Mathis Degryse <mathis.degryse@proton.me>

Problème 2024
Les modèles de Markov cachés

Partie I. Définitions et propriétés élémentaires

1. Quelques définitions

1.1. Les Marches

Question 1

a.

La marche 13444 est possible dans le modèle H_1 et ce avec probabilité

$$\begin{aligned} Prob_{H_1}(13444) &= P(1) \times T(1,3) \times T(3,4) \times T(4,4)^2 \\ &= 1 \times \frac{1}{2} \times \frac{3}{10} \times 1^2 \\ &= \frac{3}{20} = 0.15 \end{aligned}$$

b. La marche 234 n'est pas possible dans le modèle H_1 car $P(2) = 0$

c. La marche 132323232 est possible dans le modèle H_1 et ce avec probabilité

$$\begin{aligned} Prob_{H_1}(132323232) &= P(1) \times T(1,3) \times T(3,2)^4 \times T(2,3)^3 \\ &= 1 \times \frac{1}{2} \times \frac{1}{5^4} \times \frac{4^3}{5^3} \\ &= \frac{2^5}{5^7} = \frac{2^{12}}{2^7 \cdot 5^7} = \frac{4096}{10^7} = 0.0004096 \end{aligned}$$

d. La marche 1234 n'est pas possible dans le modèle H_1 car $T(1,2) = 0$

e. La marche 13432 n'est pas possible dans le modèle H_1 car $T(4,3) = 0$

1.2. Les séquences

Question 2

a. La séquence AATATA peut être produite dans le modèle H_1 par la marche 132323

b. La séquence CATGTG ne peut pas être produite dans le modèle H_1 car on a $P(1) = 1$ et $E(1, C) = 0$. C'est à dire qu'une marche commence toujours par l'état 1 et que celui-ci ne peut pas émettre la lettre C.

c. La séquence AACGAA peut être produite dans le modèle H_1 par la marche 134444

d. La séquence AAGGGG peut être produite dans le modèle H_1 par la marche 134444

Question 3

a. La séquence AGCGAT peut être produite dans le modèle H_1 par la marche 134444, la probabilité, sachant le modèle H_1, que le marche 134444 produise AGCGAT est

$$\begin{aligned}
Prob_{H_1}(134444, \text{AGCGAT}) &= Prob_{H_1}(134444) \times Prob_{134444,H_1}(\text{AGCGAT}) \\
&= \frac{3}{20} \cdot E(1,A) \cdot E(3,G) \cdot E(4,C) \cdot E(4,G) \cdot E(4,A) \cdot E(4,T) \\
&= \frac{3}{20} \cdot 1 \cdot \frac{1}{2} \cdot \frac{1}{10} \cdot \frac{2}{5} \cdot \frac{1}{5} \cdot \frac{3}{10} \\
&= \frac{18}{10^5} = 0.00018
\end{aligned}$$

b. La séquence AGAAA peut être produite dans le modèle H_1 par la marche 13333, la probabilité, sachant le modèle H_1, que le marche 13333 produise AGAAA est

$$\begin{aligned}
Prob_{H_1}(13333, \text{AGAAA}) &= Prob_{H_1}(13333) \times Prob_{13333,H_1}(\text{AGAAA}) \\
&= P(1) \cdot T(1,3) \cdot T(3,3)^3 \cdot E(1,A) \cdot E(3,G) \cdot E(3,A)^3 \\
&= 1 \cdot \frac{1}{2} \cdot \frac{1}{2^3} \cdot 1 \cdot \frac{1}{2} \cdot \frac{1}{2^3} \\
&= \frac{1}{2^8} = \frac{5^8}{10^8} = \frac{390625}{10^8} = 0.00390625
\end{aligned}$$

Question 4

La même marche 134444 peut produire les séquences des questions 2.c) et 2.d).

On peut montrer qu'il existe une unique séquence par marche possible dans un modèle H si et seulement si pour chaque état $q \in S$ accessible depuis un état initial et pour chaque lettre $c \in \Sigma$ on a $E(q,c) \in \{0,1\}$

Question 5

Dans le modèle H_1, les deux marches 134 et 133 peuvent toutes les deux produire la même séquence AAA avec probabilité respective $\frac{1}{16}$ et $\frac{3}{200}$.

Question 6

Les marches $113, 133, 132, 134$ sont les 4 marches qui peuvent produire la séquence AAG dans le modèle H_1. On a alors

$$\begin{aligned}
Prob_{H_1}(113, \text{AAG}) &= \frac{1}{8} \\
Prob_{H_1}(133, \text{AAG}) &= \frac{1}{16} \\
Prob_{H_1}(132, \text{AAG}) &= \frac{1}{40} \\
Prob_{H_1}(134, \text{AAG}) &= \frac{3}{100}
\end{aligned}$$

Finalement, la marche maximisant la probabilité de produire la séquence AAG sachant le modèle H_1 est 113

Question 7

Soit w une séquence sur l'alphabet Σ. On considère la valeur

$$\begin{aligned}\Pr_w &= \sum_{\substack{m\in S^* \\ Prob_H(m)\neq 0}} Prob_H(m,w) \\ &= \sum_{m\in S^*} Prob_H(m,w)\end{aligned}$$

Si on effectue une marche aléatoire sur le modèle H, c'est à dire qu'on se déplace aléatoirement une infinité de fois et qu'on considère le mot produit $\omega \in \Sigma^{\mathbb{N}}$. Alors, $\Pr_w$ est la probabilité que $w \sqsubseteq \omega$, i.e. que w soit un préfixe de ω. Ainsi, il s'agit de la probabilité qu'une séquence ADN suivant ce modèle commence par w.

