TP 1 Assemblage de séquences IFT3295

Pierre Emery 20278920 Mehdi Qostali 20260322

Automne 2025

Partie I — Chevauchement de séquences

1) Différence avec alignement global

L'alignement global pénaliserait les gaps sur toute la longueur et forcerait l'alignement de tout X_i avec X_j. Alors que l'alignement du suffixe de X_i et du préfixe de X_j que l'on fait ne pénalise pas le préfixe de X_i qui ne serait pas aligné et il ne pénalise pas le suffixe de X_j qui ne serait pas alligné non plus.

2) Initialisation de V

— **Première ligne.** $\forall j,\ V(0,j) = j \cdot \text{gap}$: on autorise de consommer un préfixe de X_j et chaque caractère non aligné au début de X_j coûte un gap.

— **Première colonne.** $\forall i,\ V(i,0) = 0$: on ne pénalise pas le préfixe de X_i puisque l'on cherche à aligner un suffixe de X_i.

3) Récurrence

Il faut qu'on considère la pondération proposée, soit de +4 pour un match, -4 pour un mismatch et -8 pour un indel (insertion ou supression). On a donc une formule de récurrence Pour $i \geq 1$, $j \geq 1$,

$$V(i,j) = \max \begin{cases} V(i-1,j-1) + s(x_i, y_j), & \text{(diagonale)} \\ V(i-1,j) - 8, & \text{(gap dans } Y\text{)} \\ V(i,j-1) - 8, & \text{(gap dans } X\text{)} \end{cases}$$

avec $s(a,b) = +4$ si $a = b$, sinon -4.

4) Retrouver l'alignement de meilleur chevauchement

Une fois la table de programmation dynamique V remplie, la remontée (ou traceback) consiste à retrouver pas à pas le chemin qui amène au score optimal. On commence par repérer la meilleure valeur dans la dernière ligne de la matrice. À partir de cette position (i, j), on remonte dans la table en suivant les transitions qui conservent le score optimal. Lorsque la valeur courante $V(i, j)$ provient de la diagonale, c'est que les nucléotides X_i et X_j sont appariés : on les ajoute tous deux à l'alignement. Si le score provient de la cellule supérieure, on considère un appariement de X_i avec un `-` (gap dans X_j), et inversement, s'il provient de la cellule de gauche, on aligne X_j avec un `-`. On continue ainsi jusqu'à atteindre le bord supérieur ou gauche de la matrice. Finalement on inverse les deux alignements partiels que nous venons de trouver ce qui donne les alignements finaux.

5)* Implémentation en $O(|X_i|\,|X_j|)$: score, alignement et longueur de chevauchement

Réponse. Convention pour les questions marquées * (code). Toutes les questions marquées d'un astérisque renvoient à un script Python `.py` annoté et commenté. L'en-tête de chaque fichier explique comment l'exécuter et ce qu'il imprime. Dans le rapport, on indique le fichier associé et les sorties (console, tables, graphes) avec des commentaires parfois ou des choix d'implémentations.

Fichier associé : `prefixe_suffixe.py`.
Commande : `python prefixe_suffixe.py two_reads.fastq --match 4 --mismatch -4 --gap -8`
Sortie console :

```
IDs: read1 (X_i) -> read2 (X_j)
Score: 12
Overlap length: 3
Alignement X_i:
CCC
Alignement X_j:
CCC
```

Partie II — Assemblage de fragments

1)* Matrice 20×20

Réponse. Fichier associé : `matrice.py`.
Commande : `python matrice.py reads.fq --out matrice_20x20.csv`
Sortie (tableau) :

