Projet de chaînes de Markov

Ugo Devoille & Zoé Joubert
Référent : Rabehasaina Adrianjatovo Landy-Nirina

Contents

1 Introduction

Le sujet de ce projet de chaîne de Markov concerne les mouvement de foules, ainsi que la modélisation par chaîne de Markov cachée.

En voici l'énoncé :

Une foule composée de N personnes est répartie entre deux pièces A et B. On note N_A^n le nombre de personnes dans la pièce A à l'instant n, de sorte qu'il y a $N_B^n = N - N_A^n$ personnes dans la pièce B, et on note $Y_n = (N_A^n, N_B^n)$. Deux capteurs associés à des caméras situées dans chacune des pièces tentent de compter le nombre de personnes dans chaque pièce: on suppose que chacune des personnes d'une pièce donnée à une probabilité $p \in]0, 1[$ d'être prise en compte dans le comptage. On suppose que les instants d'observations sont très peu espacés, de sorte qu'entre deux instants n et n+1, une seule personne piochée uniformément parmi les N personnes totale change de salle.

2 Modélisation

Décrivons maintenant le modèle de chaîne de Markov cachée associée. On pose $X_n = (M_A^n, M_B^n)$ le nombre d'individus observés par les caméras en salle A et B. On suppose que X_n est une chaîne de Markov au même titre que Y_n, et que l'état du processus observé à l'instant n+1 ne dépend que de l'état de la chaîne à l'instant n.

Pour mieux se représenter le modèle, il est possible faire un schéma de la situation.

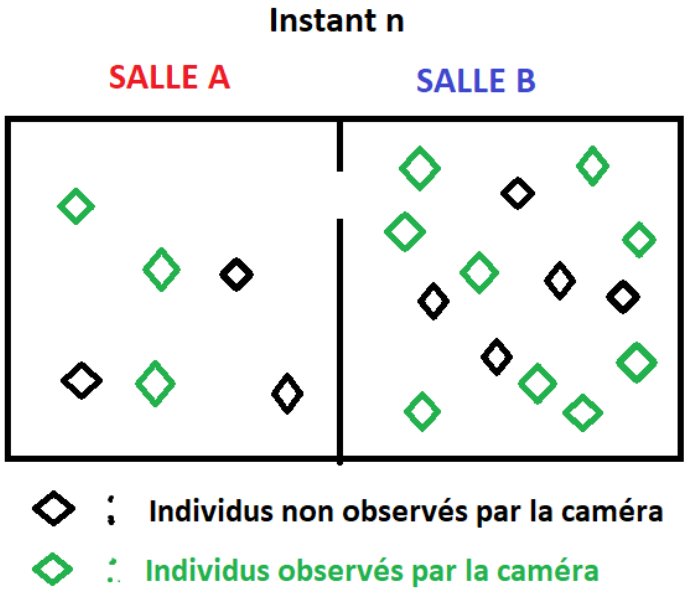

Figure 1: Schéma représentatif du modèle

Sur le schéma donné, on peut observer dans un premier temps que la somme des croix vertes et noires donne N, le nombre total d'individus. A chaque instant n, un individu choisi uniformément va changer de pièce. Cela signifie que par exemple, si 1 personne se trouve en pièce A et 99 se trouvent en pièce B, alors il y a beaucoup plus de chance de voir une personne passer de la pièce B à la pièce A, que l'inverse.

On peut même dire exactement que sur le schéma, on a N=20, et à l'instant n, on a $Y_n = (6, 14)$, et $X_n = (3, 9)$.

Ainsi, le processus $Y_n = (N_A^n, N_B^n)$ est une chaîne de Markov, car le nombre de personnes présentes dans les pièces ne dépend que de l'instant précédent (On notera que c'est une modélisation discutable ici, car ce processus ne dépend pas uniquement de l'instant précédent en théorie...).

L'espace d'état de Y_n est ici $E = \{0, ..., N\}^2$, car il y a dans chacune des pièces entre 0 et N personnes. Cependant, on peut noter que la somme des personnes dans les deux pièces est forcément de N, ainsi l'espace d'état peut

être réduit à $E = \big\{(0,N) \times (1,N-1) \times ... \times (N-1,1) \times (N,0)\big\}$.

Le code Scilab permettant la création de cet espace d'état est le suivant :

```
1  E=[0:N;N:-1:0]
```

Puisqu'il est compliqué de modéliser des couples sur Scilab, on fait le choix ici de créer une matrice de taille 2x(N+1), ou la taille 2 permet de représenter un couple. La création de cet espace d'état se fait assez simplement.

