

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique



Université des Sciences et de la Technologie Houari Boumédiène

Faculté d'Informatique
Département d'Intelligence Artificielle

Master 2 Systèmes Informatiques intelligents

Module : Représentation des connaissances II

Rapports des TPs

Réalisé par :
BOUROUINA Rania, 181831052716
CHIBANE Ilies, 181831072041

Table des matières

1 TP1 : Logique des fonctions de croyances	1
1 introduction	1
2 Exercice 4	1
2.1 Modélisation des connaissances	1
2.2 Solution de l'exercice 4	5
3 Exercice 5	6
3.1 Modélisation des connaissances	6
3.2 Conclusion	10
2 TP2 : Contrôleurs flous	11
1 introduction	11
2 Étapes de la conception d'un contrôleur flou	11
3 Définition des E/S du contrôleur	11
4 Subdivision des E/S en sous ensembles flou (Fuzzification)	12
5 Définition de la base de règle floue	15
6 Application de la méthode d'inférence	17
7 Application de la défuzzification	18
8 Conclusion	19
3 TP3 : Réseaux Bayésiens	20
1 Introduction	20
2 ÉTAPE 1 : Installation de PGMPY	20
3 ÉTAPE 2 : Poly Arbre	21
3.1 Nos composantes	21
3.2 Solution	22
4 ÉTAPE 3 : Conception d'un erbre multi-connected	27
4.1 Nos composantes	27
4.2 Solution	28
4 TP4 : Théorie des possibiliste	33
1 introduction	33
2 Création de notre base de connaissance	33
3 Traitement de notre base de connaissance	35
4 Création et exécution de notre algorithme	36
5 Conclusion	37
5 TP5 : Inférence logique et propagation graphique en théorie des possibilités	38
1 Utilisation de la Toolbox	38
1.1 L'utilisation de Cygwin	38
1.2 Une évidence	39

1.3	Deux evidences	40
-----	--------------------------	----

Introduction Générale

La représentation des connaissances désigne un ensemble d'outils et de procédés destinés d'une part à représenter et d'autre part à organiser le savoir humain pour l'utiliser et le partager. Il s'agit d'un domaine de l'intelligence artificielle (IA) consacré à la représentation des informations sur le monde sous une forme qu'un système informatique peut utiliser pour résoudre des tâches complexes. Dans cette série de TPs, nous allons voir comment peut-on faire ceci tout en découvrant les différentes logiques et les types de raisonneurs qui font des déductions dans ces dernières.

TP1 : Logique des fonctions de croyances

1 introduction

Dans ce premier TP qui concerne la logique des fonctions de croyances, nous avons utilisé **pyds** : Une bibliothèque Python pour effectuer des calculs dans la théorie de Dempster-Shafer, Cette librairie est disponible sur le repository Github suivant : pdys de python et nous l'avons par la suite utilisé pour résoudre l'exercice 4 et 5 de la série de TD.

La théorie de Dempster-Shafer est une théorie mathématique basée sur la notion des preuves en utilisant les fonctions de croyance et le raisonnement plausible. Le but de cette théorie est de permettre de combiner des preuves distinctes afin de raisonner dans une situation incertaine. Cette théorie a été développée par Arthur P.Dempster et Glenn Shafer.

2 Exercice 4

Nous allons résoudre l'exercice 4 en utilisant la bibliothèque présentée précédemment.

Exercice 4:

Trois experts analysent les causes de la pollution de l'air.

Le premier expert atteste que la combustion des énergies fossiles est responsable à 40%, les moyens de transport à 25% et les centrales thermiques à 15%.

Le second expert affirme que les moyens de transport y contribuent à 75%.

Le troisième expert pense que les transports sont responsables à 35%, les centrales thermiques à 50%, le chauffage au bois à 18%. Il pense aussi que la pollution peut avoir une origine naturelle à 2%.

- a- Représentez ces connaissances en utilisant la théorie de Dempster-Shafer.
- b- Dans le cas du premier expert, calculez les degrés de croyance et de plausibilité.
- c- Comment prendre en compte ces différents indices afin de définir les causes de la pollution de l'air. Explicitez chaque étape. Que peut-on conclure?

FIGURE 1.1 – Enoncé de l'exercice 4

2.1 Modélisation des connaissances

Voici les symboles que nous allons utiliser pour faire notre modélisation :

- F : La combustion des énergies fossiles.
- M : Moyens de transports.
- C : Centrales thermiques.
- B : Chauffage au bois.
- N : Origine naturelle.

1- Représentation des fonctions de masse pour chaque expert :

```
● ● ●  
1 from pyds import MassFunction  
2  
3  
4 #affichage des masses pour chaque expert  
5 print('Expert 1')  
6 m1 = MassFunction({'F':0.4, 'M':0.45, 'C':0.15})  
7 print('m_1 =', m1)  
8 print('Expert 2')  
9 m2 = MassFunction({'M':0.75, '':0.25})  
10 print('m_2 =', m2)  
11 print('Expert 3')  
12 m3 = MassFunction({'M':0.35, 'C':0.5, 'B':0.18, 'N':0.02})  
13 print('m_3 =', m3)  
14
```

FIGURE 1.2 – Instruction : Création de fonction de masse

- Résultats :

```
● ● ●  
1 [Running] python -u "d:\SII2\CR\TPS\TP1\main.py"  
2 Expert 1  
3 m_1 = {'M':0.45; 'F':0.4; 'C':0.15}  
4 Expert 2  
5 m_2 = {'M':0.75; set():0.25}  
6 Expert 3  
7 m_3 = {'C':0.5; 'M':0.35; 'B':0.18; 'N':0.02}  
8  
9 [Done] exited with code=0 in 1.714 seconds  
10
```

FIGURE 1.3 – Résultats : Création de fonction de masse

2- Calcul de croyance et Plausibilité pour chaque expert :

```

● ● ●

1 #affichage de croyance et plausibilité pour chaque expert
2 print('Croyance et plausibilité pour l\'expert 1')
3 print('bel_1 =', m1.bel())
4 print('pl_1 =', m1.pl())
5
6 print('Croyance et plausibilité pour l\'expert 2')
7 print('bel_2 =', m2.bel())
8 print('pl_2 =', m2.pl())
9
10 print('Croyance et plausibilité pour l\'expert 3')
11 print('bel_3 =', m3.bel())
12 print('pl_3 =', m3.pl())
13

