     T - A A - C G T G A A C
X: G A T C A A G C - T G
```

che partono dalla riga 3 dando entrambe luogo a una sovrapposizione di lunghezza nove, tra sette caratteri e due spazi in Y, e otto caratteri e uno spazio in X. Il valore di similarità M[10,5] = +2 corrisponde all'allineamento:

```
Y:     T - A A - C - G T G A A C
X: G A T C A A G C T G
```

che parte dalla riga 3 dando luogo a una sovrapposizione di lunghezza otto, tra cinque caratteri e tre spazi in Y, e otto caratteri in X. Infine il valore di similarità M[10,8] = +2 corrisponde all'inserzione di uno spazio alla fine di S_x accoppiato al carattere A in P_y, scartata perché non vi è alcun vantaggio a protrarre con uno spazio la porzione finale di X allineata a Y.

Una sovrapposizione utilizzabile nel fragment assembly deve avere lunghezza *k* e similarità *s* relativamente alte. Il criterio di accettazione, o di scelta tra varie possibilità, è stabilito dal biologo. Nel nostro esempio sono ovviamente preferibili i due primi allineamenti ove le sovrapposizioni hanno maggiore lunghezza e similarità del terzo, ma può essere necessario un confronto tra due sovrapposizioni con valori (k_1,s_1) e (k_2,s_2) tali $k_1>k_2$ e $s_1<s_2$, caso che qui non appare. Deve comunque essere chiaro che negli esperimenti reali i frammenti possono contenere migliaia di caratteri e le sovrapposizioni accettabili tra frammenti devono avere lunghezza fino a centinaia di basi con numero massimo di differenze di qualche unità; e che, per esempio nelle sequenze geniche degli eucarioti, vi sono zone più *conservate* di altre su cui sono tollerabili meno errori.

8.3 Qualche altra considerazione

Il problema dei pesi. Nel paragrafo precedente abbiamo adottato i pesi +1, -1, e -1 per match, mismatch e carattere-spazio. Tuttavia, in dipendenza dal problema trattato, questi pesi possono non essere i più significativi e deve essere il biologo a indicare i pesi da scegliere, notando che non sono importanti i loro valori assoluti ma la differenza tra di essi, perché è questa differenza a influenzare il risultato dell'algoritmo di confronto.

I punti particolarmente rilevanti sono il peso associato alla corrispondenza carattere-spazio rispetto al mismatch (che finora abbiamo posto entrambi uguali a -1), e il peso di differenti situazioni di mismatch. Infatti una sostituzione tra purine (A e G) o tra pirimidine (T e C) è in genere più probabile che una sostituzione tra una purina e una pirimidina e il peso negativo del mismatch dovrebbe essere maggiore in valore assoluto nel secondo caso. Una regola ragionevole per i pesi di match e mismatch è espressa nella seguente tabella che non dovrebbe richiedere spiegazioni, ma ancora una volta sarà il biologo a stabilire i valori da inserirvi in relazione al problema trattato:

	A	T	C	G
A	+2	-2	-2	-1
T	-2	+2	-1	-2
C	-2	-1	+2	-2
G	-1	-2	-2	+2

Individuazione di un gene in una sequenza di DNA. Un altro problema assai importante riguarda la ricerca di un gene, di cui si è determinata la sequenza S, all'interno di una sequenza nota T del DNA di un intero organismo. Scopo dell'operazione è per esempio stabilire se la S compare (approssimativamente) come sotto-sequenza U di T, o controllare quali mutazioni presenta la S rispetto alla U. Questo è ovviamente lo stesso problema di individuare un pattern (S) in un testo (T) già trattato nella dispensa 7, cui ora si applicheranno pesi significativi per il problema biologico.

Però, se per esempio la S ha una lunghezza $|S|$ di migliaia di basi perché rappresenta il gene di un eucariote che contiene esoni e introni, e la T ha lunghezza $|T|$ di centinaia di milioni di basi perché è la sequenza di un cromosoma umano, l'algoritmo quadratico di programmazione dinamica richiede un numero di operazioni di ordine $|S|\cdot|T|$ che è elevatissimo anche se polinomiale. Il meccanismo di ricerca si può organizzare allora in un modo differente che spiegheremo senza presentare l'algoritmo relativo che è piuttosto complicato.

Come sappiamo il risultato che S è contenuta in T viene accettato se il numero di errori *e* nell'allineamento ottimo non supera un limite prefissato, poniamo *e* = 49 errori in totale. Se un tale allineamento esiste, dividendo S in *e* + 1 = 50 tratti consecutivi di |S|/50 basi ciascuno, almeno uno di questi tratti deve essere privo di errori: si noti che ne esiste uno solo nel caso in cui gli errori siano ugualmente distanziati, altrimenti possono esistere molti tratti senza errori.