Travailler avec cet espace d'état sera beaucoup plus simple pour la suite du projet, car celui ci est beaucoup plus petit que le précédent (de cardinal N au lieu de N^2). Une autre idée serait de ne considérer que l'une des deux variables (N_A^n par exemple) puisque l'autre dépend de celle ci.

A partir de ces informations, il est possible de tracer le graphe Markovien de Y_n :

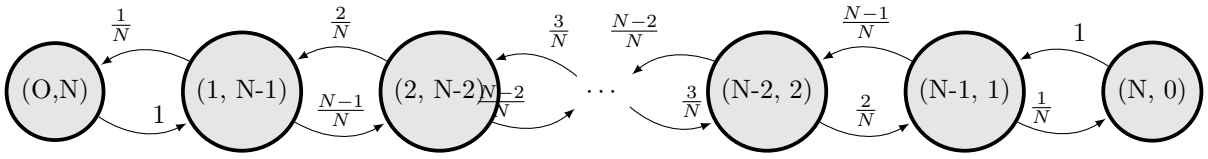

Figure 2: Graphe markovien de $Y_n = (N_A^n, N_B^n)$

La matrice associée à cette chaîne de Markov est :

$$P = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ \frac{1}{N} & 0 & \frac{N-1}{N} & \ddots & \vdots \\ 0 & \frac{2}{N} & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \frac{1}{N} \\ 0 & \cdots & 0 & 1 & 0 \end{pmatrix} = \frac{1}{N} \begin{pmatrix} 0 & N & 0 & \cdots & 0 \\ 1 & 0 & N-1 & \ddots & \vdots \\ 0 & 2 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 1 \\ 0 & \cdots & 0 & N & 0 \end{pmatrix}$$

Le code scilab ci-dessous est celui qui permet la création de la matrice P. En fournissant N au préalable. Celui ci est très court car des commandes permettent de créer des matrices avec des diagonales sans souci.

```
1  P=(1/N)*(diag(1:N,-1)+diag(N:-1:1,1))
```

Voici les sorties que retourne le logiciel Scilab lorsqu'on crée les matrices P pour N=4, puis N=6 :

```
P  =

   0.     1.     0.     0.     0.
   0.25   0.     0.75   0.     0.
   0.     0.5    0.     0.5    0.
   0.     0.     0.75   0.     0.25
   0.     0.     0.     1.     0.
```

Figure 3: Matrice P pour N=4

```
P  =

  0.          1.          0.          0.          0.          0.          0.
  0.1666667   0.          0.8333333   0.          0.          0.          0.
  0.          0.3333333   0.          0.6666667   0.          0.          0.
  0.          0.          0.5         0.          0.5         0.          0.
  0.          0.          0.          0.6666667   0.          0.3333333   0.
  0.          0.          0.          0.          0.8333333   0.          0.1666667
  0.          0.          0.          0.          0.          1.          0.
```

Figure 4: Matrice P pour N=6

Pour revenir au processus X_n, il est important de remarquer que l'espace d'état est ici $O = \{0, ..., N\}^2$, où, contrairement à E, la somme des membres du couple ne fait pas nécessairement N. Cependant, un détail très important est de noter que la somme de ce couple ne peut pas dépasser N, puisque dans le cas où les caméras remarquent chaque individu, on obtient N, et qu'il est impossible d'aller au delà.
Ainsi, on a :

$$O = \{i, j \in \{0, ..., N\} \ tq \ i + j \leq N\}$$

Nous pouvons créer cet espace d'état sur Scilab avec le code suivant :

```
O=zeros(nbcase(N), 2);
Obis=zeros((N+1)^2,2);
for(i=0:N)
    Obis(((N+1)*i+1):((N+1)*i+N+1),1)=linspace(i,i,N+1);
    Obis(((N+1)*i+1):((N+1)*i+N+1),2)=[0:N];
end
k=1
for(i=1:size(Obis)(1))
    if(Obis(i,1)+Obis(i,2)<=N)
        O(k,1)=Obis(i,1);
        O(k,2)=Obis(i,2)
        k=k+1
    end
end
```

Cet espace d'état a été codé en utilisant d'abord une fonction qui calcule son cardinal (la récursivité marchait très bien ici), on a ensuite créé un vecteur qui prenait tous les couples dans $\{0, ..., N\}^2$, et on a utilisé une condition "si" qui ne prenait donc que les couples dont la somme était inférieure ou égale à N.
Le code n'est sûrement pas très bien optimisé, mais nous n'avons pas trouvé plus compact... L'important est d'obtenir quelque chose de valide.