```

FIGURE 1.4 – Instruction : Calcul de croyance et de plausibilité

- Résultats :

```

● ● ●
1 [Running] python -u "d:\SII2\CR\TPS\TP1\main.py"
2 Croyance et plausibilite pour l'expert 1
3 bel_1 = {frozenset(): 0.0, frozenset({'M'}): 0.45, frozenset({'F'}): 0.4, frozenset({'C'}): 0.15, frozenset({'M', 'F'}): 0.8500000000000001, frozenset({'M', 'C'}): 0.65, frozenset({'F', 'C'}): 0.55, frozenset({'M', 'F', 'C'}): 1.0}
4 pl_1 = {frozenset(): 0.0, frozenset({'M'}): 0.45, frozenset({'F'}): 0.4, frozenset({'C'}): 0.15, frozenset({'M', 'F'}): 0.8500000000000001, frozenset({'M', 'C'}): 0.65, frozenset({'F', 'C'}): 0.55, frozenset({'M', 'F', 'C'}): 1.0}
5
6 [Done] exited with code=0 in 1.682 seconds
7

```

FIGURE 1.5 – Résultats : Calcul de croyance et de plausibilité de l'expert 1

```

● ● ●
1 [Running] python -u "d:\SII2\CR\TPS\TP1\main.py"
2 Croyance et plausibilite pour l'expert 2
3 bel_2 = {frozenset(): 0.0, frozenset({'M'}): 0.75}
4 pl_2 = {frozenset(): 0.0, frozenset({'M'}): 0.75}
5
6 [Done] exited with code=0 in 1.622 seconds
7

```

FIGURE 1.6 – Résultats : Calcul de croyance et de plausibilité de l'expert 2

```

● ● ●
1 [Running] python -u "d:\SII2\CR\TPS\TP1\main.py"
2 Croyance et plausibilite pour l'expert 3
3 bel_3 = {frozenset(): 0.0, frozenset({'M'}): 0.4, frozenset({'F'}): 0.4, frozenset({'C'}): 0.2, frozenset({'M', 'F'}): 0.8, frozenset({'M', 'C'}): 0.6, frozenset({'F', 'C'}): 0.5, frozenset({'M', 'F', 'C'}): 1.0}
4 pl_3 = {frozenset(): 0.0, frozenset({'M'}): 0.4, frozenset({'F'}): 0.4, frozenset({'C'}): 0.2, frozenset({'M', 'F'}): 0.8, frozenset({'M', 'C'}): 0.6, frozenset({'F', 'C'}): 0.5, frozenset({'M', 'F', 'C'}): 1.0}
5
6 [Done] exited with code=0 in 1.622 seconds
7

```

FIGURE 1.7 – Résultats : Calcul de croyance et de plausibilité de l'expert 3

a - Combinaison de Dempster & Shaffer

Cette méthode nous aide à déterminer la raison derrière la pollution

```
● ● ●  
1 #combinaiseons  
2 print('\nDempster shaffer\'s combination rule')  
3 print('Dempster\'s combination rule for m_1 and m_2 =', m1 & m2)  
4 print('Dempster\'s combination rule for m_2 and m_3 =', m2 & m3)  
5 print('Dempster\'s combination rule for m_1 and m_3 =', m1 & m3)  
6  
7 print('Dempster\'s combination rule for m_1, m_2, and m_3 =', m1.combine_conjunctive(m2, m3))  
8 print('Dempster\'s combination rule for m_2, m_1, and m_3 =', m2.combine_conjunctive(m1, m3))  
9 print('Dempster\'s combination rule for m_3, m_1, and m_2 =', m3.combine_conjunctive(m1, m2))
```

FIGURE 1.8 – Combinaison de Dempster & Shaffer

```
● ● ●  
1 [Running] python -u "d:\SII2\CR\TPS\TP1\exos4\main.py"  
2  
3 Dempster shaffer's combination rule  
4 Dempster's combination rule for m_1 and m_2 = {{'M'}:1.0}  
5 Dempster's combination rule for m_2 and m_3 = {{'M'}:1.0}  
6 Dempster's combination rule for m_1 and m_3 = {{'M'}:0.6774193548387097; {'C'}:0.3225806451612903}  
7 Dempster's combination rule for m_1, m_2, and m_3 = {{'M'}:1.0}  
8 Dempster's combination rule for m_2, m_1, and m_3 = {{'M'}:1.0}  
9 Dempster's combination rule for m_3, m_1, and m_2 = {{'M'}:0.6774193548387097; {'C'}:0.3225806451612903}  
10  
11 [Done] exited with code=0 in 0.94 seconds  
12
```

FIGURE 1.9 – Résultat : Combinaison de Dempster & Shaffer

2.2 Solution de l'exercice 4

```

● ● ●

1 #solution exercice 4
2 def exercise4():
3     print('Exercice 4')
4     print('Creation des fonction de masse :')
5
6     print('Expert 1')
7     m1 = MassFunction({'F':0.4, 'M':0.45, 'C':0.15})
8     print('m_1 =', m1)
9
10    print('Expert 2')
11    m2 = MassFunction({'M':0.75, '':0.25})
12    print('m_2 =', m2)
13
14    print('Expert 3')
15    m3 = MassFunction({'M':0.35, 'C':0.5, 'B':0.18, 'N':0.02})
16    print('m_3 =', m3)
17
18    print('\n-----\n')
19    print('Croyance et plausibilité pour expert 1')
20    print('bel_1 =', m1.bel())
21    print('pl_1 =', m1.pl())
22    print('\n-----\n')
23
24

```

FIGURE 1.10 – Solution de l'exercice 4

Résultats :

```

● ● ●
1 [Running] python -u "d:\STUDI\GRVPSVTP\main.py"
2 Exercice 4
3 Creation des fonction de masse :
4 Expert 1
5 m1 = {'M':0.45; 'F':0.4; 'C':0.15}
6 Expert 2
7 m2 = {'M':0.75; set():0.25}
8 Expert 3
9 m3 = {'C':0.5; 'M':0.35; 'B':0.18; 'N':0.02}
10
11
12
13 Croyance et plausibilité pour expert 1
14 bel_1 = (frozenset(): 0.0, frozenset({'F'}): 0.4, frozenset({'M'}): 0.15, frozenset({'C'}): 0.45, frozenset({'F', 'M'}): 0.45, frozenset({'F', 'C'}): 0.15, frozenset({'M', 'C'}): 0.6, frozenset({'F', 'M', 'C'}): 0.6)
15 pl_1 = (frozenset(): 0.0, frozenset({'F'}): 0.4, frozenset({'M'}): 0.25, frozenset({'C'}): 0.5, frozenset({'F', 'M'}): 0.25, frozenset({'F', 'C'}): 0.5, frozenset({'M', 'C'}): 0.75, frozenset({'F', 'M', 'C'}): 1.0)
16
17
18
19
20
21 [Done] exited with code=0 in 1.778 seconds

```

FIGURE 1.11 – Résultat de l'exercice 4

3 Exercice 5

Nous allons résoudre l'exercice 5 en utilisant la bibliothèque présentée précédemment.