Nel ricco mondo degli algoritmi classici su sequenze ne esiste uno detto **KMP** (dalle iniziali di Knuth, Morris e Pratt che lo hanno proposto) che permette di determinare la presenza **esatta** (cioè senza errori) di un pattern S in un testo T in tempo di ordine |T|+|S|, lineare nella dimensione dell'input. Per tornare al nostro esempio questo ordine è |T|, perché |T| è molto maggiore di |S|. Affrontiamo dunque il problema applicando 50 volte l'algoritmo KMP per ricercare senza errori in T ognuno dei tratti di S. Due risultati sono possibili:

1) nessuno di questi tratti appare in T, dunque $ non appare in T senza superare il numero massimo di errori richiesto;

2) uno di questi tratti, diciamo W, appare in T: dunque la zona Z di T dove dovrebbe trovarsi la S si sviluppa attorno a W e ha lunghezza di ordine |S| (può essere maggiore per la presenza di basi in T che non appartengono a S e danno luogo a spazi in S nell'allineamento). Si confrontano quindi S e Z con l'algoritmo di programmazione dinamica.

Il tempo richiesto dal caso 1) è di ordine *e*|T| per la ricerca degli *e*+1 tratti in T. Il tempo richiesto dal caso 2) è anch'esso di ordine *e*|T| per la stessa ricerca (anche se questa si ferma appena si trova W) più il tempo $|S|^2$ richiesto dall'algoritmo di programmazione dinamica applicato a due sequenze lunghe circa |S|; complessivamente il tempo rimane di ordine *e*|T| se $|T|>|S|^2$ come nel nostro esempio. L'applicazione diretta dell'algoritmo di programmazione dinamica a S e T avrebbe invece richiesto un tempo |S|•|T| >> *e*|T| (tipicamente 100 volte superiore nel nostro esempio).

Determinazione della sequenza di DNA di un organismo. La tecnica di confronto approssimato attraverso la ricerca esatta di un pattern in un testo è anche utilizzata, con metodo praticamente identico a quanto appena visto, quando si tratta di sequenziare l'intero DNA di un organismo di cui già si conosce il genoma di riferimento memorizzato in una base di dati: per esempio sequenziare il genoma **G** di un uomo particolare, che avrà circa 1 milione di basi sconcordanti con quelli del genoma umano **R** di riferimento sequenziato a suo tempo (questo è l'ordine di grandezza medio del

numero di differenze: 1 Mega su un totale di oltre 3 Giga basi, cioè circa 0.3 millesimi).

Partendo da un insieme di frammenti di G ottenuti come per il suo sequenziamento, anziché assemblarli come visto sopra si cercano questi frammenti in R col metodo detto sopra per stabilire in che zona di G si dovranno collocare e quindi con quali altri frammenti collegarli. Il procedimento è naturalmente molto più complicato di così, ma l'idea permette di capire come sequenziare un genoma ex novo con questo metodo richieda un numero di operazioni estremamente inferiore a quelle richieste da un completo fragment assemby.

Qualche considerazione conclusiva. Quanto discusso sopra ci permette di fare una considerazione generale sulle proprietà degli algoritmi, a commento di quanto esposto in tutte queste dispense. L'apparizione esatta di un pattern in un testo può essere determinata sia con l'algoritmo KMP, sia con quello di programmazione dinamica per l'apparizione approssimata: infatti in questo caso le apparizioni esatte del pattern corrispondono alle celle della matrice che riportano un errore zero. Il primo algoritmo è molto più complicato ma molto più efficiente del secondo, e mostra che il problema è (in gergo) *molto facile* perché ha complessità lineare, che è anche un limite inferiore in quanto è proporzionale al tempo necessario a esaminare tutti i dati (quindi KMP è un algoritmo *ottimo*). Nel gergo degli algoritmi la *facilità* di un problema è legata alla sua natura che gli permette di risolverlo in modo efficiente, non alla facilità con cui si scopre e si progetta un algoritmo di soluzione.

Esistono infine molti problemi su sequenze biologiche che appartengono alla classe dei problemi **NP-completi**, per la cui soluzione si deve inevitabilmente ricorrere a algoritmi approssimati. Questi studi non trovano spazio in queste note: per essi rimandiamo a qualsiasi buon teso di Biologia Computazionale.