IFT6299 A05 — Devoir 2

Miklós Csűrös

14 novembre 2005

À remettre au cours du 1 décembre.

3 points de boni pour une soumission imprimée ou numérique (au lieu de manuscrite).

1 D’ailleurs, la pondération la plus simple n’est pas aussi simple que ça (20 points)

En parlant de la pondération de substitutions, on a travaillé souvent avec la pondération simple de $+1$ pour match et -1 pour mismatch. Dans le contexte de maximum vraisemblance, quelles distributions de nucléotides et probabilités de substitutions mènent à un tel système de scores ? En d’autres mots, trouvez les solutions pour $\pi_{\sigma'}$ et $p_{\sigma \rightarrow \sigma'}$ quand $\sigma, \sigma' \in \{A, C, G, T\}$ (notation de scores LODS). (Les scores sont choisis en utilisant un facteur d’échelle α quelconque, mais sans l’arrondi au plafond.) **10 points de boni** : À quelles probabilités de substitution correspondent les scores $+2$ et -1 ? À quelles probabilités de substitution correspondent les scores $+3$ et -1 ?

2 Alignement davidien (30 points)

Notre ami David veut trouver le meilleur alignement global de deux séquences S et T avec tout au plus d trous. Les trous dans les alignements ne contribuent pas au score ; les identités et substitutions sont pondérées selon une matrice de scores C . Les trous peuvent être de n’importe quelle longueur. Exemple : (match = $+1$, mismatch = -1)

```
AACTGAATG---GTTA---GGACCC  
ACCT---TGAAAGCTACCCGGTCCC
```

C’est un alignement avec trois trous et un score de 10 (13 identités et 3 substitutions). Donnez un algorithme pour trouver le meilleur alignement global avec $\leq d$

trous où d est un paramètre d'entrée. Analysez la complexité d'espace et de temps de votre algorithme.

3 Une extravagance de checkpoints (30 points)

Au cours j'ai mentionné l'idée d'utiliser des *checkpoints* afin de réduire l'usage de mémoire lors du calcul de probabilités postérieures dans la segmentation par un modèle de Markov caché. Plus précisément, le problème est le suivant. On veut calculer les probabilités postérieures pour la segmentation d'une séquence de longueur n : on utilise un HMM avec des états (classes de segments) Q . Soit $m = |Q|$. Dans une première phase, les probabilités suffixes $\beta_j(i)$ sont calculées pour $i = n, n-1, \dots, 1$ et stockées aux *checkpoints* $i \in \{kB+1 : k = 1, \dots, n/B\}$ où B est une taille quelconque. Dans une deuxième phase, les probabilités préfixes $\alpha_j(i)$ sont calculées pour $i = 1, 2, \dots, n$. Dès que le calcul arrive dans le bloc $(k-1)B+1..kB$, les probabilités suffixes sont calculées à partir du *checkpoint* $kB+1$. Par conséquent, on a besoin d'un mémoire de taille $m(n/B + B)$ pour stocker les $\beta_j(i)$. Le choix de $B = \sqrt{n}$ minimise le mémoire. Comment peut-on réduire l'usage de mémoire davantage ? L'idée est d'appliquer la méthode de *checkpoints* de manière récurrente. Pour le début, supposons deux niveaux de *checkpoints*. Au niveau 1, la séquence est coupée en blocs de taille $B_1 = B$. Quand on arrive dans le k -ème bloc au niveau 1 dans la deuxième phase, le calcul de $\beta_j(i)$ procède en utilisant des blocs de taille B_2 (niveau 2) : les $\beta_j(i)$ sont stockées dans les premières positions de ces blocs ($i = (k-1)B_1 + B_2 + 1, (k-1)B_1 + 2B_2 + 1, \dots$), calculées dans un premier parcours de droite à gauche dans le bloc du premier niveau. Toutes les valeurs dans un bloc de niveau 2 sont calculées à partir du *checkpoint* de niveau 2 pour le bloc quand on y arrive. Avec deux niveaux, on a donc besoin d'un mémoire de taille $m(n/B_1 + B_1/B_2 + B_2)$. Le temps de calcul est deux fois plus grand que dans le cas simple sans *checkpoints* (une unité de temps pour calculer les $\alpha_j(i)$; chaque $\beta_j(i)$ est calculé trois fois qui donne trois unités de temps). En général, on a besoin d'un mémoire de taille

