Algorithmes pour la Bio-informatique

1. Alignement de familles de séquences
2. Recherche d'un motif dans un texte
3. Alignement local
4. Sous-séquences maximales
5. Reconstruction d'arbres d'évolution
6. Estimation de distance entre génomes

Alignement de familles de séquences

Les bases de données pour les protéines les regroupent habituellement en familles. Trouver la famille à laquelle appartient un nouveau gène.

Représentation de la famille: alignement de tous ses membres.

Alignements de deux séquences

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

Similarité : $p : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \to \mathbf{R}_+$

$$p(a, a) > 0, \qquad p(a, b) < 0 \text{ si } b \neq a.$$

Exemple : **GATGCAT et ATCGAT**

G	A	T	–	G	C	A	T
–	A	T	C	G	–	A	T

Algorithme d'alignement

Entrée : $u_1 \cdots u_n$ et $v_1 \cdots v_m$ mots sur $\{A, C, G, T\}$.

Soit $s(i, j)$ valeur du meilleur alignement entre $u_1 \cdots u_i$ et $v_1 \cdots v_j$.

Définition récursive :

$$\begin{aligned}
s(i,0) &= \sum_{k \le i} p(u_k, -) \\
s(0,j) &= \sum_{k \le j} p(-, v_k) \\
s(i,j) &= \max \begin{cases} s(i-1, j-1) + p(u_i, v_j) \\ s(i-1, j) + p(u_i, -) \\ s(i, j-1) + p(-, v_j) \end{cases}
\end{aligned}$$

But : $s(n, m)$

Implantation : Tableau $s[0 \ldots n, 0 \ldots m]$.

Principe d'un algorithme en espace linéaire

- Le calcul des meilleurs scores peut se faire sur une seule ligne
- Le problème est de retrouver l'alignement une fois que l'on a le score
- On procède par divide and conquer
- Pour aligner u et v on prend i comme la moitié de la longueur de u
- On calcule les valeurs a_j des meilleurs alignements de $u_1, u_2, \ldots u_{i-1}$ avec tous les facteurs gauches de v
- On calcule les valeurs b_j des meilleurs alignements de $u_{i+1}, u_{i+2}, \ldots u_n$ avec tous les facteurs droits de v
- On cherche le meilleur $a_{j-1} + b_{j+1} + p(u_i, v_j)$

Calcul de la complexité de l'algorithme en espace linéaire

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

Alignement de famille de séquences

Insérer des blancs dans chaque $S_i \longrightarrow S'_i$ pour obtenir $|S'_1| = |S'_2| = \ldots = |S'_k|$ en minimisant

$$\sum_{1 \leq i < j \leq k} D(i,j)$$

$$D(i,j) = \sum_{1 \leq \ell \leq N} \delta(S'_i[\ell], S'_j[\ell]).$$

NP-difficile

Un algorithme d'approximation

Centre : S_i qui minimise $\sum_j D(S_i, S_j)$, où D est calculée avec l'algorithme d'alignement de deux séquences.

1. Trouver le centre S_1.
2. Pour j allant de 2 à k faire:

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

Analyse

Complexité : pour k séquences de longueur n:

$$O(\frac{k(k-1)}{2}n^2 + \sum_{2\leq j\leq k} O(jn\cdot n)) = O(k^2n^2).$$

Qualité de la solution :

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

$$= (k-1)\sum_{1\leq i\leq k} D(S'_i, S'_1) = (k-1)\sum_{1\leq i\leq k} D(S_i, S_1).$$

Borne inférieure

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

$$\sum_{1\leq i<j\leq k} D(S^*_i, S^*_j) \geq \sum_{1\leq i<j\leq k} D(S_i, S_j) = \frac{1}{2}\sum_i \sum_{j\neq i} D(S_i, S_j)$$

$$\geq \frac{1}{2}\sum_i \sum_j D(S_i, S_1) = \frac{k}{2}\sum_j D(S_i, S_1).$$

Conclusion : c'est une 2-approximation.

Recherche de motif

Entrée : mot P de longueur m petite (le motif), texte T de longueur n grande (le texte).

Sortie : Décider si P est un facteur de T.

Algorithme naïf : $O(mn)$

Algorithme de Knuth-Morris-Pratt : $O(m+n)$

Principe de l'algorithme

Sachant $P[1\ldots i] = T[k+1\ldots k+i]$, on compare $P[i+1]$ à $T[k+i+1]$.