2. Quelques propriétés

Question 8

La Fig. 1 est une représentation sous forme de graphe de H_0. On peut vérifier que ce modèle vérifie les conditions (1), (2) et (3). On a $n = 1$ et donc

$$\sum_{i=1}^{n} P(i) = P(1) = 1$$

$$\sum_{j=1}^{n} T(i,j) = T(1,1) = 1$$

ainsi que

$$\begin{aligned}\sum_{c\in\Sigma} E(i,c) &= E(1,A) + E(1,C) + E(1,G) + E(1,T) \\ &= 4 \times \frac{1}{4} = 1\end{aligned}$$

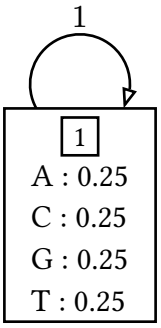

Fig. 1. – Représentation de H_0

Question 9

Pour tout $n \in \mathbb{N}$, on considère la séquence A^n, on a alors

$$\begin{aligned}Prob_{H_0}(1^n, A^n) &= P(1) \times T(1,1)^{n-1} \times E(1,A)^n \\ &= \frac{1}{4^n} > 0\end{aligned}$$

On en déduit que la séquence A^n peut être produite par H_0, et donc que ce modèle peut produire des séquences de longueur arbitraire.

Question 10

Soit $w \in \Sigma_0^+$, soit n la longueur de w, qu'on note $w = w_1 w_2 ... w_n$, on a de manière similaire

$$Prob_{H_0}(1^n, w) = P(1) \times T(1,1)^{n-1} \times \prod_{i=1}^{n} E(1, w_i)$$
$$= \frac{1}{4^n} > 0$$

On en déduit qu'avec la marche 1^n, H_0 peut produire n'importe quelle séquence non vide sur son alphabet.

Question 11

a. On a $|\Sigma_0| = 4$ et donc $|\Sigma_0^k| = |\Sigma_0|^k = 4^k$.

Si on veut une preuve un peu plus formelle, on considère la fonction $c : \Sigma_0 \longrightarrow [\![0, 3]\!]$ qui à A associe 0, à C associe 1... Puis la fonction

$$f : \Sigma_0^k \to [\![0, 4^k - 1]\!]$$
$$w \mapsto \sum_{i=1}^{k} c(w_i) \cdot 4^{i-1}$$

On peut alors se convaincre facilement que cette fonction réalise une bijection de Σ_0^k dans $[\![0, 4^k - 1]\!]$, car elle calcule le nombre dont l'écriture en base 4 avec les lettres de Σ_0 est w.

On peut aussi faire une preuve par récurrence, plus fastidieuse.

b. La question 10 nous donne déjà le résultat, car 1^k est l'unique marche de longueur k, et qu'un mot de longueur k ne peut être produit que par une marche de longueur k.Ainsi, la probabilité d'une séquence de Σ_0^k dans H_0 est $\frac{1}{4^k}$

c. On a

$$\sum_{w \in \Sigma_0^k} Prob_{H_0}(1^k, w) = \sum_{w \in \Sigma_0^k} \frac{1}{4^k}$$
$$= 4^k \times \frac{1}{4^k} = 1$$

Question 12

Une séquence de taille 1 ne peut être produite que par une marche de taille 1. L'unique marche de taille 1 pouvant être produite par H_1 est 1. Cette marche ne peut produire qu'une seule séquence, la séquence A, avec probabilité 1.

Question 13

Les marches de taille 2 possibles dans H_1 sont 11 et 13.

- 11 produit uniquement la séquence AA avec probabilité $\frac{1}{2}$
- 13 produit soit la séquence AA avec probabilité $\frac{1}{4}$, soit AG avec probabilité $\frac{1}{4}$.

Cela fait donc une probabilité de $\frac{3}{4}$ pour AA et $\frac{1}{4}$ pour AG, ce qui somme encore une fois à 1.

Question 14

La séquence A^n peut être produite par le modèle H_1, avec la marche 1^n et ce avec probabilité

$$Prob_{H_1}(1^n, A^n) = P(1) \cdot T(1,1)^{n-1} \cdot E(1, A)^n$$
$$= \frac{1}{2^{n-1}} > 0$$

On en déduit que H_1 peut produire des séquences de longueur arbitraires.

Question 15

Il suffit pour cela que le graphe représentant H ait un cycle accessible depuis un état inital. Une condition encore plus forte (impliquant la première) peut être que chaque sommet ait un arc sortant. La relation entre ces deux propriétés peut être démontrée rapidement.