	$READS_1$	$READS_2$	$READS_3$	$READS_4$	$READS_5$	$READS_6$	$READS_7$	$READS_8$	$READS_9$	$READS_{10}$	$READS_{11}$	$READS_{12}$	$READS_{13}$	$READS_{14}$	$READS_{15}$	$READS_{16}$	$READS_{17}$	$READS_{18}$	$READS_{19}$	$READS_{20}$
$READS_1$	0	8	8	0	4	12	12	16	4	804	20	20	620	4	524	12	12	1072	4	608
$READS_2$	4	0	8	12	44	8	4	416	8	12	424	16	0	1060	12	12	12	8	4	0
$READS_3$	396	12	0	24	1016	16	48	0	36	128	4	20	40	8	20	20	16	8	8	8
$READS_4$	28	8	28	0	12	640	1024	16	1068	12	12	220	16	24	12	656	152	16	8	16
$READS_5$	500	4	696	28	0	24	52	8	52	208	16	20	36	4	8	24	36	64	4	12
$READS_6$	52	4	4	16	0	0	4	4	12	28	4	16	4	0	28	8	672	16	12	20
$READS_7$	32	12	32	648	16	720	0	4	772	12	12	24	24	24	24	176	232	12	4	0
$READS_8$	8	12	644	4	540	16	8	0	8	4	944	12	4	12	0	4	8	36	4	16
$READS_9$	36	12	20	956	16	668	1044	4	0	8	4	64	12	24	20	492	196	8	4	12
$READS_{10}$	4	0	0	8	32	8	8	24	12	0	32	4	12	12	640	4	8	904	184	868
$READS_{11}$	20	20	620	4	524	12	12	1072	4	12	0	24	28	24	8	8	16	20	12	12
$READS_{12}$	28	28	48	900	16	460	820	0	856	4	16	0	28	24	16	996	20	16	8	4
$READS_{13}$	68	32	32	744	24	272	656	0	692	16	8	860	0	36	16	848	32	24	4	28
$READS_{14}$	16	756	12	12	52	20	8	452	12	16	492	16	8	0	4	12	20	16	8	16
$READS_{15}$	20	8	28	12	16	12	16	12	8	4	12	80	260	28	0	28	20	24	632	24
$READS_{16}$	20	16	24	1032	16	552	944	12	988	8	20	692	8	20	12	0	80	12	8	8
$READS_{17}$	16	4	32	16	16	32	8	12	4	24	20	12	4	28	12	8	0	16	8	20
$READS_{18}$	16	0	8	12	8	4	4	8	8	240	16	8	4	16	808	8	4	0	368	1004
$READS_{19}$	48	20	16	388	8	20	316	8	368	16	20	520	732	16	24	492	20	20	0	4
$READS_{20}$	8	4	0	4	0	8	4	0	0	0	4	0	64	8	908	4	16	644	428	0

2)* Graphe orienté de chevauchement $G = (V, E)$

Réponse. Fichier associé : `graph.py`.
Commande : `python graph.py matrice_20x20.csv --threshold 80`
Sortie console :

```
Écrit : graph_1.dot
Écrit : graph_2.dot
```

Nous avons visualisé ces fichiers avec Graphviz (`dot`) pour produire les figures ci-dessous

2a)* **Filtrage par seuil : effet de 80 sur le graphe**

Réponse.

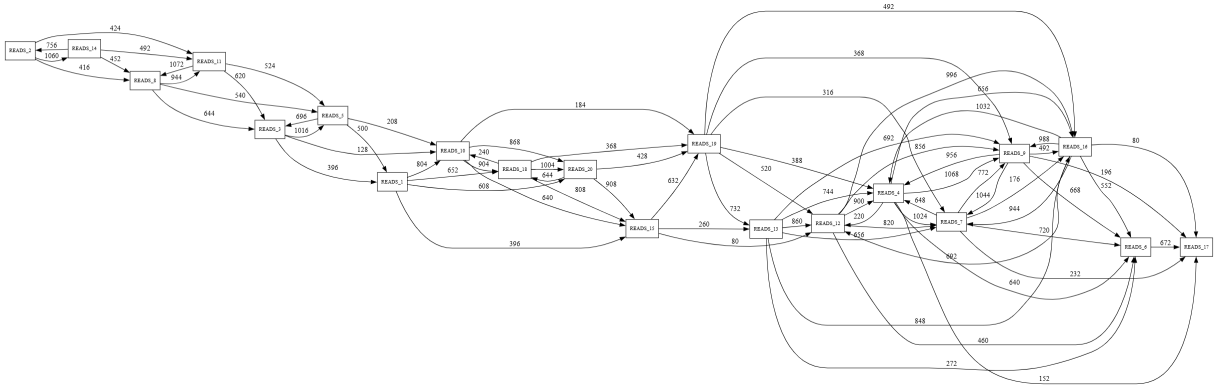


FIGURE 1 – Graphe d'overlap filtré au seuil 80 (visualisation de graph_1.dot)

Effet du seuil 80. Le seuil élimine toutes les arêtes dont le score de chevauchement est < 80. Le graphe devient donc plus lisible vu que beaucoup d'arêtes faibles disparaissent, les degrés des sommets diminuent, et malgré cela le graphique reste relativement difficile à lire.