Exercice 5 :

Trois experts discutent à propos du bruit présent sur une image. Le premier atteste qu'il est causé lors du processus d'acquisition à 38% et lors de la transmission à 55%. Le second expert affirme qu'il est dû lors du processus de stockage à 88%. Le troisième expert atteste que les différentes hypothèses sont équiprobables.

1. Modélisez ces connaissances à l'aide de la théorie de Dempster & Shafer. Quelles sont les spécificités des différentes distributions de masse.
2. Calculez les degrés de croyance et de plausibilité associés aux différents éléments. Que peut-on conclure ?
3. Calculez les degrés de doute associés aux différents éléments.
4. Que représentent les degrés de croyance et de plausibilité associés aux trois distributions ?
5. Combinez les trois sources d'expertise en explicitant chaque étape. Que pouvez-vous conclure ?

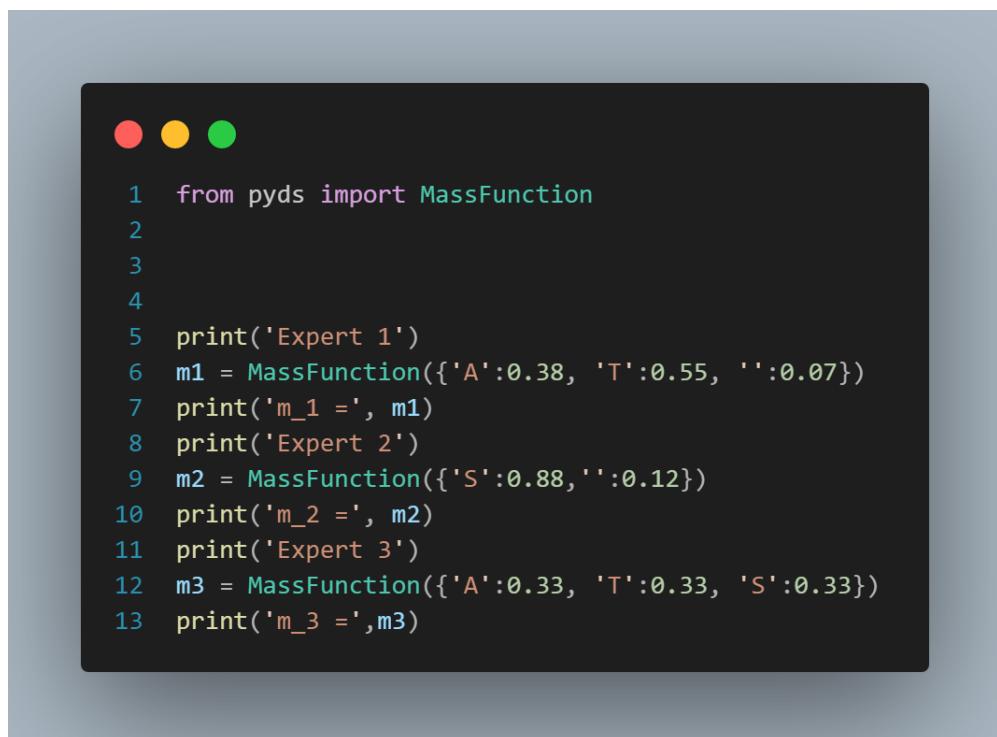
FIGURE 1.12 – Enoncé de l'exercice 5

3.1 Modélisation des connaissances

Voici les symboles que nous allons utiliser pour faire notre modélisation :

- A : Acquisition.
- T : Transmission.
- S : Stockage.

1- Réprésentation des fonctions de masse pour chaque expert :



```
● ● ●

1 from pyds import MassFunction
2
3
4
5 print('Expert 1')
6 m1 = MassFunction({'A':0.38, 'T':0.55, 'S':0.07})
7 print('m_1 =', m1)
8 print('Expert 2')
9 m2 = MassFunction({'S':0.88, 'T':0.12})
10 print('m_2 =', m2)
11 print('Expert 3')
12 m3 = MassFunction({'A':0.33, 'T':0.33, 'S':0.33})
13 print('m_3 =', m3)
```

FIGURE 1.13 – Instruction : Création de fonction de masse

- Résultats :



```
1 [Running] python -u "d:\SII2\CR\TPS\TP1\exo5.py"
2 Expert 1
3 m_1 = {{'T'}:0.55; {'A'}:0.38; set():0.07}
4 Expert 2
5 m_2 = {{'S'}:0.88; set():0.12}
6 Expert 3
7 m_3 = {{'A'}:0.33; {'T'}:0.33; {'S'}:0.33}
8
9 [Done] exited with code=0 in 0.913 seconds
10
```

FIGURE 1.14 – Résultats : Création de fonction de masse

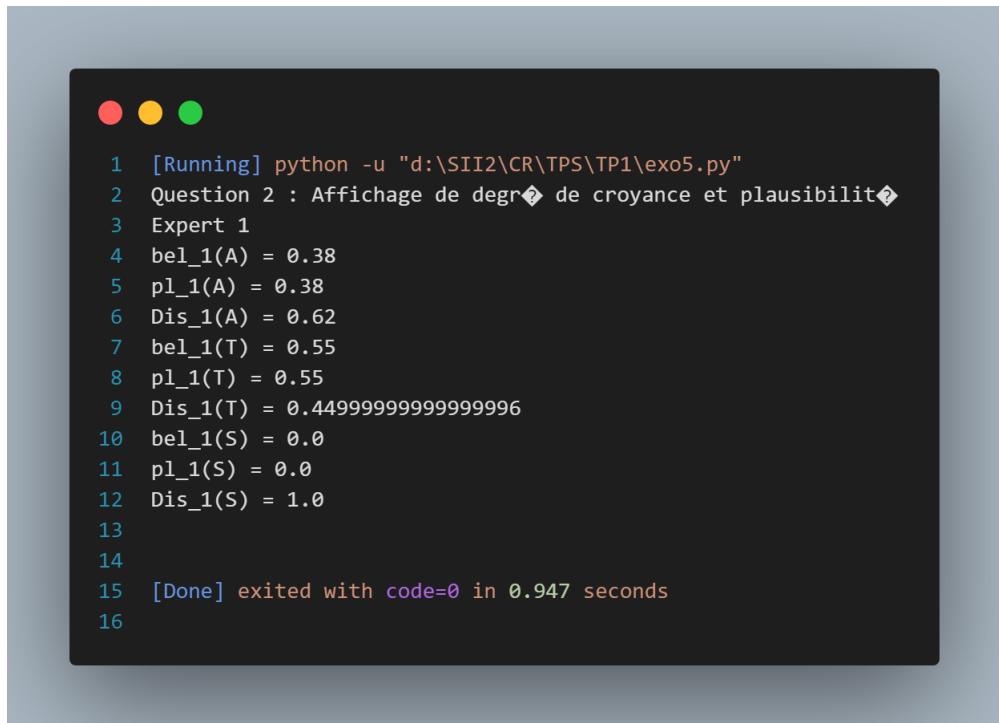
2- Calcul de croyance et Plausibilité pour chaque expert :



```
1 print("Question 2 : Affichage de degré de croyance et plausibilité")
2 print("Expert 1")
3 print('bel_1(A) =', m1.bel({'A'}))
4 print('pl_1(A) =', m1.pl({{'A'}}))
5 print('Dis_1(A) =', 1-m1.pl({{'A'}}))
6 print('bel_1(T) =', m1.bel({'T'}))
7 print('pl_1(T) =', m1.pl({{'T'}}))
8 print('Dis_1(T) =', 1-m1.pl({{'T'}}))
9 print('bel_1(S) =', m1.bel({'S'}))
10 print('pl_1(S) =', m1.pl({{'S'}}))
11 print('Dis_1(S) =', 1-m1.pl({{'S'}}))
12 print()
13
```