En effet, supposons que tous les sommets aient un arc sortant, on note n le nombre de sommets du graphe. On considère un sommet initial du graphe. On peut alors construire un chemin de longueur $n+1$ partant de ce sommet, en suivant des arcs sortants. Par principe des tiroirs, au moins un des sommets du graphe apparait deux fois dans ce chemin, ce qui démontre l'existence d'un cycle accessible depuis un sommet initial.

Le fait que n'importe quelle représentation d'un modèle de Markov caché vérifie cette propriété est une conséquence directe de la propriété (2) de l'énoncé.

Question 16

On va démontrer par récurrence sur $k \in \mathbb{N}^*$ que pour tout $k \geq 1$ on a

$$\sum_{w \in \Sigma^k} \sum_{m \in S^k} Prob_H(m, w) = 1$$

Commencons par démontrer le cas où $k = 1$. On a

$$\begin{aligned}
\sum_{w \in \Sigma^1} \sum_{m \in S^1} Prob_H(m, w) &= \sum_{w \in \Sigma} \sum_{m \in S^1} P(m) \cdot E(m, w) \\
&= \sum_{m \in S^1} \sum_{w \in \Sigma} P(m) \cdot E(m, w) \\
&= \sum_{m \in S^1} P(m) \sum_{w \in \Sigma} E(m, w) \\
&= \sum_{m \in S^1} P(m) \\
&= 1
\end{aligned}$$

Procédons maintenant à la récurrence, on considère un certain $k \in \mathbb{N}^*$, et on suppose que

$$\sum_{w \in \Sigma^k} \sum_{m \in S^k} Prob_H(m, w) = 1$$

Montrons alors que

$$\sum_{w \in \Sigma^{k+1}} \sum_{m \in S^{k+1}} Prob_H(m, w) = 1$$

On a

$$\sum_{w\in\Sigma^{k+1}}\sum_{m\in S^{k+1}} Prob_H(m,w) = \sum_{w\in\Sigma^{k+1}}\sum_{m\in S^{k+1}} P(m_1)\cdot\prod_{i=1}^{k} T(m_i,m_{i+1})\cdot\prod_{i=1}^{k+1} E(m_i,w_i)$$

$$= \sum_{w\in\Sigma^{k+1}}\sum_{m\in S^{k+1}} \left[P(m_1)\cdot\left(\prod_{i=1}^{k-1} T(m_i,m_{i+1})\right)\cdot\left(\prod_{i=1}^{k} E(m_i,w_i)\right)\cdot T(m_k,m_{k+1})\cdot E(m_{k+1},w_{k+1})\right]$$

$$= \sum_{w\in\Sigma^{k+1}}\sum_{m\in S^{k+1}} Prob_H(m_{1..k},w_{1..k})\cdot T(m_k,m_{k+1})\cdot E(m_{k+1},w_{k+1})$$

$$= \sum_{\substack{w\in\Sigma^k\\ c\in\Sigma}}\sum_{\substack{m\in S^k\\ s\in S}} Prob_H(m,w)\cdot T(m_k,s)\cdot E(s,c)$$

$$= \sum_{w\in\Sigma^k}\sum_{m\in S^k}\sum_{s\in S}\sum_{c\in\Sigma} Prob_H(m,w)\cdot T(m_k,s)\cdot E(s,c)$$

$$= \sum_{w\in\Sigma^k}\sum_{m\in S^k}\left[Prob_H(m,w)\sum_{s\in S}\left[T(m_k,s)\sum_{c\in\Sigma}E(s,c)\right]\right]$$

$$= \sum_{w\in\Sigma^k}\sum_{m\in S^k}\left[Prob_H(m,w)\sum_{s\in S}T(m_k,s)\right]$$

$$= \sum_{w\in\Sigma^k}\sum_{m\in S^k} Prob_H(m,w) = 1$$

Ce qui conclut la récurrence.

Partie II. Représentation des modèles de Markov cachés

3. La classe `HMM`

Question 17

Un objet de la classe `HMM` doit avoir comme attributs :

- un entier `n` représentant son nombre d'états.
- une matrice de transition `trans` sous forme de tableau bidimensionnel.
- un alphabet `sigma`, sous forme d'ensemble
- le vecteur initial `first`, sous forme de tableau.
- une matrice d'émission `emission`, sous forme de tableau de dictionnaire.

Pour les méthodes, il faut un constructeur, prenant en entrée `sigma, emission, trans`, cette méthode peut alors initialier les attributs et vérifier la conformité des valeurs, comme par exemple que les clés des dictionnaires dans `emission` ne sont que des lettres de `sigma`, ou que les équations $(1), (2), (3)$ soient vérifiées, mais ceci n'est pas forcément possible si on manipule des nombres floattants.

On peut aussi envisager d'autres méthodes, comme une qui pourrait calculer $Prob_H(m,w)$ en prenant en entrée deux listes.

