Algorithmes pour la biologie génomique

1. Alignement de deux séquences: programmation dynamique
2. Techniques pour diminuer l'espace mémoire utilisé
3. Alignement de plusieurs séquences est NP-complet
4. Algorithme approché
5. Technique de construction d'arbres d'évolution

Distance entre deux séquences

- On veut déterminer si des suites de A,C,G, T sont similaires, ou bien très différentes.
- Il faut définir une distance entre ces suites
- Une possibilité: la distance d'édition ou nombre d'opérations pour passer de l'une à l'autre
- Les biologistes préfèrent la notion de meilleur alignement
- Les techniques de calcul sont identiques

Alignements de séquences

Pour deux mots sur l'alphabet $\{A, C, G, T\}$ on cherche un alignement qui optimise une fonction qui mesure la similarité

Exemple: GATGCAT et $ATCGAT$

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

Distance entre lettres

- On se donne un alphabet A et une matrice symétrique Δ représentant des distances entre les couples de lettres
- On suppose que $\Delta[x, y] = 0$ si et seulement si $x = y$ et que la matrice satisfait l'inégalité triangulaire:

$$\Delta[x, y] \leq \Delta[x, z] + \Delta[z, y]$$

- La distance brute δ' entre deux mots f et g de même longueur est égale à la somme des distances entre les lettres qui les composent, soit:

$$\delta'(f, g) = \sum_{i=1}^{m} \Delta(f_i, g_i)$$

- La distance brute pour des mots de même longueur satisfait trivialement les axiomes des espaces métriques

Distance entre deux mots

- On ajoute un symbole supplémentaire - à l'alphabet que l'on interprète comme un espacement, et on définit une distance constante $\Delta[-, x]$ entre chaque lettre x et cet espacement.
- Un alignement entre deux mots f et g sur l'alphabet A de longueurs quelconques est constitué de deux mots de même longueur f' et g' sur l'alphabet $A \cup \{-\}$ tels que f et g s'obtiennent à partir de f' et g' en supprimant les espacements -
- Le **score** de l'alignement est la distance brute entre f' et g'
- La distance entre deux mots est alors le score minimal pour un alignement entre f et g

Un exemple:

Δ	A	C	G	T	$-$
A	0	2	1	2	3
C	2	0	2	1	3
G	1	2	0	2	3
T	2	1	2	0	3
$-$	3	3	3	3	0

Algorithme de calcul du meilleur alignement

- $$\delta(\varepsilon, f) = \sum_{i=0}^{n} \Delta[-, f_i]$$

- Pour chaque couple (fa, gb) on a le choix entre trois possibilités:
  1. Aligner a avec b et f avec g
  2. Aligner a avec $-$ et f avec gb
  3. Aligner b avec $-$ et fa avec g
- Il faut choisir celle des trois qui minimise la valeur

$$\delta(fa, gb) = \min \begin{cases} \delta(f, g) + \Delta[a, b] \\ \delta(fa, g) + \Delta[-, b] \\ \delta(f, gb) + \Delta[a, -] \end{cases}$$

		A	T	G	C	G	G	C	A	T	G
	0	3	6	9	12	15	18	21	24	27	30
G	3	1	4	6	9	12	15	18	21	24	27
A	6	3	3	5	8	10	13	16	18	21	24
T	9	6	3	5	6	9	12	14	17	18	21
G	12	9	6	3	6	6	9	12	15	18	18
G	15	12	9	6	5	6	6	9	12	15	18
C	18	15	12	9	6	7	8	6	9	12	15
T	21	18	15	12	9	8	9	9	8	9	12
A	24	21	18	15	12	10	9	11	9	10	10

Principe de l'algorithme de programmation dynamique

- Le calcul de la distance minimale se fait en temps $O(mn)$
- Le problème est de retrouver l'alignement une fois que l'on a la distance
- une méthode consiste à parcourir le tableau en partant de la case sud-est et en remontant au nord à l'ouest ou au nord-ouest
- Une difficulté: l'espace mémoire est quadratique!

Principe d'un algorithme en espace linéaire

- Le calcul des distances peut se faire sur une seule ligne puisque la valeur d'une case ne dépend que de celle à gauche et celle plus haut (donc l'ancienne valeur) si on décide de n'utiliser qu'une ligne
- Le problème est de retrouver l'alignement une fois que l'on a le score
- On procède par divide and conquer

Algorithme en espace linéaire deux fois plus lent

- Pour aligner u et v on prend i comme la moitié de la longueur de u
- On calcule le score a_j du meilleur alignement de $u_1u_2 \ldots u_{i-1}$ avec chaque facteur gauche $v_1v_2 \ldots v_j$ de v
- C'est l'algorithme de programmation dynamique précédent en gardant la dernière ligne
- On calcule le score b_k du meilleur alignement de $u_{i+1}u_{i+2} \ldots u_n$ avec chaque facteur droit $v_kv_{k+1} \ldots v_n$ de v
- C'est l'algorithme de programmation dynamique précédent modifié: commencer par la droite de v

