

# BIN702 - Série d'exercices sur les structures secondaires d'ARN

Manuel Lafond

Note : ici j'utilise  $V[i, j]$  pour dénoter les tables de programmation dynamique (car j'utilise  $M$  pour dénoter une matrice de scores). La récurrence de Nussinov calcule  $V[i, j]$  pour tout  $i, j$ , où  $V[i, j]$  est le nombre maximum d'appariements pour  $S[i..j]$ . On a  $V[i, j] = 0$  pour  $i \geq j$  et sinon,

$$V[i, j] = \max \begin{cases} V[i + 1, j] \\ \max_{k=i+1..j} (V[i + 1, k - 1] + V[k + 1, j] + \delta_{ik}) \end{cases}$$

où  $\delta_{ik} = 1$  si  $S[i]$  s'apparie avec  $S[k]$ , et  $\delta_{ik} = -\infty$  sinon.

**Exercice 1:** En classe, nous avons présenté une version récursive de l'algorithme de Nussinov. Écrivez le pseudo-code d'une version itérative de l'algorithme. C'est-à-dire, calculez les valeurs de la table de programmation dynamique avec des boucles, sans avoir recours à des appels récursifs.

**Solution.** Il faut seulement s'assurer de parcourir la table de programmation dynamique en remplissant diagonale par diagonale. La variable  $d$  représentera l'indice (i.e. le décalage horizontal) de la diagonale courante. On ajuste  $i$  et  $j$  en conséquence.

```

fonction nussi3(S)
  V = [, ];
  pour d = 1..n faire
    pour i = 1..n - d + 1 faire
      j = i + d - 1;
      si i ≥ j alors
        | V[i, j] = 0;
      sinon
        | V[i, j] = V[i + 1, j];
        pour k = i + 1..j faire
          | si S[i] s'apparie avec S[k] alors
            | | tmp = V[i + 1, k - 1] + V[k + 1, j] + 1;
            | | V[i, j] = max(V[i, j], tmp);
          fin
        | ; // Note: si on voulait faire du backtracking,
        |   c'est ici qu'il faudrait mémoriser le cas qui
        |   a mené à la valeur de V[i, j]
      fin
    fin
  fin
  return V[1, n];

```

□

**Exercice 2:** La récurrence de Nussinov calcule un nombre maximum d'appariements. Supposons plutôt que nous avons une matrice  $M$  qui donne un score d'appariement  $M[a, b]$  pour chaque paire de nucléotides  $a, b$ . Donnez une récurrence qui permet de maximiser la somme des scores des appariements, toujours avec condition d'emboîtement des appariements.

**Solution.** Une fois qu'on a bien compris, c'est trop facile !

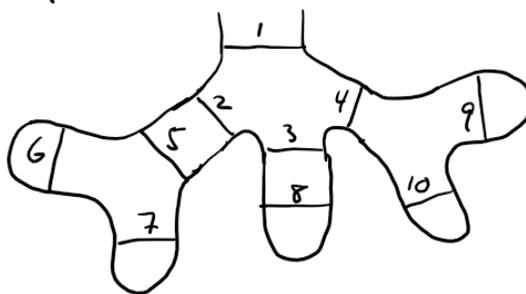
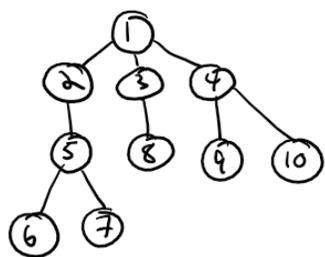
$$V[i, j] = \max \begin{cases} V[i + 1, j] \\ \max_{k=i+1..j} (V[i + 1, k - 1] + V[k + 1, j] + M[S[i], S[k]]) \end{cases}$$

□

**Exercice 3:** Considérez la séquence d'ARN ci-dessous avec ses paires de bases inférées. Dessinez l'arbre correspondant à ces appariements, puis dessinez un repliement en 2D qui illustre les sous-structures principales de la molécule.