Si $P[i+1] = T[k+i+1]$, parfait, on incrémente i.

Mais si $P[i+1] \neq T[k+i+1]$, on décale P vers la droite.

De combien décaler ? Le plus possible, sachant $P[1\ldots i] = T[k+1\ldots k+i]$ et $P[i+1] \neq T[k+i+1]$.

Choix du décalage

Condition nécessaire : On décale P en P'. Les x premiers caractères de P' sont les x derniers de P, le suivant est différent. x est maximum avec cette propriété.

Table s précalculée : $s(i)$ est le plus grand suffixe de $P[1\ldots i]$ qui soit aussi préfixe de P, avec $P[i+1] \neq P[s(i)+1]$.

Décalage : de $i - s(i)$ caractères vers la droite.

Analyse de complexité

Construction de s : $O(m)$ (cours Majeure 1).

Comparaisons entre P et T :

- si mismatch : on décale P vers la droite $\Longrightarrow$ au plus n telles comparaisons.
- si match : on m'emorise grâce à s et on ne fait plus jamais de comparaison à ce caractère de T $\Longrightarrow$ au plus n telles comparaisons.

Total au plus $2n$ comparaisons.

Sous-séquences maximales

Motivation : Identifier de nouveaux gènes. Le codage des gènes occupe moins de 3% du génome humain.

Idée : Les nouveaux gènes ressembleront à ceux qui sont déjà déterminés. En particulier par la distribution des codons.

Score : Chaque codon a une fréquence dans le génome et une fréquence a priori différente dans les gènes déjà connus. On se sert de ces deux fréquences pour lui associer un “score” (log-likelihood).

Idée : Une région dont le score total est élevé correspond potentiellement à un nouveau gène.

Le problème

Entrée : Suite $X_1, X_2, \ldots, X_n$ de réels positifs ou négatifs, les scores.

Sortie : Facteur $X_i \ldots X_j$ qui maximise $X_i + X_{i+1} + \ldots + X_j$.

Comment trouver ce facteur rapidement ?

Algorithme

Soit a_k le meilleur score d'un facteur de $X_1 \dots X_k$, et b_k le meilleur score d'un suffixe de $X_1 \dots X_k$.

$$a_0 = b_0 = 0$$

$$\begin{aligned} a_k &= \max(a_{k-1}, b_{k-1} + X_k) \\ b_k &= \max(b_{k-1} + X_k, 0) \end{aligned}$$

Complexité : linéaire

Reconstruction d'arbre d'évolution

Entrée : n espèces; matrice $D(i, j)$: distance entre les espèces i et j et leur plus proche ancêtre commun.

Sortie : l'arbre d'évolution lorsqu'il en existe un compatible avec D.

Ultramétrique

Définition : Une matrice (n, n) définit une ultra-métrique si pour tout triplet $\{i, j, k\}$, le maximum de $D(i, j), D(j, k)$ et $D(k, i)$ est atteint par au moins deux de ces trois valeurs.

Théorème : Etant donné n espèces avec leur matrice D symétrique associée, le probléme de reconstruction de l'arbre d'évolution a une solution si et seulement si D définit une ultra-métrique.

Distance entre génomes

Entrée : Deux génomes $g = g_1 \dots g_n$ et $g' = g_{\sigma(1)} \dots g_{\sigma(n)}$.

Modèle de mutation : Prendre un facteur et le renverser.

$$g_1 \dots g_i \dots g_j \dots g_n \to g_1 \dots g_{i-1} g_j g_{j-1} \dots g_i g_{j+1} \dots g_n$$

Question : Distance entre g et g', en termes de nombre de mutations ?

Une approche stochastique

Coupure : i tel que g_i et g_{i+1}, voisins dans g, ne sont pas voisins dans g'

Facile à calculer : nombre de coupures entre g et g'.

Idée : s'en servir comme d'un estimateur de la distance.

Algorithme

Soit D le nombre de coupures entre g et g'.

Soit D_t le nombre de coupes entre $g(t)$ et $g(0)$, où $g(t)$ est le descendant de g après t mutations.

1. On calcule $E(D_t)$ pour $t = 1, 2, 3, \ldots$ en procédant par récurrence.
2. On choisit le t tel que $|E(D_t) - D|$ soit minimum.