FIGURE 1.15 – Instruction : Calcul de croyance et de plausibilité

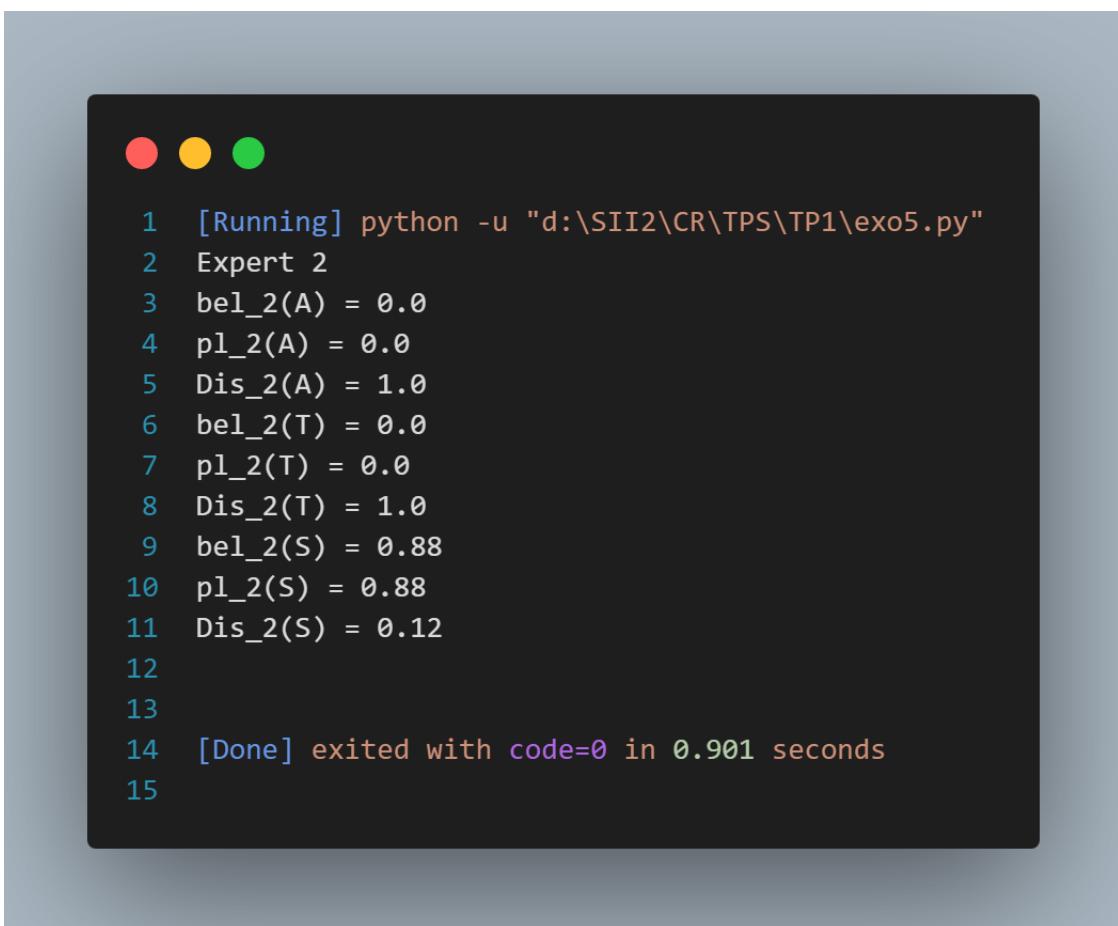
- Résultats :



```
● ● ●

1 [Running] python -u "d:\SII2\CR\TPS\TP1\exo5.py"
2 Question 2 : Affichage de degr  de croyance et plausibilit 
3 Expert 1
4 bel_1(A) = 0.38
5 pl_1(A) = 0.38
6 Dis_1(A) = 0.62
7 bel_1(T) = 0.55
8 pl_1(T) = 0.55
9 Dis_1(T) = 0.4499999999999996
10 bel_1(S) = 0.0
11 pl_1(S) = 0.0
12 Dis_1(S) = 1.0
13
14
15 [Done] exited with code=0 in 0.947 seconds
16
```

FIGURE 1.16 – Résultats : Calcul de croyance et de plausibilité de l’expert 1



```
● ● ●

1 [Running] python -u "d:\SII2\CR\TPS\TP1\exo5.py"
2 Expert 2
3 bel_2(A) = 0.0
4 pl_2(A) = 0.0
5 Dis_2(A) = 1.0
6 bel_2(T) = 0.0
7 pl_2(T) = 0.0
8 Dis_2(T) = 1.0
9 bel_2(S) = 0.88
10 pl_2(S) = 0.88
11 Dis_2(S) = 0.12
12
13
14 [Done] exited with code=0 in 0.901 seconds
15
```

FIGURE 1.17 – Résultats : Calcul de croyance et de plausibilité de l’expert 2



```

1 [Running] python -u "d:\SII2\CR\TPS\TP1\exo5.py"
2 Expert 3
3 bel_3(A) = 0.33
4 pl_3(A) = 0.33
5 Dis_3(A) = 0.6699999999999999
6 bel_3(T) = 0.33
7 pl_3(T) = 0.33
8 Dis_3(T) = 0.6699999999999999
9 bel_3(S) = 0.33
10 pl_3(S) = 0.33
11 Dis_3(S) = 0.6699999999999999
12
13
14 [Done] exited with code=0 in 0.898 seconds