2b)* **Réduction transitive, ordre des reads et longueurs de chevauchement**

Réponse.

FIGURE 2 – Graphe réduit après suppression des 2-cycles et réduction transitive (visualisation de graph_2.dot)

Interprétation. Pour ce graphique on a gardé que l'arête au meilleur score puis on enlève une arête directe si un chemin indirect existe, on obtient un graphe beaucoup plus linéaire et très facile à lire. Et ici on nous demandait l'ordre des reads et la longueur de chevauchement que nous n'avons pas imprimés à la console mais ces informations sont tout simplement sur le graphe. l'ordre se lit en suivant le chemin orienté du graphe réduit et la longueur du chevauchement entre deux reads consécutifs est le poids associé aux arêtes.

2c)* **Séquence et taille**

Réponse. Fichier associé : `sequence_frag.py`.
Commande : `python sequence_frag.py --dot graph_2.dot`
Sortie console :

```
1586
```

```
CTAGGGACTTGGAGAACACAAGTATTATGAAAAGTACTGATGAAAGTTATTAACAGGTTTCGAAAAATAACTTTACTATC
↪ TGACGTGTTGCTTCTGCCGAGGATGACCGTTATTCCTGGTTTTGCATTTATATTTCACGTATGGTTAAATGTGCCA
↪ GCGTTGTGGTTTAAAACTAATAGTAATAATATGCTTCTTTGTTCAGTTGGCTAGAGATTTACTACATCCGTCCTTG
↪ GAAGAGGAAAAGAAAAAACATAACAAGAAACGCCTAGTACAAAGTCCAAATTCTTACTTTATGGATGTAAAATGTC
↪ CAGGTAAAATTTGAAATCTTAATTCCTTTACTAAAGAAAATTTCTGTAGGGATTGCTAGTGTGGTGTGTATAGTTA
↪ AGATACATTAGAATCCTCTGTTGAGTAGAAGTGGGATTACAGAATTGGACATGTCAGGGACAATTTCATAATAAAT
↪ GTACTGAAACCATTGTAGAAACATTTCGGGGTAAGTGAAAACAGGCATTCAAAGTAGATATGCCATTTGAGTGCAC
↪ TTGTGGCTAATCAAACTGTGGATACTCAAAATAGAGATGTTCTTTAATTGTAAAATCCTCACTGGATTTTGACGAT
↪ GTAGCACAGAAAAAAAATACATTGATTACACTGTTTTTAAAAAATTTTGTGTCGCTGCTAGAAAAGTTTAAGTTAC
↪ ACATGTGGCTTGGTGTTTCATAGCACTGAAGTTACTGATTTTTTTACATATTACCAGAAATGTCGAACAGGTGCTG
↪ TTTTCCTCTGCTTTTCATTTTTAAGATTGCCTTTTTTTTTTTTTTAGGTTGCTACAAGATCACCACGGTTTTCAGCC
↪ ATGCTCAGACTGTGGTTCTTTGTGTAGGTTGTGCAACAGTGTTGTGCCAGCCTACAGGAGGAAAGGCCAGACTCAC
↪ AGAAGGTATATCATTTGGCATTCTCCAACCCAGTGATGAGATTGATGATTTTAAATGTCTCTATATTAACTGAAAA
↪ GTTTAAAGAAATCTTAATGATTCCCAAAATAACTTATCTCACACTGGAAGAGTTCAAGTGGATTGGCAGCAAATCT
↪ GAGATCTATTTGGTGTGACCTGGTGAGATCTAAATATGGAGTCAGCACATGATTTTTTTAAGAGTAATATTGCTAA
↪ GTAATATTGCTAAGTATCGTCTGAAAATACCTCTAATCAAAATTATTTACTTGAGAAAAGTATTCAGAATAGTTCC
↪ TAAAAATTAATAGTATATTTCTGGTATATAAGCATAAATAATCTGTATATGAGTATTAATCCAATATTCTTAAAAC
↪ TTCAGTATTTTACTTAAAAGTACTATTTGTCATTAAAATTATACCAAAGGTAGAATGCACTTGTTTAATATACTCT
↪ CATGATTCTTTTGCAGGGTGTTCATTTAGAAGAAAGCAACACTAATGATTCAAACAGCTTCCTGAATTTTAATTTT
↪ GTGTTGTCTCTCAGAAAGCCTTATCATAAATTCCATAATTCTAATTAATTTACCAAGATAATGTCATTACATTTGG
↪ TTATGTAAGTTATACAGCAGTAATCTCCTATTTTGGTGTCAGTTTTTCACTAAAGTTTTTAT
```