Question 18

Les vérifications pour le constructeur évoquées dans la question précédente ne sont pas implémentées car elles sont très nombreuses et certaines sont inadaptées à Python. Comme par exemple vérifier que les éléments de `trans` sont des floattants.

```python
class HMM:
    def __init__(self, trans, emission, first, sigma):
```

```
3           self.n = len(trans)
4           self.trans = trans
5           self.emission = emission
6           self.sigma = sigma
7           self.first = first
8
9       def prob(self, m, w):
10          p = self.first[m[0]] * self.emission[m[0]][w[0]]
11          k = len(m)
12          for i in range(1,k):
13              p *= self.trans[m[i-1]][m[i]] * self.emission[m[i]][w[i]]
14          return p
```

Question 19

Il suffit d'initaliser les arguments selon la définition de H_0 :

```python
def __init__(self, trans = [[1]],
             emission = [{A : 1/4, C : 1/4, G : 1/4, T : 1/4}],
             first = [1],
             sigma = {A,C,G,T}):
    self.n = len(trans)
    self.trans = trans
    self.emission = emission
    self.sigma = sigma
    self.first = first
```

Question 20

Les instructions suivantes permettent d'instancier le modèle H_1.

```python
trans1 = [
    [0.5, 0  , 0.5, 0  ],
    [0  , 0.1, 0.8, 0.1],
    [0  , 0.2, 0.5, 0.3],
    [0  , 0  , 0  , 1  ],
]

emission1 = [
    {A : 1  , C : 0  , G : 0  , T : 0  },
    {A : 0  , C : 0  , G : 0.5, T : 0.5},
    {A : 0.5, C : 0  , G : 0.5, T : 0  },
    {A : 0.2, C : 0.1, G : 0.4, T : 0.3}
]

first1 = [1, 0, 0, 0]

```

```
17  H1 = HMM(trans1, emission1, first1)
```

Question 21

Il faut bien faire attention à vérifier les égalités dans le bon ordre pour ne pas créer d'erreur. Si on compare des matrices de transitions qui n'ont pas la même taille, on peut avoir un *out of bounds* !

```python
1   def __eq__(self, other):
2       # check number of states
3       if self.n != other.n:
4           return False
5       # check transition matrix
6       for i in range(self.n):
7           for j in range(self.n):
8               if self.trans[i][j] != other.trans[i][j]:
9                   return False
10      # check sigma1 included in sigma2
11      for c in self.sigma:
12          if c not in other.sigma:
13              return False
14      # assuming the above, check sigma1 = sigma2
15      if len(self.sigma) != len(other.sigma):
16          return False
17      # check emission matrix
18      for i in range(self.n):
19          for c in self.sigma:
20              if self.emission[i][c] != other.emission[i][c]:
21                  return False
22      return True
```

Question 22

```python
1  def estValide(self):
2      if sum(self.first) != 1:
3          return False
4      for i in range(self.n):
5          if sum(self.trans[i]) != 1:
6              return False
7          if sum(self.emission[i].values()) != 1:
8              return False
9      return True
```

Cette fonction ne marche pas si on utilise des flottants, auquel cas on peut se contenter de vérifier que chacune des sommes est comprise dans un petit intervalle autour de 1.

4. Adaptation à une famille de séquences

Question 23

Le modèle de Markov caché H_2 correspondant à l'alignement de la figure 2 de l'énoncé est représenté en Fig. 2

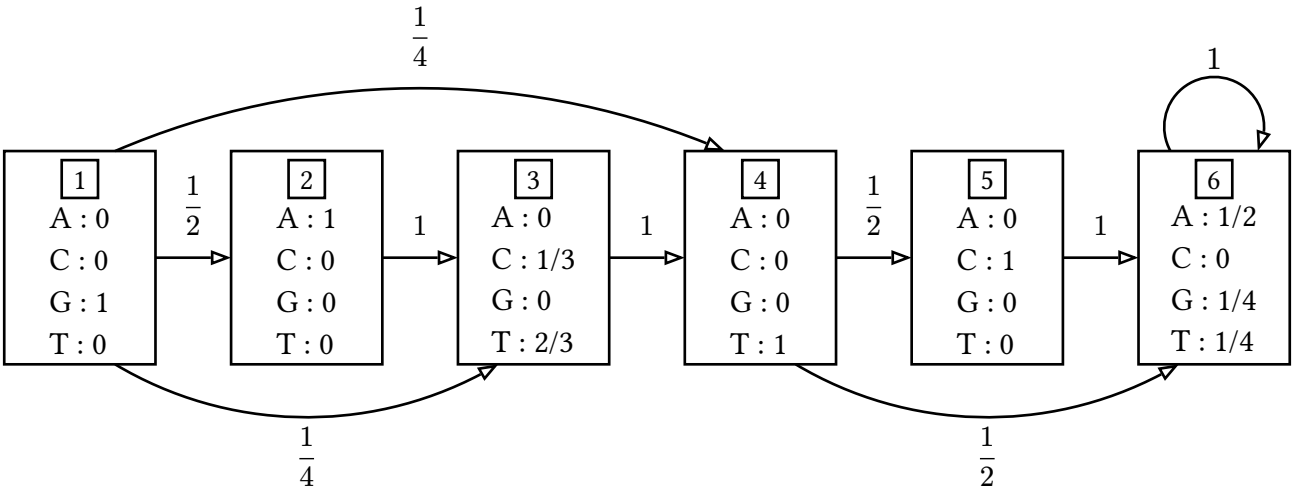

Fig. 2. – Représentation de H_2

Le vecteur de probabilités de départ est simplement le vecteur $(1, 0, 0, 0, 0, 0)$

Question 24

Comme on a un nombre fixe d'alignements qui sont tous de même longueur, on peut utiliser un tableau à deux dimensions, autrement dit en python une liste de liste. Cela permet d'accéder à n'importe quel caractère de n'importe quel alignement en temps constant.