```

FIGURE 1.18 – Résultats : Calcul de croyance et de plausibilité de l’expert 3

a - Combinaison de Dempster & Shaffer

Cette méthode nous aide à déterminer la raison derrière la pollution



```

1 #combinaiseons
2 print('\nDempster shaffer\'s combination rule')
3 print('Dempster\'s combination rule for m_1 and m_2 =', m1 & m2)
4 print('Dempster\'s combination rule for m_2 and m_3 =', m2 & m3)
5 print('Dempster\'s combination rule for m_1 and m_3 =', m1 & m3)
6
7 print('Dempster\'s combination rule for m_1, m_2, and m_3 =', m1.combine_conjunctive(m2, m3))
8 print('Dempster\'s combination rule for m_2, m_1, and m_3 =', m2.combine_conjunctive(m1, m3))
9 print('Dempster\'s combination rule for m_3, m_1, and m_2 =', m3.combine_conjunctive(m1, m2))

```

FIGURE 1.19 – Combinaison de Dempster & Shaffer

```
● ● ●
1 [Running] python -u "d:\SII2\CR\TPS\TP1\exo5.py"
2
3 Dempster shaffer's combination rule
4 Dempster's combination rule for m_1 and m_2 = {}
5 Dempster's combination rule for m_2 and m_3 = {{'S'}:1.0}
6 Dempster's combination rule for m_1 and m_3 = {{'T'}:0.5913978494623655; {'A'}:0.4086021505376344}
7 Dempster's combination rule for m_1, m_2, and m_3 = {}
8 Dempster's combination rule for m_2, m_1, and m_3 = {}
9 Dempster's combination rule for m_3, m_1, and m_2 = {{'T'}:0.5913978494623655; {'A'}:0.4086021505376344}
10
11 [Done] exited with code=0 in 0.944 seconds
12
```

FIGURE 1.20 – Résultat : Combinaison de Dempster & Shaffer

3.2 Conclusion

La théorie de Dempster-Shafer est une théorie de l'incertain qui permet de modéliser la notion de degré de croyance. Elle permet de distinguer l'ignorance de l'incertitude et elle tient en compte plusieurs sources (Par la règle de combinaison de Dempster et Shafer).

TP2 : Contrôleurs flous

1 introduction

Le but de ce et de concevoir et d'implémenter un contrôleur flou. Pour réaliser cela, nous avons utilisé la librairie open source fuzzylogic de python et nous l'avons par la suite utilisé pour résoudre l'exercice 1 de la série de TD.

2 Étapes de la conception d'un contrôleur flou

La conception d'un contrôleur flou consiste en 5 étapes distinctes qui sont :

- Définition des E/S du contrôleur.
- Subdivision des E/S en sous ensembles flou (Fuzzification).
- Définition de la base de règle floue.
- Application de la méthode d'inférence.
- Application de la défuzzification.

Chaque étape sera réaliser et illustrer lors de la suite de ce chapitre.

3 Définition des E/S du contrôleur

En utilisant les données fournies par l'exercice 1 nous définissons les E/S de notre contrôleur. La représentation graphique se traduit comme ceci :



FIGURE 2.1 – Représentation graphique de nos E/S

Afin de définir nos E/S en python, nous utilisons le code suivant :



```
1 # initialisation de notre premier contrôleur d'entrée
2 TC = Domain('Technologie de la cyber securite', 20, 80)
3
4 # initialisation de notre deuxième contrôleur d'entrée
5 NC = Domain('Normes de la cyber securite', 9, 70)
6
7 # initialisation de notre troisième contrôleur d'entrée
8 PI = Domain('Portee de l information', 5, 50)
9
10 # initialisation de notre contrôleur de sortie
11 RC = Domain('Risques de la cybercriminalite ', -80, 70)
```

FIGURE 2.2 – Code Python permettant de définir et d’initialiser nos E/S

Une fois le code exécuté, on obtient nos différentes variables contenant nos E/S.

4 Subdivision des E/S en sous ensembles flou (Fuzzification)

Afin d’opérer la fuzzification, nous allons ajouter les fonctions d’appartenance des différentes variables, TC, NC, PI et RC. le code ainsi que les résultats obtenus sont présentés ci-dessous :



```
1 # Initisation des paramètres d'entrée de TC
2 TC.AV = trapezoid(20, 35, 35, 45)
3 TC.AC = trapezoid(35, 45, 45, 60)
4 TC.IN = trapezoid(45, 60, 60, 80)
5 TC.AV.plot(), TC.AC.plot(), TC.IN.plot()
```