On remarque aussi que $M_A^n \sim B(N_A^n, p)$ et $M_B^n \sim B(N_B^n, p)$, comme l'explique l'énoncé. Cette notion permet donc de faire le lien entre le processus X_n et le processus Y_n, ce qui va s'avérer très utile pour le travail suivant.

Il s'agit ensuite de calculer les coefficients de la matrice :

$$e = e(x|y) := P(X_n = x|Y_n = y), \quad x \in O, \ y \in E$$

Pour calculer les coefficients de cette matrice, la meilleure méthode reste la théorie. Ainsi, on a :

$$\begin{aligned}
P(X_n = x|\ Y_n = y) &= P(X_n = (x_A, x_B)|\ Y_n = (y_A, y_B)) \\
&= P(M_A^n = x_A,\ M_B^n = x_B|\ N_A^n = y_A,\ N_B^n = y_B) \\
&= P(B(N_A^n, p) = x_A,\ B(N_B^n, p) = x_B|\ N_A^n = y_A,\ N_B^n = y_B) \\
&= P(B(y_A, p) = x_A,\ B(y_B, p) = x_B|\ N_A^n = y_A,\ N_B^n = y_B) \\
&= \left(\binom{y_A}{x_A} p^{x_A}(1-p)^{y_A - x_A}\right)\left(\binom{y_B}{x_B} p^{x_B}(1-p)^{y_B - x_B}\right)
\end{aligned}$$

La formule finale permet donc de calculer facilement les coefficients de la matrice e. On notera que l'écriture "$M_A^n = B(N_A^n, p)$" pour passer de la ligne 2 à la ligne 3 est un peu approximative, mais elle permet de comprendre au mieux l'idée donnée pour le calcul.

Le code permettant la création de cette matrice nécessite au préalable la création d'une fonction permettant de calculer k parmi n. On rempli ensuite chaque coefficient de cette matrice grâce à la formule calculée ci-dessus.

```
1  function res=bin(k, n)
2      if(k<=n)
3          res=prod(1:n)/(prod(1:k)*prod(1:(n-k)));
4      else
5          res=0;
6      end
7  endfunction
8
9  e=zeros(size(O)(1), size(E)(2));
10 for(i=1:size(O)(1))
11     for(j=1:size(E)(2))
12         xA=O(i,1);
13         xB=O(i,2);
14         yA=E(1,j);
15         yB=E(2,j);
16         e(i,j)=bin(xA,yA)*p^xA*(1-p)^(yA-xA)*bin(xB,yB)*p^xB*(1-p)^(yB-xB);
17     end
18 end
```

Une fois implémenté dans le logiciel Scilab, il est finalement possible d'observer à quoi ressemble cette matrice pour N=3, et p=$\frac{3}{4}$ (On rappelle que p est la probabilité qu'une caméra observe un individu).

```
e  =

  0.015625   0.015625   0.015625   0.015625
  0.140625   0.09375    0.046875   0.
  0.421875   0.140625   0.         0.
  0.421875   0.         0.         0.
  0.         0.046875   0.09375    0.140625
  0.         0.28125    0.28125    0.
  0.         0.421875   0.         0.
  0.         0.         0.140625   0.421875
  0.         0.         0.421875   0.
  0.         0.         0.         0.421875
```

Figure 5: Matrice e pour N=3 et p=$\frac{3}{4}$

On notera que la somme de chaque colonne pour cette matrice donne 1. C'est assez facilement vérifiable pour la matrice ci-dessus. Cette propriété vient de la formule des probabilités totales.

Enfin, on définit la loi initiale de X_1 par :

$$\pi = \begin{pmatrix} \frac{1}{N+1} & \cdots & \frac{1}{N+1} \end{pmatrix}$$

Celle ci se traduit par le fait que les N individus sont répartis au départ de manière purement aléatoire; chaque cas a autant de chance d'arriver.
Le code Scilab permettant la création de ce vecteur est :

```
1  pi = linspace(1/(N+1),1/(N+1),N+1)
```

3 Implémentation des fonctions

Nous allons donc maintenant donner l'implémentation de l'algorithme Forward et l'algorithme de Viterbi pour ce modèle de chaîne de Markov cachée.