**Solution.** Voici, avec les appariements numérotés (je ne dessine pas tous les nucléotides, seulement les appariements) :

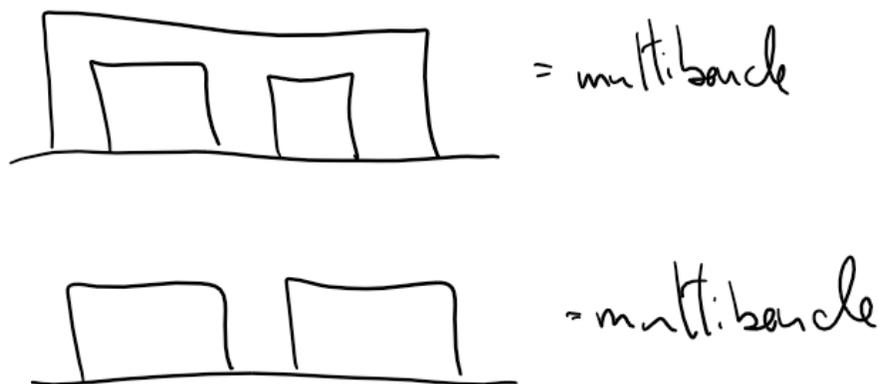


□

**Exercice 4:** Supposons que l'on veut interdire les boucles multiples dans le contexte de maximisation du nombre d'appariements. On suppose que les seuls appariements possibles sont  $A - U$  et  $C - G$ . Donnez un algorithme en temps  $O(n^2)$  qui trouve ce maximum.

*Indice* : sans multi-boucle, un appariement ne peut pas avoir plus d'un appariement se trouvant directement à l'intérieur.

Solution.



La figure illustre les imbrications interdites : il ne peut pas y avoir deux appariements au même niveau. Ceci dit, il y a au maximum un seul appariement qui n'est contenu dans aucun autre appariement (qui correspond à la racine de l'arbre correspondant). Comment le trouver ? Supposons qu'on sait que cet appariement est  $A - U$ . On observe que sans multi-boucle, il vaut mieux prendre le  $A$  le plus à gauche et le  $U$  le plus à droite. Ceci est vrai parce que chaque appariement autre sera à l'intérieur de ce  $A - U$ , et donc on maximisera les appariements possibles en prenant la paire  $A - U$  la plus éloignée.

Par contre, on ne sait pas quels sont les caractères qui forment l'appariement au plus haut niveau. Ça pourrait être  $A - U$  ou bien  $C - G$ . On essaie deux possibilités et on forme une récurrence, comme d'habitude.

Soit  $V[i, j]$  le nombre maximum d'appariements sur  $S[i..j]$ . On pose  $V[i, i] = 0$  et  $V[i, j] = V[j, i]$  si  $i > j$ .

Pour chaque appariement possible  $X - Y$ , soit  $i_X$  la position du premier  $X$  sur  $S[i..j]$  et soit  $j_Y$  la position du dernier  $Y$  sur  $S[i..j]$ . Il est possible que  $i_X > j_Y$ , dans quel cas on inverse les deux positions. On a donc

$$V[i, j] = \max_{X-Y \text{ possible}} (1 + V[i_X + 1, j_Y - 1])$$

Puisque le nombre d'appariements possibles  $X - Y$  est constant, chaque  $V[i, j]$  peut se calculer en temps  $O(1)$ . Il y a  $O(n^2)$  entrées à calculer, et donc on peut implémenter cette récurrence en temps  $O(n^2)$ .

Ce qui n'a pas été expliqué, c'est comment trouver  $i_X$  et  $j_Y$ . Une façon est de faire un prétraitement pour avoir accès à  $i_X$  en temps  $O(1)$  pendant l'exécution de la récurrence. C'est-à-dire, on va d'abord préstocker, pour chaque position  $i$  et chaque caractère  $X \in \{A, C, G, U\}$ , la position  $i_X$  du prochain caractère  $X$  à partir de  $i$ . Je vous laisse l'exercice d'effectuer ce prétraitement, qui est facile à faire en temps  $O(n^2)$ . De la même façon, on peut préstocker tous les  $j_Y$ . Donc, ajouter un prétraitement  $O(n^2)$  à un algorithme déjà  $O(n^2)$  ne change pas la complexité.

Vous devriez vous demander pourquoi cette récurrence ne fonctionne pas si on permet les multi-boucles.  $\square$

**Exercice 5:** La récurrence de l'algorithme de Nussinov pour  $V[i, j]$  était basée sur deux cas possibles : soit  $i$  n'est pas apparié, ou soit  $i$  est apparié avec un certain  $k \in \{i + 1, \dots, j\}$ .