FIGURE 2.3 – Code Python pour créer la fonction d’appartenance de TC

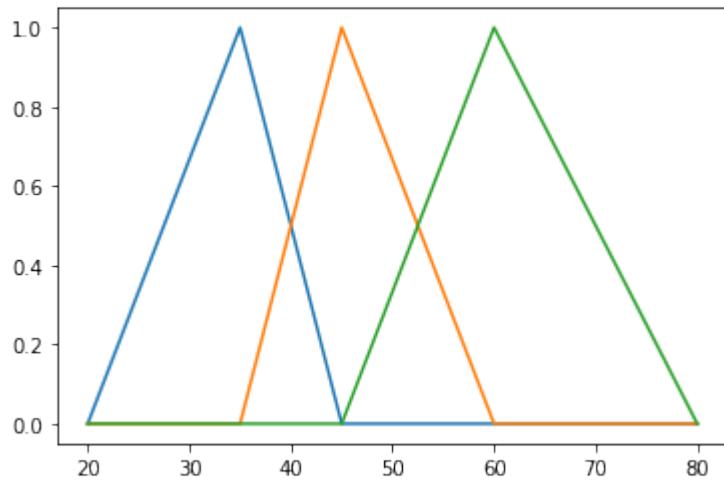


FIGURE 2.4 – Fonction d'appartenance de TC

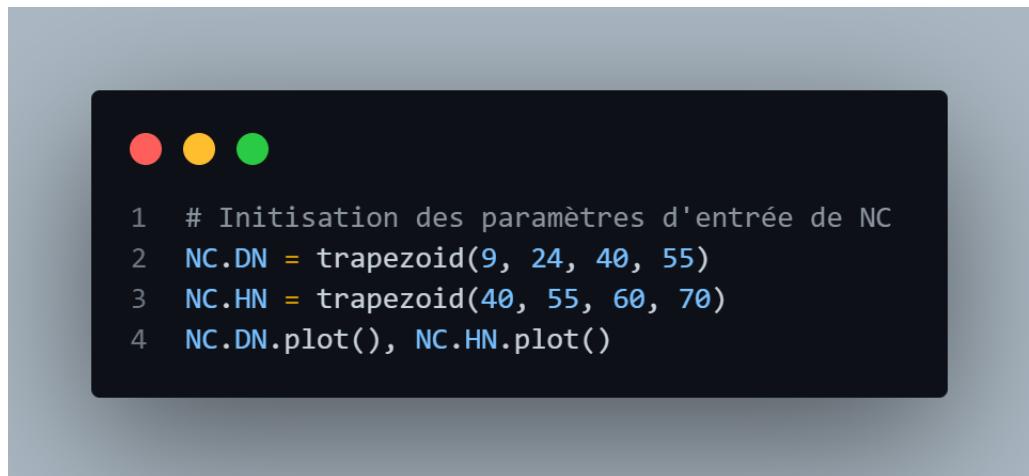


FIGURE 2.5 – Code Python pour créer la fonction d'appartenance de NC

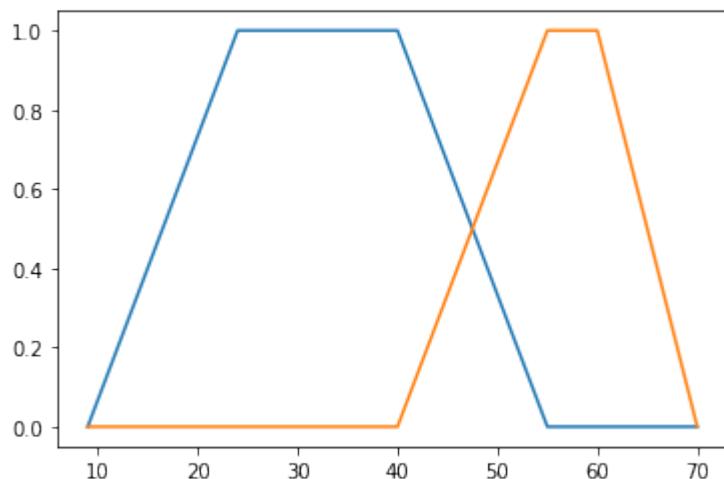


FIGURE 2.6 – Fonction d'appartenance de NC



```
1 # Initisation des paramètres d'entrée de PI
2 PI.TG = trapezoid(5, 10, 15, 20)
3 PI.GR = trapezoid(15, 20, 25, 30)
4 PI.MO = trapezoid(25, 30, 35, 40)
5 PI.FA = trapezoid(35, 40, 45, 50)
6 PI.TG.plot(), PI.GR.plot(), PI.MO.plot(), PI.FA.plot()
```

FIGURE 2.7 – Code Python pour créer la fonction d'appartenance de PI

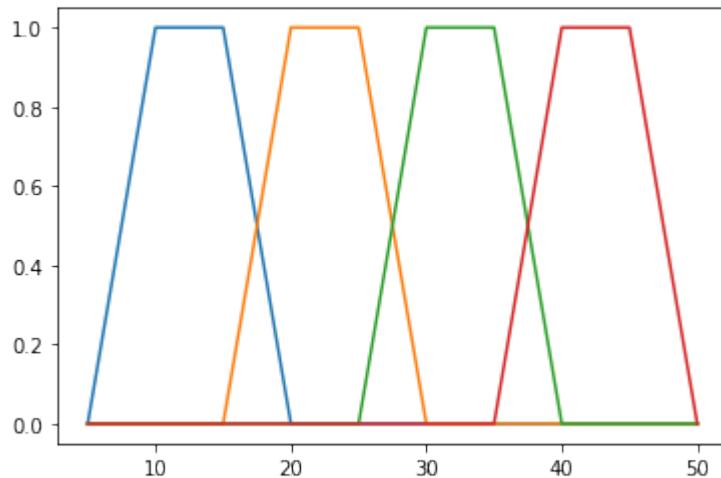


FIGURE 2.8 – Fonction d'appartenance de PI



```
1 # Initisation des paramètres de sortie de RC
2 RC.TF = trapezoid(-80, -50, -50, -10)
3 RC.FO = trapezoid(-50, -10, -10, 10)
4 RC.MO = trapezoid(-10, 10, 10, 40)
5 RC.FA = trapezoid(10, 40, 40, 70)
6 RC.TF.plot(), RC.FO.plot(), RC.MO.plot(), RC.FA.plot()
```

FIGURE 2.9 – Code Python pour créer la fonction d'appartenance de RC

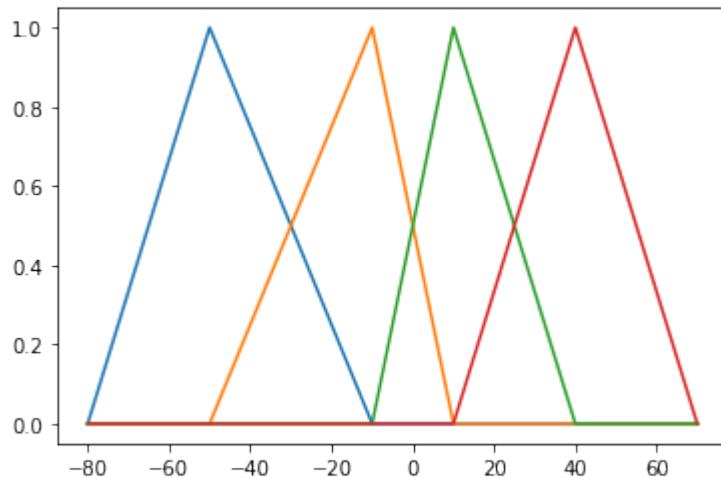


FIGURE 2.10 – Fonction d'appartenance de RC

5 Définition de la base de règle floue

Notre base de règle peut être représentée avec la matrice d'inférence suivante :