$$m \left(\frac{n}{B_1} + \frac{B_1}{B_2} + \frac{B_2}{B_3} + \dots + \frac{B_{d-1}}{B_d} + B_d \right)$$

où d est le nombre de niveaux de *checkpoints*. Le temps de calcul est $\frac{d+2}{2}$ fois plus grand que dans le cas simple sans *checkpoints*.

Voici alors les questions. Quel est le choix de $B_1, \dots, B_d$ qui minimise le mémoire pour un d fixe ? Quel est le nombre de niveaux qui minimise l'usage de mémoire ? Quel est la complexité du calcul (temps et espace) avec autant de niveaux ?

4 Levenchtein et le glouton (20 points)

Ce problème vous demande de concevoir un algorithme glouton qui permet de calculer la distance d'édition entre deux séquences S, T en temps $O((|S| + |T|)D)$ où D est la distance entre S et T . Évidemment, c'est une amélioration sur le temps $O(|S| \cdot |T|)$ de programmation dynamique dans le cas où S et T sont très similaires.

Soit $D(i, j)$ la distance d'édition entre les préfixes de longueur i et j . On a démontré que $D(i, 0) = i$, $D(0, j) = j$ et

$$D(i, j) = \min\left\{D(i-1, j) + 1, D(i, j-1) + 1, D(i-1, j-1) + \{S[i] \neq T[j]\}\right\},$$

si $i, j > 0$ où $\{Z\}$ est l'indicateur de Z (1 si Z est vrai ou 0 sinon). L'idée de clé de la solution est de considérer comment $D(i, j)$ change sur les *diagonales*. La diagonale k est l'ensemble de cases (i, j) dans le tableau de PD où $i = j + k$. Donc si $|S| = n$ et $|T| = m$, le tableau contient les diagonales $k = -m, -m + 1, \dots, 0, 1, \dots, n - 1, n$. Pour $d = 0, 1, \dots, n + m$ et $|k| \leq d$, on définit

$$R(d, k) = \max\{j : D(k + j, j) = d\},$$

donc $R(d, k)$ est la colonne de la dernière case sur la diagonale k dont la valeur est égale à d . S'il n'y a pas de telle case (tous les $D(i, j) < d$ dans la diagonale), elle n'est pas définie : $R(d, k) = -\infty$. Notez que si $R(d, |T| - |S|) = |T|$, alors $D(|S|, |T|) = d$.

Si on a trouvé une case avec $D(i, j) = d$, alors on peut trouver l'extension la plus longue pour obtenir $R(d, i - j)$.

Procédure Extension

Entrée : (i, j) [case du tableau PD]

- 1 **tant que** $i < |S|$ et $j < |T|$ et $S[i + 1] = T[j + 1]$ **faire** $j \leftarrow j + 1; i \leftarrow i + 1$
- 2 retourner j // (c'est $R(D(i, j), i - j)$)

Par exemple, comme $D(0, 0) = 0$, on trouve $R(d, 0)$ qui correspond au plus petit j tel que $S[j + 1] \neq T[j + 1]$. Pour calculer $R(d + 1, k)$ à partir des $R(d, k')$, on utilise de la récurrence et la procédure d'extension. L'idée de la récurrence est que pour calculer $R(d + 1, k)$, on ne doit considérer que la diagonale k et ses deux voisines $(k - 1)$ et $(k + 1)$. Montrez comment combiner $R(d, k - 1)$, $R(d, k)$ et $R(d, k + 1)$ avec la procédure Extension pour trouver $R(d + 1, k)$. Basé sur ce résultat, décrivez un algorithme qui calcule $R(d, k)$ pour $d = 0, 1, \dots$, jusqu'à ce qu'il arrive à $R(d, |T| - |S|) = |T|$. Analysez la complexité de temps et d'espace de l'algorithme.