Question 25

```python
def fromAlignement(self, align):
        lettres = {A, C, G, T}
        q = len(align)
        p = len(align[0])
        trans = [[0] * p for _ in range(q)]
        trans[p-1][p-1] = 1
        emission = [{A : 0, C : 0, G : 0, T : 0}]
        first = [0] * p
        first[0] = 1
        for j in range(p):
            total = 0
            for i in range(q):
                if align[i][j] in lettres:
                    emission[j][align[i][j]] += 1
                    total += 1
                if align[i][j] in lettres:
                    next = j+1
                    while align[i][next] not in lettres:
```

```
                        next += 1
                    trans[j][next] += 1/total
            for c in lettres:
                emission[j][c] /= total
        return HMM(trans, emission, first)
```

Question 26

- La condition (1) est évidemment vérifiée, car on a $P(1) = 1$ et pour tout $i \neq 1$, on a $P(i) = 0$, par définition d'un modèle associé à un alignement.
- La condition (2) est vérifiée car on a

$$\begin{aligned}\sum_{j=1}^{n} T(i,j) &= \sum_{j=1}^{n} \frac{|\{l \in [\![1,q]\!] \mid j_{i,l} = j\}|}{c_i} \\ &= \frac{1}{c_i} \sum_{j=1}^{n} |\{l \in [\![1,q]\!] \mid j_{i,l} = j\}|\end{aligned}$$

  Or, dans la i-eme colonne, si le l-eme caractère est un « - », alors $j_{i,l}$ n'existe pas, et si c'est une lettre, alors il existe un unique l telle que $j_{i,l} = j$, ainsi on a

$$\begin{aligned}\sum_{j=1}^{n} T(i,j) &= \frac{1}{c_i} \sum_{j=1}^{n} |\{l \in [\![1,q]\!] \mid j_{i,l} = j\}| \\ &= \frac{1}{c_i} \cdot c_i = 1\end{aligned}$$

- Comme les probabilités d'émissions correspondent aux fréquences des lettres dans la colonne correspondante, il est direct que ces valeurs somment à 1, donc la condition (3) est vérifiée.

5. Sur la représentation du modèle

Question 27

Comme les gaps sont de longueurs au plus g, il ne peut exister de transitions de i à j que si $j \in [\![i+1, i+g+1]\!]$ (sauf dans le cas de l'état final).

On en déduit que sur chaque ligne , il y a au plus $g+1$ entrées non nulles (dans le cas de l'état final, il y en a au plus 1, et donc moins de $g+1$).

On en déduit qu'il y a au plus $q \cdot (g+1)$ entrées non nulles dans T.

Question 28

Comme il y a exactement $q \cdot p$ éléments dans T, on en déduit directement que la densité de T est majorée par $\frac{g+1}{p}$

Question 29

Si g est petit devant p, alors une liste d'adjacence est beaucoup plus efficace en mémoire qu'une matrice d'adjacence. Dans le cas de H_2, on aurait la liste d'adjacence suivante :

```python
[
    [(1,1/2), (2,1/4), (3,1/4)],
    [(2,1)],
    [(3,1)],
```

```
5      [(4,1/2), (5,1/2)],
6      [(5,1)]
7  ]
```

Il y a une transition de i à j avec probabilité a si et seulement si (j, a) est dans la liste $T[i]$

Question 30

On pourrait déjà envisager une liste de dictionnaires à la place de liste de listes. On obtiendrait alors une complexité en temps constant en moyenne, grâce à l'implémentation des dictionnaires en python par des tables de hachage.

On ne perd donc pas beaucoup en complexité en temps, et on gagne grandement en espace (un facteur $\frac{g}{p}$).

Il faudrait alors modifier les arguments par défaut du constructeur de .

Il ne faut rien changer d'autre, car les manipulations de liste de liste fonctionnent aussi sur les listes de dictionnaires !

Partie III. Algorithmes

6. L'algorithme de décodage

Question 31

On considère une séquence $w = w_1...w_k$ et une marche $m_1...m_k$ de probabilité maximale qui produit w. Alors, pour tout $k' \leq k$, et pour toute marche $n_1...n_{k'-1}m_{k'}$, la marche $m_1...m_k$ produit w avec plus grande probabilité que $n_1...n_{k'-1}m_{k'}...m_k$, ce qu'on obtient simplement par définition de m. On a alors

$$P(m_1)\cdot\prod_{i=1}^{k}E(m_i,w_i)\cdot\prod_{i=1}^{k-1}T(m_i,m_{i+1}) \geq P(n_1)\cdot\prod_{i=1}^{k'-1}E(n_i,w_i)\cdot\prod_{i=1}^{k'-2}T(n_i,n_{i+1})\cdot T(n_{k'-1},m_k)\cdot\prod_{i=k'}^{k}E(m_i,w_i)\cdot\prod_{i=k'}^{k-1}T(m_i,m_{i+1})$$