Partie III — Recherche d'introns et Blast

1) Identifier la position de la protéine X

a)* **Fichier associé :** `codon_start.py`.
Commande : `python codon_start.py --genome sequence.fasta --protein geneX.fasta`
Sortie console :

```
Cadre 2
```

b) L'algorithme qu'on implémenterait ici serait inspiré des alignements globaux mais on changerait le score puisque les indels ne sont pas permis dans les exons. On ajouterait à la place la possibilité d'introns ou de saut pour une pénalité indéfinie ω.

Scores :

$$\text{match} = 0, \qquad \text{mismatch} = +1, \qquad \text{intron (saut)} = +\omega$$

où ω représente le coût d'ouverture d'un intron (pénalité fixe).

Conditions initiales : On considère la protéine $P = P_1P_2 \dots P_m$ (sur l'axe vertical i) et la traduction du génome $A = A_1A_2 \dots A_n$ (sur l'axe horizontal j). La table $D(i, j)$ contient le coût minimal d'alignement des i premiers acides aminés de la protéine avec les j premiers codons du génome.

$$\forall j \quad D(0, j) = 0 \qquad \text{et} \qquad \forall i \quad D(i, 0) = +\infty$$

Relations de récurrence :

$$D(i,j) = \min \begin{cases} D(i-1, j-1) + 0, & \text{si } A_j = P_i \quad \text{(match)} \\ D(i-1, j-1) + 1, & \text{si } A_j \neq P_i \quad \text{(mismatch)} \\ D(i-k, j) + \omega, & \text{si l'on saute un bloc de } k \text{ codons (intron)} \end{cases}$$

Retrouver l'alignement optimal : On commence par rechercher la valeur minimale dans la dernière ligne du tableau. La remontée de l'algorithme suit la provenance de chaque cellule : une transition diagonale correspond à un exon (match ou mismatch), tandis qu'un saut vertical indique un intron.

Complexité : n est la longueur de la séquence génomique traduite et m celle de la protéine. La complexité de l'algorithme est alors

— $O(nm)$ en temps.
— $O(nm)$ en espace.

.

2) Identifier le nom et la fonction de la protéine X

Réponse. Pour le nom de la protéine X nous avons trouvé avec l'outil BLASTp partagé qu'il s'agit de 40S ribosomal protein S27-like (RPS27L). Ensuite, après avoir fait le recherche BLAST cette même ressource contient une section appelée Gene Ontology qui explique ces diverses fonctions. On comprend donc qu'elle se lie à l'ARN, aux protéines et aux ions métalliques (notamment zinc). Elle contribue aussi à la structure du ribosome et à la traduction de l'ARN messager en protéines.