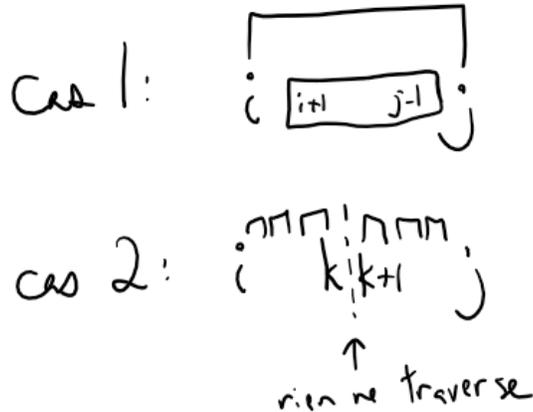
Il existe une autre version (équivalente) de la récurrence de Nussinov. Celle-ci est basée sur une autre séparation des cas possibles : soit  $i$  est apparié avec  $j$ , ou bien  $i$  n'est pas apparié avec  $j$ . Dans ce deuxième cas, il doit exister un  $k \in \{i, \dots, j - 1\}$  tel que aucun appariement ne traverse  $k$ , c'est-à-dire que tous les appariements sont entre  $i$  et  $k$ , ou entre  $k + 1$  et  $j$ .

Donnez une récurrence de programmation dynamique avec cette séparation en deux cas.

**Solution.** Les deux cas sont illustrés dans la figure ci-dessous. Vous devriez prendre le temps de vous convaincre que le deuxième cas s'applique bel et bien. La récurrence correspondante est

$$V[i, j] = \max \begin{cases} V[i + 1, j - 1] + \delta_{ij} \\ \max_{k=i..j-1} (V[i, k] + V[k + 1, j]) \end{cases}$$

$\square$



**Exercice 6:** Considérez l'algorithme qui minimise l'énergie libre des sous-structures d'un repliement 2D. On avait une récurrence pour  $V[i, j]$  avec quatre cas possibles, un pour chaque sous-structure  $T, E, B$  et  $M$ . Les multi-boucles ( $M$ ) étaient le cas le plus complexe à gérer, alors supposons qu'on s'en débarrasse et qu'on ne permette que les cas  $T, E$  et  $B$ . Décrivez la structure des arbres d'appariements qui sont possibles avec cette contrainte, puis décrivez informellement la forme des molécules d'ARN qui peuvent résulter de l'algorithme avec cette contrainte.

**Solution.** Puisqu'un noeud avec  $\geq 2$  enfants correspond à une multi-boucle, tout sommet de l'arbre aura 1 ou 0 enfants. Donc l'arbre est en fait un chemin. La structure de l'ARN aura une forme phallique (oui, phallique), i.e. ce sera essentiellement deux lignes droites jointes par une tige-boucle.  $\square$

**Exercice 7:** Supposons que l'on veut maximiser le nombre d'empilements, c'est-à-dire le nombre d'appariements  $(i, j)$  tels que  $(i + 1, j - 1)$  sont aussi appariés. Donnez un algorithme qui maximise le nombre d'empilements.

**Solution.** Finalement ce n'était pas si compliqué. Prenons l'approche habituelle de programmation dynamique et d'énumération des cas possibles. Soit  $V[i, j]$  le nombre maximum d'empilements possibles sur  $S[i..j]$ . On a deux cas possibles :

- a.  $i$  ne participe pas à un empilement. Dans ce cas, ce n'est même pas la peine d'apparier  $i$  et on peut donc supposer que  $i$  n'est pas apparié. Dans ce cas on a  $V[i, j] = V[i + 1, j]$ .
- b.  $i$  participe à un empilement. Dans ce cas, il existe  $k \geq i + 3$  tel que l'on

peut former les appariements  $(i, k), (i + 1, k - 1)$ . On ne connaît pas  $k$ , alors on les essaie tous. Ce cas peut être évalué avec

$$V[i, j] = \max_{k=i+3..j} (V[i + 2, k - 2] + V[k + 1, j] + \delta(i, k, i + 1, k - 1))$$

où on définit  $\delta(i, k, i + 1, k - 1) = 1$  si les paires  $(S[i], S[k])$  et  $(S[i + 1], S[k - 1])$  sont appariables, et sinon c'est  $-\infty$ .

Bref, la récurrence est

$$V[i, j] = \max \begin{cases} V[i, j] = V[i + 1, j] \\ \max_{k=i+3..j} (V[i + 2, k - 2] + V[k + 1, j] + \delta(i, k, i + 1, k - 1)) \end{cases}$$

ce qui peut se calculer en temps  $O(n^3)$ . □