

8-1 IL PROBLEMA DEL FRAGMENT ASSEMBLY

Motivazioni ed 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** (fragments). 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:

T₁ ABC T₂ ABC T₃ ABC T₄

ove T₁, T₂, T₃, T₄ sono tratti diversi, e ABC è un repeat composto da tre sotto-sequenze A, B, C. Se tra i frammenti sono presenti:

F₁ = C T₂ A che lega il primo e il secondo repeat con T₂

F₂ = C T₃ A che lega il secondo e il terzo repeat con T₃

la sequenza di consenso può risultare indifferentemente T₁ R T₂ R T₃ R T₄ oppure T₁ R T₃ R T₂ R T₄ 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.

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:

$$M[\emptyset, j] = -j, \text{ per } \emptyset \leq j \leq m; \quad M[i, \emptyset] = \emptyset, \text{ per } \emptyset \leq i \leq n$$

ove i valori $M[\emptyset, j] = -j$ nella prima riga hanno il consueto significato e i valori $M[i, \emptyset] = \emptyset$ 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 $x_h \dots x_i$ di X per un opportuno valore $1 \leq h \leq i$, e $y_1 \dots y_j$ 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.

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

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.