Partie IV — Repliements d'ARNs

7) Graphique Tige-Boucle

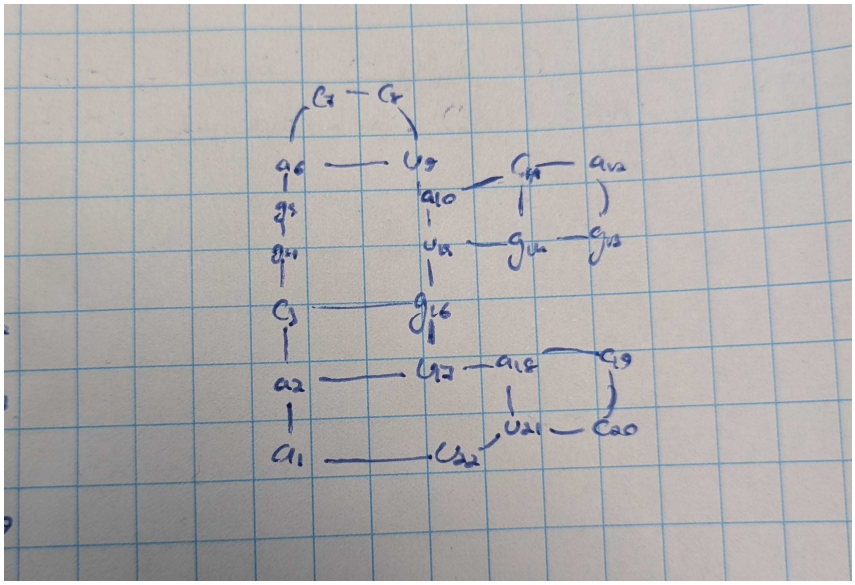
Nous avons commencé par numéroter les nucléotides de la séquence partagée afin de mieux nous situer dans le graphique. La séquence est la suivante :

`aacggaccuacagguguaccuu`

et nous lui associons les indices $a_1, a_2, c_3, g_4, g_5, a_6, c_7, c_8, u_9, a_{10}, c_{11}, a_{12}, g_{13}, g_{14}, u_{15}, g_{16}, u_{17}, a_{18}, c_{19}, c_{20}, u_{21}, u_{22}$

Voici notre graphique Tige-Boucle. Le repliement obtenu correspond à la structure secondaire suivante :

`(((..(..)((..))))(..))`



8) Bonus

Ici, on s'inspire de l'algorithme de Nussinov, mais on l'adapte afin qu'il maximise non pas le nombre d'appariements, mais plutôt le nombre d'empilements (c'est-à-dire les paires consécutives (i, j) et $(i+1, j-1)$ toutes deux appariées).

Initialisation. On conserve les conditions initiales du Nussinov classique et on ajoute celles pour la matrice des empilements E :

- $D[i][i] = 0 \quad \forall i$ tq $1 \leq i \leq n$
- $D[i][i-1] = 0 \quad \forall i$ tq $2 \leq i \leq n$
- Si les nucléotides S_i et S_j ne peuvent pas s'apparier, alors $E[i][j] = -\infty$

Équations de récurrence (pour $i < j$).

$$D(i,j) = \max \begin{cases} D(i+1, j) & \text{si } S_i \text{ non apparié} \\ D(i, j-1) & \text{si } S_j \text{ non apparié} \\ \max\limits_{i<k<j} \big(D(i,k) + D(k+1,j)\big) & \text{(bifurcation)} \\ E(i,j) & \text{si } S_i \text{ et } S_j \text{ s'apparient} \end{cases}$$

Et pour la matrice E (qui compte les empilements) :

$$E(i,j) = \begin{cases} \max\big(D(i+1,j-1),\ 1 + E(i+1,j-1)\big) & \text{si } S_i, S_j \text{ peuvent s'apparier} \\ -\infty & \text{sinon.} \end{cases}$$

Idée clé. Le terme $1+E(i+1,j-1)$ n'est activé que lorsque les deux paires (i,j) et $(i+1,j-1)$ sont toutes deux présentes, ce qui correspond exactement à un empilement.

Reconstruction On part de la case $(1,n)$ dans D et on remonte le chemin optimal en suivant la valeur qui réalise le maximum : si $D(i,j) = D(i+1,j)$, on marque i non apparié et on avance à $(i+1,j)$; si $D(i,j) = D(i,j-1)$, on marque j non apparié et on passe à $(i,j-1)$; si $D(i,j) = D(i,k)+D(k+1,j)$ pour un k, on bifurque en traitant séparément (i,k) et $(k+1,j)$. Sinon (donc $D(i,j) = E(i,j)$ et (S_i, S_j) s'apparient), on ajoute la paire (i,j) ; si $E(i,j) = 1 + E(i+1,j-1)$ on prolonge l'hélice en allant à $(i+1,j-1)$, sinon on descend simplement à $(i+1,j-1)$.

Complexité.

- $O(n^3)$ en temps.
- $O(n^2)$ en espace.

.