NC	DN			HN		
PI/TC	AV	AC	IN	AV	AC	IN
TG	FA	FA	FA	MO	MO	MO
GR	FA	MO	MO	MO	FO	FO
MO	MO	MO	MO	FO	FO	TF
FA	FO	FO	FO	TF	TF	TF

FIGURE 2.11 – Matrice d'inférence de notre base de règles

Afin de pouvoir programmer notre base de règle en python, nous utilisons le code suivant utilisant le module Rule de notre librairie.

```

1 # initialisation des regles de notre systeme
2 rules = Rule({
3     (TC.AV, NC.DN, PI.TG): RC.FA,# R1
4     (TC.AV, NC.DN, PI.GR): RC.MO,# R2
5     (TC.AV, NC.DN, PI.MO): RC.FO,# R3
6     (TC.AV, NC.DN, PI.FA): RC.TF,# R4
7     (TC.AV, NC.HN, PI.TG): RC.FA,# R5
8     (TC.AV, NC.HN, PI.GR): RC.MO,# R6
9     (TC.AV, NC.HN, PI.MO): RC.FO,# R7
10    (TC.AV, NC.HN, PI.FA): RC.TF,# R8
11    (TC.AC, NC.DN, PI.TG): RC.FA,# R9
12    (TC.AC, NC.DN, PI.GR): RC.MO,# R10
13    (TC.AC, NC.DN, PI.MO): RC.FO,# R11
14    (TC.AC, NC.DN, PI.FA): RC.TF,# R12
15    (TC.AC, NC.HN, PI.TG): RC.FA,# R13
16    (TC.AC, NC.HN, PI.GR): RC.MO,# R14
17    (TC.AC, NC.HN, PI.MO): RC.FO,# R15
18    (TC.AC, NC.HN, PI.FA): RC.TF,# R16
19    (TC.IN, NC.DN, PI.TG): RC.FA,# R17
20    (TC.IN, NC.DN, PI.GR): RC.MO,# R18
21    (TC.IN, NC.DN, PI.MO): RC.FO,# R19
22    (TC.IN, NC.DN, PI.FA): RC.TF,# R20
23    (TC.IN, NC.HN, PI.TG): RC.FA,# R21
24    (TC.IN, NC.HN, PI.GR): RC.MO,# R22
25    (TC.IN, NC.HN, PI.MO): RC.FO,# R23
26    (TC.IN, NC.HN, PI.FA): RC.TF # R24
27 })

```

FIGURE 2.12 – Code Python pour créer notre base de règles

Une fois compilé, nous obtenons l'intégralité de nos règles contenue dans la variable rule.

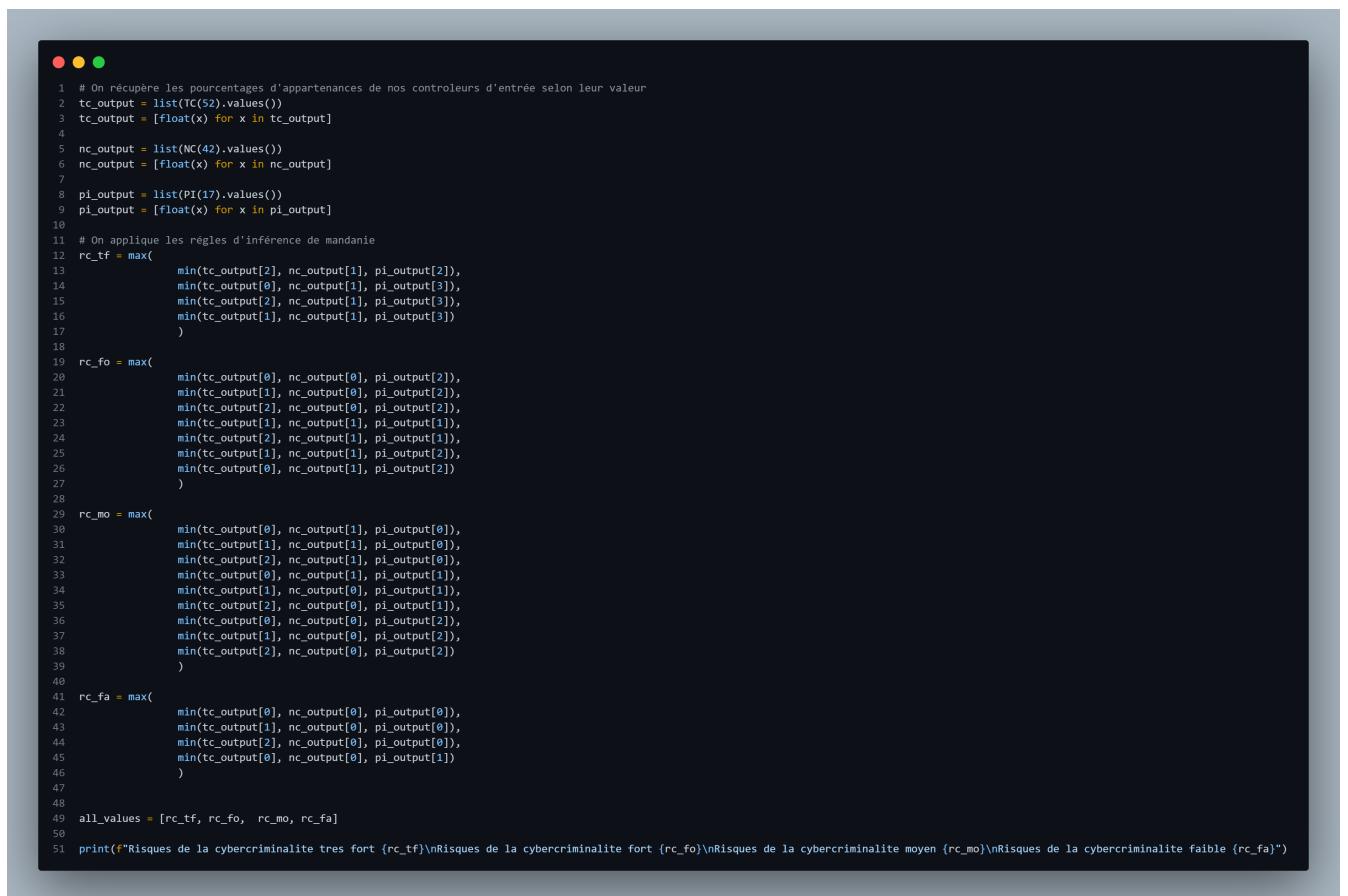
Note : On peut noter que notre matrice est totalement informé qui est une spécificité de la matrice d'inférence

6 Application de la méthode d'inférence

Dans cette étape, nous appliquons la méthode d'inférence Mandani. Pour cela, nous effectuons une simulation avec les valeurs $TC = 52$, $NC = 42$, et $PI = 17$ comme spécifié dans l'exercice. Une fois les pourcentages de nos contrôleurs récupérés, il suffit d'invoquer les règles activées et d'y appliquer la règle d'inférence de Mandani qui consiste en :