Algorithme en espace linéaire deux fois plus lent

- On cherche le $a_{j-1} + b_{j+1} + \Delta[u_i, v_j]$ le meilleur
- On recommence avec les couples $u_1 u_2 \ldots u_{i-1}, v_1, v_2 \ldots v_{j-1}$ et $u_{i+1} u_{i+2} \ldots u_n, v_{j+1}, v_{j+2} \ldots v_m$

Calcul de la complexité de l'algorithme en espace linéaire

- On montre que le nombre de comparaison $C(m,n) \leq 2mn$
- Vrai pour $m = 1$
- $$C(m,n) \leq \frac{mn}{2} + \frac{mn}{2} + C(m/2, j) + C(m/2, n-j)$$
- $$C(m,n) \leq \frac{mn}{2} + \frac{mn}{2} + mj + m(n-j) \quad \leq 2mn$$

Alignement de plusieurs séquences

- On se donne des séquences $f_1, f_2, \ldots f_k$
- On cherche des mots f'_i tous de même longueur, obtenus à partir des f_i par ajouts de la lettre $-$
- Tels que

$$\sum_{1 \leq i < j \leq n} d(f'_i, f'_j)$$

- soit minimale

Alignement de plusieurs séquences est NP-complet

Etant donné un entier k, une famille $f_1, \ldots, f_k$ de de mots , une fonction de distance Δ sur les lettres , et un entier C, décider s'il existe un alignement de valeur au plus C est un problème NP-complet.

On réduit RECOUVREMENT PAR SOMMETS à ALIGNEMENT DE SÉQUENCES

On utilise l'alphabet $\{a, b, 0, 1, c, d\}$ et la distance Δ suivante

	a	b	0	1	c	d	–
a	0	1	1	1	1	2	2
b	1	0	1	1	2	1	2
0	1	1	0	2	2	2	2
1	1	1	2	0	2	1	2
c	1	2	2	2	0	2	1
d	2	1	2	1	2	0	2
–	2	2	2	2	1	2	5

Soit G un graphe à n sommets et m arêtes, et k l'entier donnés pour le problème de couverture par sommets.

Pour chaque arête $\{v_i, v_j\}$ avec $i < j$, on introduit le mot (deux lettres b en positions $3i+1$ et $3j$)

$$a^{3i}ba^{3(j-i)-2}ba^{3(n-j)+3}$$

(longueur $3n+3$).

On ajoute p copies de

$$t = c(001)^n 00c$$

(longueur $r = 3n+4$) , et q copies de

$$x = cd^k c$$

Alignement de x et t

- Les deux c au début et à la fin des deux mots doivent être alignés
- Il y a autant de 1 dans t que de sommets dans le graphe
- Les d de x sont de préférence à aligner avec des 1 qu'avec des 0
- La distance est alors $2(3n+4)-4-k = 6n+4-k$

Alignement de mot d'arête et t

- On peut soit mettre un $-$ au début du mot d'arête soit à la fin
- Le b du premier sommet ou celui du deuxième est alors aligné avec un 1
- Le minimum est atteint avec l'alignement de x pour aligner ce b avec un d

Tout alignement a la forme suivante:

- L'espacement de chaque mot d'arête doit être ou bien tout au début ou bien tout à la fin
- les mots x et t sont alignés identiquement
- il est possible d'aligner un des b de chaque mot d'arête avec un d si et seulement si G possède une couverture comportant k sommets.

Algorithme approché

1. Pour chaque paire i, jtelle que $1 \leq i < j \leq k$, en utilisant l'algorithme d'alignement de deux séquences, calculer $\delta(f_i, f_j)$.
2. Soit f^* le f_i qui minimise

$$\sum_{j \neq i} \delta(f_i, f_j)$$

. Le mot f^* est appelée le centre de la famille. Quitte à renuméroter, on peut supposer que $f^* = f_1$. On pose $f'_1 := f_1$.

3. Pour j allant de 2 à k faire:

   - Aligner f_i et f'_1 en utilisant l'algorithme d'alignement de deux séquences.
   - Ceci définit f'_i et modifie f'_1 en y insérant des blancs.
   - Mettre à jour $f'_2, \ldots, f'_{i-1}$ en y insérant des blancs aux mêmes endroits que dans f'_1.

Analyse de l'algorithme

Evaluation de l'alignement obtenu:

$$\sum_{1\leq i<j\leq k} \delta'(f'_i, f'_j) \leq \sum_{1\leq i<j\leq k} \delta'(f'_i, f'_1) + \delta'(f'_1, f'_j)$$

$$= (k-1) \sum_{1\leq i\leq k} \delta'(f'_i, f'_1) = (k-1) \sum_{1\leq i\leq k} \delta(f_i, f_1).$$

Soit $f^*_1, \ldots, f^*_k$ l'alignement optimal. Sa valeur satisfait:

$$\sum_{1\leq i<j\leq k} \delta'(f^*_i, f^*_j) \geq \sum_{1\leq i<j\leq k} \delta(f_i, f_j) =$$

$$\frac{1}{2} \sum_i \sum_{j\neq i} \delta(f_i, f_j) \geq \frac{1}{2} \sum_i \sum_j \delta(f_i, f_1) = \frac{k}{2} \sum_j \delta(f_i, f_1).$$

La dernière inégalité découle de la définition du centre f_1.