Commencons avec l'algorithme Forward, dont le but est de déterminer $P(X_1 = x_1, ..., X_L = x_L)$ pour $x_1, ..., x_L$ dans O donné, de manière efficace. La procédure Forward consiste à calculer (et implémenter) toutes ces fonctions $\alpha_k(.), k = 1, ..., L$, où $\alpha_k(.)$ définis par :

$$\alpha_1(y) = e(x_1|y)\pi(y)$$

$$\alpha_k(y) = e(x_{k+1}|y) \sum_{z \in E} P(z, y)\alpha_k(y).$$

On a alors :

$$P(X_1 = x_1, ..., X_L = x_L) = \sum_{y \in E} \alpha_L(y).$$

On notera que la méthode ne nécessite que $Card(E)^2 \times$L termes à calculer, ce qui est représente un temps quadratique très avantageux.
A partir de ces informations, voici le code Scilab que nous avons pu implémenter:

```
1  function ret=alpha(x, L)
2      ret=0
3      for y=1:N+1
4          if L==1 then
5              res=pi(y)*e(x(1),y)
6          else
7              res=0
8              for i=1:N+1
9                  res=res+(delta(x, i, L-1)*P(i,y))
10             end
11             res=res*e(x(L),y)
12         end
13         ret=ret+res
14     end
15 endfunction
```

Nous pouvons ensuite passer à l'algorithme de Viterbi. Voici comment il se construit :
En posant $x_i = (x_A^i, x_B^i)$ l'état de X_n à l'instant i, et $Y_i = (Y_A^i, Y_B^i)$ l'état de Y_n à l'instant i, le but de cet algorithme et de déterminer :

$$(y_1^*, ..., y_L^*) = \underset{(y_1,...,y_L) \in E^L}{\operatorname{argmax}} P(Y_1 = y_1, ..., Y_L = y_L | X_1 = x_1, ..., X_L = x_L)$$

où L est un entier fixé.
$(y_1^*, ..., y_L^*)$ correspond aux L couples successifs d'individus se trouvant dans chaque salle, optimisés de sorte que la probabilité de ces L couples sachant les L couples successifs $(x_1, ..., x_L)$ soit maximale.

On fixe x = $(x_1, ..., x_L)$ et on note, pour tout $y \in E$,

$$\delta_k(y) = \delta_k(x, y) = \max_{y_1,...,y_{k-1}} P(Y_1 = y_1, ..., Y_{k-1} = y_{k-1}, Y_k = y, X_1 = x_1, ..., X_k = x_k)$$

$$\psi_{k+1}(y) = \psi_{k+1}(x, y) = \underset{z \in E}{\operatorname{argmax}} \left[\delta_k(z)p(z, y)\right]$$

Finalement, ces définitions nous permettent d'obtenir la relation :

$$\delta_{k+1}(y) = \max_{z \in E} \left[\delta_k(z)p(z, y)\right] e(x_{k+1}|y)$$

Enfin, la dernière étape de l'algorithme consistera à calculer y^* qui est encore une fois défini de manière récursive par:

$$y_k^* = \psi_{k+1}(y_{k+1}^*), \quad k = 2, ..., L-1$$

$$y_L^* = \underset{y \in E}{\operatorname{argmax}}\, \delta_L(y)$$

Les fonctions correspondant à l'algorithme Forward et celui de Viterbi on été vues en TP précédemment, mais celles ci n'étaient adaptées que pour L=4, et elles manquaient aussi d'optimisation.
Ces fonctions ont par conséquent été recodées afin de pouvoir entrer un vecteur x qui soit de taille $L \in \mathbf{N}$. Les fonctions sont désormais récursives, un processus toujours élégant par sa conceptualisation.

```
1  // Fonction delta
2
3  function ret=delta(x, y, L)
4      if L==1 then
5          res=pi(y)*e(x(1),y)
6      else
7          for z=1:N+1
8              u(z)=delta(x, z, L-1)*P(z,y)
9          end
10         res=max(u)*e(x(L),y)
11     end
12     ret=res
13 endfunction
14
15 // Fonction Psi
16
17 function res=psi(x, y, L)
18     for z=1:N+1
19         u(z)=delta(x, z, L-1)*P(z,y)
20     end
21     [a, res]=max(u)
22 endfunction
23
24 // Fonctions Yet
25
26 function res=yLet(x, L)
27     if L==length(x) then
28         for y=1:N+1
29             u(y)=delta(x, y, L)
30         end
31         [a, b]=max(u)
32     else
33         b = psi(x,yLet(x, L+1), L+1)
34     end
35     res=b
36 endfunction
37
38 function res=yet(x)
39     res=1:length(x)
40     for k=1:length(x)
41         res(k) = yLet(x, k)
42     end
43 endfunction
```