En simplifiant les termes en commun des deux côtés, on obtient

$$P(m_1)\cdot\prod_{i=1}^{k'-1}E(m_i,w_i)\cdot\prod_{i=1}^{k'-1}T(m_i,m_{i+1}) \geq P(n_1)\cdot\prod_{i=1}^{k'-1}E(n_i,w_i)\cdot\prod_{i=1}^{k'-2}T(n_i,n_{i+1})\cdot T(n_{k'-1},m_k)$$

On en déduit que $m_1...m_{k'}$ produit $w_1...w_{k'}$ avec plus grande probabilité que $n_1...m_{k'}$ et comme ceci est vrai indépendamment du choix de la deuxième marche, $m_1...m_{k'}$ est la marche produisant $w_{1..k'}$ avec la plus grande probabilité (parmi celles terminant dans l'état $m_{k'}$).

Question 32

a. La première ligne de ce tableau contient les probabilités maximales qu'une marche de longueur 1 se terminant en l'état j produise w_1. On a donc

$$\forall j \in [\![1,n]\!], \delta[1,j] = P(j)\cdot E(j,w_1)$$

b. Admettons qu'on veuille déterminer $\delta[i,j]$ pour $i > 1$ et un certain $j \in [\![1,n]\!]$. On sait qu'il existe une marche $m_1...m_{i-1}j$ produisant $w_{1..i}$ avec probabilité $\delta[i,j]$ (il s'agit de la définition de δ).

D'après la question précédente, $m_1 ... m_{i-1}$ produit la séquence $w_{1..(i-1)}$ avec probabilité $\delta[i-1, m_{i-1}]$ et donc que

$$\delta[i,j] = \delta[i-1, m_{i-1}] \cdot T(m_{i-1}, j) \cdot E(j, w_i)$$

Mais cette propriété est dépendante de m_{i-1}, or le tableau ne nous donne pas cette information. On a plus généralement

$$\delta[i,j] \in \{\delta[i-1,j'] \cdot T(j',j) \cdot E(j,w_i) \mid 1 \leq j' \leq n\}$$

Or tous les éléments de cet ensemble sont clairement des probabilités de marches produisant $w_{1..i}$ terminant dans l'état j. On en déduit alors que

$$\begin{aligned}\delta[i,j] &= \max_{1 \leq j' \leq n} (\delta[i-1,j'] \cdot T(j',j) \cdot E(j,w_i)) \\ &= E(j,w_i) \cdot \max_{1 \leq j' \leq n} (\delta[i-1,j'] \cdot T(j',j))\end{aligned}$$

c. Pour obtenir la probabilité maximale qu'une marche produise w, il s'agit simplement de calculer

$$\max_{1 \leq j' \leq n} \delta[n,j]$$

Question 33

Voici les deux tableaux obtenus, pour le tableau b, un ? signifie que n'importe quelle valeur aurait pu être placée ici.

δ	1	2	3	4
1	1	0	0	0
2	0	0	$\frac{1}{4}$	0
3	0	$\frac{1}{40}$	$\frac{1}{16}$	$\frac{3}{100}$

b	1	2	3	4
1	0	0	0	0
2	?	?	1	?
3	?	3	3	3

On a alors $z[3] = 3, z[2] = 3, z[1] = 1$, ainsi l'algorithme renvoie `[1,3,3]`

Question 34

On peut commencer par remarquer que l'algorithme de Viterbi calcule le tableau δ en utilisant exactement la formule donnée en question 32. Il ne nous reste plus qu'à montrer que le calcul de z est correct.

Considérons d'abord $z[k]$, $\delta[k, z[k]]$ est d'après la question 32 la probabilité maximale d'une marche produisant w dans H. Il suffit donc de montrer que la séquence z est une marche produisant w avec cette probabilité dans H.

On sait qu'il en existe une qui termine dans l'état $z[k]$, par construction de δ. Cette marche $i_1, ..., i_{k-1}, z[k]$ vérifie d'après la question 31.*a* que $i_1, ..., i_{k-1}$ est optimale pour $w_1 ... w_{k-1}$ si l'état final est fixé à i_{k-1}. Or ces marches optimales $i_1 ... i_{k-1}$ ont leur probabilité dans la case $\delta[k-1, i_{k-1}]$. Par optimalité de $i_1 ... i_{k-1} z[k]$, on a aussi que $\delta[k-1, i_{k-1}] \cdot T(i_{k-1}, z[k])$ est maximal. Il existe donc une marche optimale $i_1 ... i_{k-1} z[k]$ pour $w_1 ... w_k$ telle que i_{k-1} maximise $\delta[k-1, i_{k-1}] \cdot T(i_{k-1}, z[k])$, ce que calcule $b[k, z[k]]$.

Ainsi, il existe une marche optimale $i_1 ... i_{k-2} z[k-1] z[k]$. On peut alors réitérer ce raisonnement avec une récurrence, jusqu'à obtenir que z est une marche optimale.