Le code se divise en trois parties. La première renvoie les fonctions delta telles qu'on les as définies, la seconde fait de même avec les fonctions psi, et enfin la troisième renvoi le vecteur y optimal, c'est-à-dire y^*.

Avec ce code, il va donc être possible de répondre aux deux prochaines questions qui correspondent à des applications des deux algorithmes respectivement.

4 Première Application

Le but de la troisième question est de déterminer la probabilité d'une séquence $(X_n)_{n=0,\dots,L}$. C'est précisément ce qui est calculé avec l'algorithme Forward. Nous allons donc utiliser le code Scilab associé à différentes séquences.

On distingue pour cela plusieurs cas pour N et p. On va prendre N=10 (cas N grand) et N=5 (cas N petit), et $p = \frac{4}{5}$ (cas caméras relativement fiables) et $p = \frac{1}{2}$ (cas caméras peu fiables).

- Cas N=10, $p = \frac{4}{5}$:

  $x = \big((0,3),(0,2),(1,0),(0,0),(0,7),(0,1)\big) \Longrightarrow \alpha_1(x) = 0.0001966, \alpha_2(x) = 9.026\times10^{-9}, \alpha_3(x) = 7.68\times10^{-15}.$

  On constate des valeurs très proches de 0, qui décroissent quand on augmente k qui va de 1 à L. On choisit donc de ne représenter que les premiers $\alpha_k(x)$ puisqu'on trouve vite très peu d'intérêt à le faire ensuite.

- Cas N=10, $p = \frac{1}{2}$:

  $$x = \big((0,3),(0,2),(1,0),(0,0),(0,7),(0,1)\big) \Longrightarrow \alpha_1(x) = 0.0292969, \alpha_2(x) = 0.0008017, \alpha_3(x) = 0.0000016,$$

  $$\alpha_4(x) = 1.445 \times 10^{-9}.$$

  On remarque encore une fois que les $\alpha_k(x)$ tendent vers 0 très rapidement.

- Cas N=5, $p = \frac{4}{5}$:

  $$x = \big((0,0),(0,3),(1,0),(1,4),(0,2),(1,2)\big) \Longrightarrow \alpha_1(x) = 0.00032, \alpha_2(x) = 0.0000098, \alpha_3(x) = 1.42 \times 10^{-8}.$$

  Encore une fois c'est la même chose, on a des valeurs très proches de 0 qui décroissent très vite lorsque l'on augmente k.

- Cas N=5, $p = \frac{1}{2}$:

  $$x = \big((0,0),(0,3),(1,0),(1,4),(0,2),(1,2)\big) \Longrightarrow \alpha_1(x) = 0.03125, \alpha_2(x) = 0.0014648, \alpha_3(x) = 0.0000517.$$

  $$\alpha_4(x) = 0.0000004, \alpha_5(x) = 5.595 \times 10^{-8}.$$

  Ici aussi, les $\alpha_k(x)$ tendent très vite vers 0 lorsque k augmente.

Il est aussi possible de faire la même manipulation dans des cas plus réalistes. Ainsi, nous allons faire un essai avec N grand et p proche de 1.

- Cas N=50, $p = \frac{19}{20}$:

  $$\big((25,24),(26,18),(25,22)\big) \Longrightarrow \alpha_1(x) = 0.0040497, \alpha_2(x) = 0.0000017, \alpha_3(x) = 0.0000001.$$

On constate des probabilités plus hautes que précédemment ici, malgré un N plus grand. En effet, avoir p proche de 1 permet d'obtenir des probabilités plus élevées. De plus, le premier couple est assez probable par sa définition.

5 Seconde Application

Enfin, la dernière application est sans doute la plus importante. Le but est, à partir d'une séquence observée $(X_n)_{n=0,\dots,L}$ de l'observation de la foule entre les instants 0 et L, de prédire les nombres successifs de personnes présentes dans les pièces A et B entre les instants 0 et L.

Pour résoudre cette problématique, nous allons utiliser l'algorithme de Viterbi implémenté précédemment.

Il est très important de noter qu'il faut bien choisir les séquences observées $(X_n)_{n=0,\dots,L}$. En effet, certaines combinaisons n'ont pas lieu d'être et ainsi l'algorithme peut retourner de mauvais résultats.

En voici un exemple :

On prend N=10, $p = \frac{4}{5}$.

$$x = \big((9,1),(10,0),(6,3),(5,4),(6,0),(5,0)\big) \Longrightarrow y^* = \big((0,10),(0,10),(0,10),(0,10),(0,10),(0,10)\big).$$

On voit tout de suite que le y^* obtenu est problématique puisqu'on sait qu'à chaque instant n une personne parmi les N personnes change de pièce. C'est impossible que personne ne bouge...
Le problème vient de la séquence choisie pour x qui n'est pas réaliste. Le 2ème couple est (10,0), cela signifie que les caméras détectent bien toutes les personnes dans chaque pièce et il est logique que l'unique état possible ensuite est (9,1). Le fait de mettre (6,3) n'est donc pas cohérent et pose problème pour le calcul de y^*.
Il ne faut donc pas mettre dans la séquence des couples qui ne peuvent pas se suivre en prenant en compte le fait que si une caméra détecte k personnes dans une pièce alors il y avait au moins k personnes dans cette pièce.

Nous avons vu à travers cet exemple que le choix de la séquence x doit être réfléchi et c'est ce que nous avons fait pour la suite des applications.

- Cas N=10, $p = \frac{4}{5}$:

$$x = \big((9,1),(10,0),(9,1),(7,2),(4,1),(2,4)\big) \Longrightarrow y^* = \big((9,1),(10,0),(9,1),(8,2),(7,3),(6,4)\big).$$

- Cas N=10, $p = \frac{1}{2}$:

$$x = \big((9,1),(10,0),(9,1),(7,2),(4,1),(2,4)\big) \Longrightarrow y^* = \big((9,1),(10,0),(9,1),(8,2),(7,3),(6,4)\big).$$

  On note que le fait de changer p ne change pas y^*.

- Cas N=5, $p = \frac{8}{10}$:

$$x = \big((2,3),(3,2),(0,1),(2,2),(2,3),(1,4)\big) \Longrightarrow y^* = \big((2,3),(3,2),(2,3),(3,2),(2,3),(1,4)\big).$$

  Une nouvelle fois, on peut faire la même manipulation avec N grand et p proche de 1.

- Cas N=30, $p = \frac{9}{10}$:

$$x = \big((2,15),(2,18),(3,26),(3,18)\big) \Longrightarrow y^* = \big((2,28),(3,27),(4,26),(5,25)\big)$$

$$x = \big((17,13),(18,12),(18,5)\big) \Longrightarrow y^* = \big((17,13),(18,12),(19,11)\big).$$

- Cas N=50, $p = \frac{19}{20}$:

$$x = \big((25,24),(26,18),(25,22)\big) \Longrightarrow y^* = \big((26,24),(27,23),(26,24)\big).$$

  Tout ces résultats semblent cohérents !

6 Conclusion

Le projet obtenu s'est avéré finalement complexe sur plusieurs points.
Dans un premier temps, la modélisation qui a pu être produite à partir de l'énoncé n'avait rien de facile; les espaces d'états qui étaient différents étaient par exemple complexe à mettre en relation, et c'est notamment la création de la matrice e qui demandait une certaines réflexion.
Une seconde étape de difficulté était aussi l'implémentation du code. Créer les espaces d'états qui étaient constitués de couples demandait de passer par une solution annexe, et encore une fois, la construction de la matrice e n'avait rien de trivial.
Ensuite, l'implémentation des différents algorithmes demandait moins de réflexion puisque ceux ci étaient déjà créé dans un précédent TP. Il fallait tout de même optimiser le code et le rendre récursif, obtenant ainsi des fonctions bien plus utiles.
Finalement, la dernière étape à permit de mettre en lumière tout le travail fait précédemment. Les résultats ont été concluants et nous ont permis de remarquer quelques propriétés remarquables. En bref, le sujet à été traité dans son ensemble.

Pour nous, ce projet aura finalement été très enrichissant. La modélisation était super intéressante, par sa complexité et les petites nuances qui pouvaient tout changer. Ce fut un grand soulagement de voir l'application fonctionner.