Question 35

Toutes les opérations sauf max, arg max sont en $O(1)$, ces deux dernières sont elles en $O(|S|) = O(n)$. Ainsi, on obtient facilement la complexité de cet algorithme grâce aux boucles imbriquées, i.e. les boucles entre les lignes 10 et 15 font en tout $(k-1) \cdot n$ tours de boucles pour une complexité totale en $O(kn^2)$

7. Lest algorithmes d'évaluation

Question 36

Il suffit de faire la somme des probabilités obtenues à la question 6, i.e.

$$\begin{aligned}\frac{1}{8} + \frac{1}{16} + \frac{1}{40} + \frac{3}{100} &= \frac{50}{400} + \frac{25}{400} + \frac{10}{400} + \frac{12}{400} \\ &= \frac{97}{400} \\ &= \frac{97}{2^4 \cdot 5^2} \qquad \text{pas simplifiable car } 2 \nmid 97, 5 \nmid 97\end{aligned}$$

Question 37

Cette fois, on veut juste obtenir la probabilité d'une séquence dans le modèle, il n'y a aucun antécédant, notamment car cette probabilité provient sûrement de plusieurs marches distinctes. On n'a donc aucune raison de considérer la phase de backtracking.

Ensuite, il faut changer les équations, on a :

- $\alpha[1, j] = P(j) \times E(j, w_1)$
- $\alpha[i, j] = E(j, w_i) \cdot \sum_{l=1}^{n} \alpha[i-1, l] \cdot T(l, j)$

Question 38

la valeur finale a renvoyer est alors $\sum_{l=1}^{n} \alpha[k, l]$, ce qui s'obtient avec la formule des probabilités totales :

$$\begin{aligned}Prob_H(w) &= \sum_{l=1}^{n} Prob_H(w \mid \text{dernier état} = l) \\ &= \sum_{l=1}^{n} \alpha[k, l]\end{aligned}$$

Question 39

$\beta[i, j]$ contient la probabilité d'émettre la séquence $w_i ... w_k$ depuis l'état j, d'où son nom. En effet, au lieu de calculer la probabilité voulue en avancant dans le mot, on la calcule en partant de la fin.

La formule de la ligne 12 est similaire à celle de α, mais dans l'autre sens.

Question 40

Une fois arrivé à $\beta[1, j]$ pour tout j, il faut prendre en compte la probabilité de commencer par cet état, donc on multiplie (ligne 17) chaque $\beta[1, j]$ par $P(j)$.

Ensuite, on a

$$Prob_H(w) = \sum_{l=1}^{n} Prob_H(w \mid \text{premier état} = l)$$
$$= \sum_{l=1}^{n} \mathrm{B}[1, l]$$

Il suffit alors de sommet les $\beta[1, j]$

Question 41

Une analyse similaire à celle de l'algorithme de Viterbi nous donne une complexité de $O(k \cdot n^2)$ pour l'algorithme Forward. Pour la complexité en espace, on introduit seulement un tableau β de taille $O(k \cdot n)$.

Ces résultats valent aussi pour l'algorithme backward.

Question 42

We can directly translate the pseudo code in python and we get

```python
def backward(H,w):
    k = len(w)
    n = H.n
    beta = [[0] * n for _ in range(k)]
    for j in range(n):
        beta[k-1][j] = H.emission[j][w[k-1]]
    for i in range(k-2,-1,-1):
        for j in range(n):
            beta[i][j] = 0
            for l in range(n):
                beta[i][j] += beta[i+1][l] * H.trans[j][l]
            beta[i][j] *= H.emission[j][w[i]]
    for j in range(n):
        beta[0][j] *= H.first[j]
    return sum(beta[0])
```

Partie IV. Pour aller plus loin

8. Les pseudo-counts

Question 43

Il ne faut vérifier que la contrainte sur E. Soit E' la nouvelle matrice d'émission, on a, pour tout $i \in [\![1, n]\!]$:

$$\sum_{c \in \Sigma} E'(i, c) = \sum_{\substack{c \in \Sigma \\ E(i,c)=0}} \frac{1}{n \cdot n_i} + \sum_{\substack{c \in \Sigma \\ E(i,c)>0}} \frac{n-1}{n} \cdot E(i, c)$$
$$= n_i \cdot \frac{1}{n \cdot n_i} + \frac{n-1}{n} \cdot \sum_{\substack{c \in \Sigma \\ E(i,c)>0}} E(i, c)$$
$$= \frac{1}{n} + \frac{n-1}{n} = 1$$

Donc, le nouveau modèle vérifie bien les propriétés des modèles de markov cachés.

Question 44

Pour le modèle H_2 avec des pseudo-counts, on avait initialement la matrice :

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & \frac{2}{3} \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{4} & \frac{1}{4} \end{pmatrix}$$

qui devient alors la matrice :

$$\begin{pmatrix} \frac{1}{18} & \frac{1}{18} & \frac{5}{6} & \frac{1}{18} \\ \frac{5}{6} & \frac{1}{18} & \frac{1}{18} & \frac{1}{18} \\ \frac{1}{12} & \frac{5}{18} & \frac{1}{12} & \frac{5}{9} \\ \frac{1}{18} & \frac{1}{18} & \frac{1}{18} & \frac{5}{6} \\ \frac{1}{18} & \frac{5}{6} & \frac{1}{18} & \frac{1}{18} \\ \frac{5}{12} & \frac{1}{6} & \frac{5}{24} & \frac{5}{24} \end{pmatrix}$$

Question 45

Il suffit de rajouter un post-traitement :

```python
def fromAlignement_pseudo(self, align):
        lettres = {A, C, G, T}
        q = len(align)
        p = len(align[0])
        trans = [[0] * p for _ in range(q)]
        trans[p-1][p-1] = 1
        emission = [{A : 0, C : 0, G : 0, T : 0}]
        first = [0] * p
        first[0] = 1
        for j in range(p):
            total = 0
            for i in range(q):
                if align[i][j] in lettres:
                    emission[j][align[i][j]] += 1
                    total += 1
                if align[i][j] in lettres:
                    next = j+1
                    while align[i][next] not in lettres:
                        next += 1
                    trans[j][next] += 1/total
            for c in lettres:
                emission[j][c] /= total

```

```
24          for i in range(p):
25              zero_character = 0
26              for c in lettres:
27                  if emission[i][c] == 0:
28                      zero_character += 1
29              for c in lettres:
30                  if emission[i][c] > 0:
31                      emission[i][c] *= (n-1)/n
32                  else:
33                      emission[i][c] = 1/(n * zero_character)
34
35          return HMM(trans, emission, first)
```

9. Les zones mal alignées

Question 46

On obtient le modèle en Fig. 3, où les états 2 et 3 ont été fusionnés en le nouvel état 2.

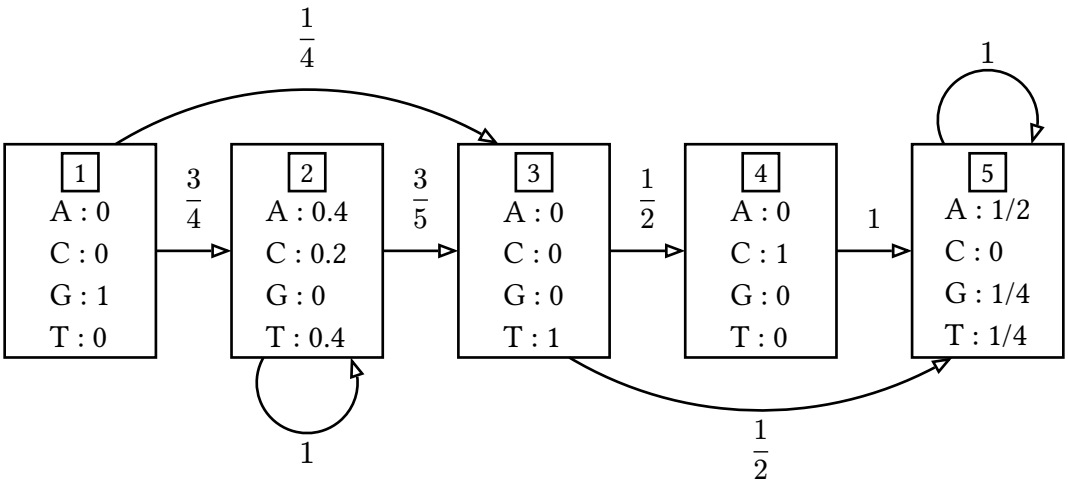

Fig. 3. – Représentation de H_2 modifié

Question 47

On peut commencer par déterminer tous les indices de colonnes mal alignées, ce qui nous donne un tableau de booléens, qu'on calcule au pire cas en $O(|\Sigma| \cdot n)$, en parcourant entièrement le tableau d'émission.

Par exemple, pour le modèle H_2, on obtient le tableau `[False,True,True,False,True,False]`. Ensuite, on peut facilement obtenir la liste voulue. En effet, si on a un `False` à l'indice i, on rajoute l'intervalle (i, i). Si on a un `True`, on cherche le prochain `False`, si leurs indices sont i, j, alors on ajoute l'intervalle $(i, j-1)$. Pour éviter les problèmes dus au fait qu'il n'y ait pas de `False` ensuite, on peut en rajouter un à la fin de la liste, puis retirer l'intervalle qui sera ajouté en trop.

Cette opération est elle en $O(n)$, donc la complexité totale est en $O(|\Sigma| \cdot k)$.

Question 48

On obtient les fonctions suivantes:

```python
def to_bool_misaligned(align):
    n = len(align[0])
    p = len(align)
    b = [False] * H.n
    for i in range(n):
        for j in range(p):
            if align[p][i] == "_":
                b[i] = True
    return b

def intervals(b):
    intervals_list = []
    b.append(False)
    n = len(b)
    i = 0
    while i < n:
        if not(b[i]):
            intervals_list.append((i,i))
            i += 1
        else:
            j = i+1
            while b[j]:
                j += 1
            intervals_list.append((i,j-1))
            i = j
    return intervals_list[:-1]
```