- \wedge : Min
- \vee : Max
- \implies : Min

Tout ceci est assez simple à réaliser en python une fois les pourcentages de nos contrôleurs récupérés grâce aux fonctions de notre librairie, le reste se réalise avec de la simple programmation sans aucun module extérieur comme illustrer ci-dessous :



```
1 # On récupère les pourcentages d'appartenance de nos contrôleurs d'entrée selon leur valeur
2 tc_output = list(TC(52).values())
3 tc_output = [float(x) for x in tc_output]
4
5 nc_output = list(NC(42).values())
6 nc_output = [float(x) for x in nc_output]
7
8 pi_output = list(PI(17).values())
9 pi_output = [float(x) for x in pi_output]
10
11 # On applique les règles d'inférence de mandani
12 rc_tf = max(
13     min(tc_output[2], nc_output[1], pi_output[2]),
14     min(tc_output[0], nc_output[1], pi_output[3]),
15     min(tc_output[2], nc_output[1], pi_output[3]),
16     min(tc_output[1], nc_output[1], pi_output[3])
17 )
18
19 rc_fo = max(
20     min(tc_output[0], nc_output[0], pi_output[2]),
21     min(tc_output[1], nc_output[0], pi_output[2]),
22     min(tc_output[2], nc_output[0], pi_output[2]),
23     min(tc_output[1], nc_output[1], pi_output[1]),
24     min(tc_output[2], nc_output[1], pi_output[1]),
25     min(tc_output[1], nc_output[1], pi_output[1]),
26     min(tc_output[0], nc_output[1], pi_output[2]),
27 )
28
29 rc_mo = max(
30     min(tc_output[0], nc_output[1], pi_output[0]),
31     min(tc_output[1], nc_output[1], pi_output[0]),
32     min(tc_output[2], nc_output[1], pi_output[0]),
33     min(tc_output[0], nc_output[1], pi_output[1]),
34     min(tc_output[1], nc_output[0], pi_output[1]),
35     min(tc_output[2], nc_output[0], pi_output[1]),
36     min(tc_output[0], nc_output[0], pi_output[2]),
37     min(tc_output[1], nc_output[0], pi_output[2]),
38     min(tc_output[2], nc_output[0], pi_output[2])
39 )
40
41 rc_fa = max(
42     min(tc_output[0], nc_output[0], pi_output[0]),
43     min(tc_output[1], nc_output[0], pi_output[0]),
44     min(tc_output[2], nc_output[0], pi_output[0]),
45     min(tc_output[0], nc_output[0], pi_output[1])
46 )
47
48 all_values = [rc_tf, rc_fo, rc_mo, rc_fa]
49
50 print("Risques de la cybercriminalité très fort {rc_tf}\nRisques de la cybercriminalité fort {rc_fo}\nRisques de la cybercriminalité moyen {rc_mo}\nRisques de la cybercriminalité faible {rc_fa}")
```

FIGURE 2.13 – Code Python pour appliquer la méthode d'inférence

Une fois compilé, nous obtenons les résultats suivants :

```
Risques de la cybercriminalité très fort 0.0
Risques de la cybercriminalité fort 0.13333333333333333
Risques de la cybercriminalité moyen 0.4
Risques de la cybercriminalité faible 0.5333333333333333
```

FIGURE 2.14 – Résultat obtenu suite à la méthode d'inférence

7 Application de la défuzzification

La défuzzification consiste à calculer le centre de gravité. Pour cela, nous allons d'abord afficher les valeurs que nous allons utiliser à cet effet en utilisant le code suivant :



```
● ● ●
1 fig, axis = plt.subplots(figsize=(7, 5))
2
3 x_RC = RC.range
4
5 RC_0 = np.zeros_like(x_RC)
6 parametres_RC = [RC.TF, RC.FO, RC.MO, RC.FA]
7 j=0
8 colors = ['y', 'b', 'g', 'm']
9 for each in parametres_RC:
10     axis.plot(x_RC, each.array(), colors[j], linewidth=1)
11     axis.fill_between(x_RC, RC_0, [min(all_values[j]), x] for x in each.array()), facecolor=colors[j], alpha=0.7)
12     j+=1
```

FIGURE 2.15 – Code Python pour afficher le graphe représentant les valeurs utilisés pour calculer le centre de gravité

Une fois exécuté, nous obtenons le résultat suivant :

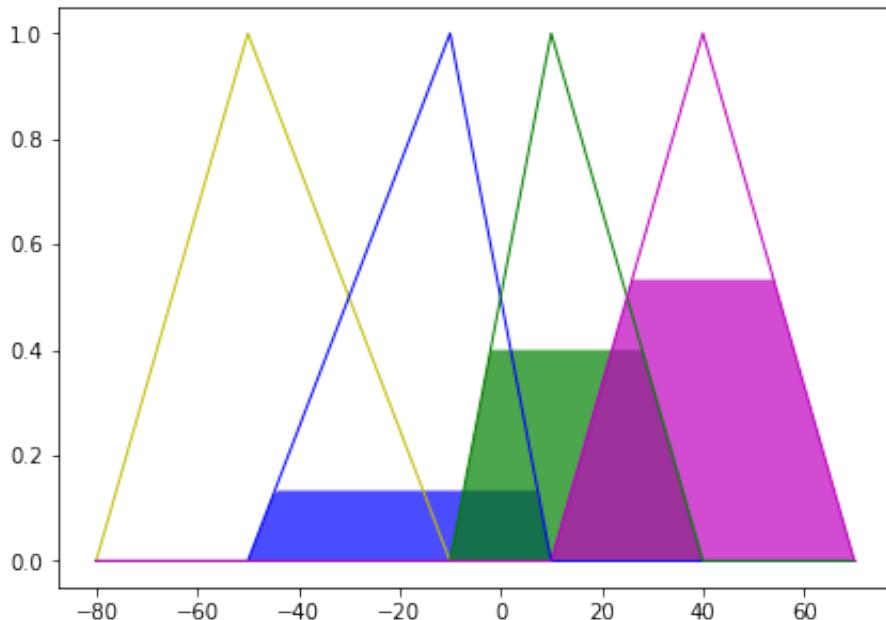


FIGURE 2.16 – Graphe visualisant les valeurs de sortie

Maintenant la dernière étape consiste à calculer le centre de gravité dont la formule est la suivante :

$$CG = \frac{\sum RC_i(x) * x}{\sum RC_i(x)}$$

En programmation, bas besoin de coder la formule, il suffit de passer les valeurs de TC, NC et PI à notre ensemble de règles pour obtenir le résultat comme montré ci-dessous :

```
● ● ●  
1 values = {TC: 52, NC: 42, PI: 17}  
2 centre_de_gravite = rules(values)  
3 print(f"Centre de gravité obtenu avec les valeurs TC = 52, NC = 42 et PI = 17 est égale à : {centre_de_gravite}")
```

FIGURE 2.17 – Code Python pour calculer le centre de gravité

Une fois compilé, nous obtenons le résultat suivant :

**Centre de gravité obtenu avec les valeurs TC = 52, NC = 42 et PI = 17 est égale à :
22.295553453169373**

Ce qui nous permet de conclure que pour notre système, si on possède :

- Technologie de la cybersécurité (TC) de 52
- Normes de la cybersécurité (NC) de 42
- Portée de l'information (PI) de 17

Nous obtiendrons un Risque de la cybercriminalité d'environ 22.3 après arrondissement.

8 Conclusion

Dans ce chapitre, nous nous sommes familiarisés avec les contrôleurs flous. En s'y exerçant dans un cas d'application nous permettant de mieux comprendre leur fonctionnement, surtout comment les concevoir et les implémenter en utilisant un langage d'actualité très utilisé en industrie et dans la recherche.

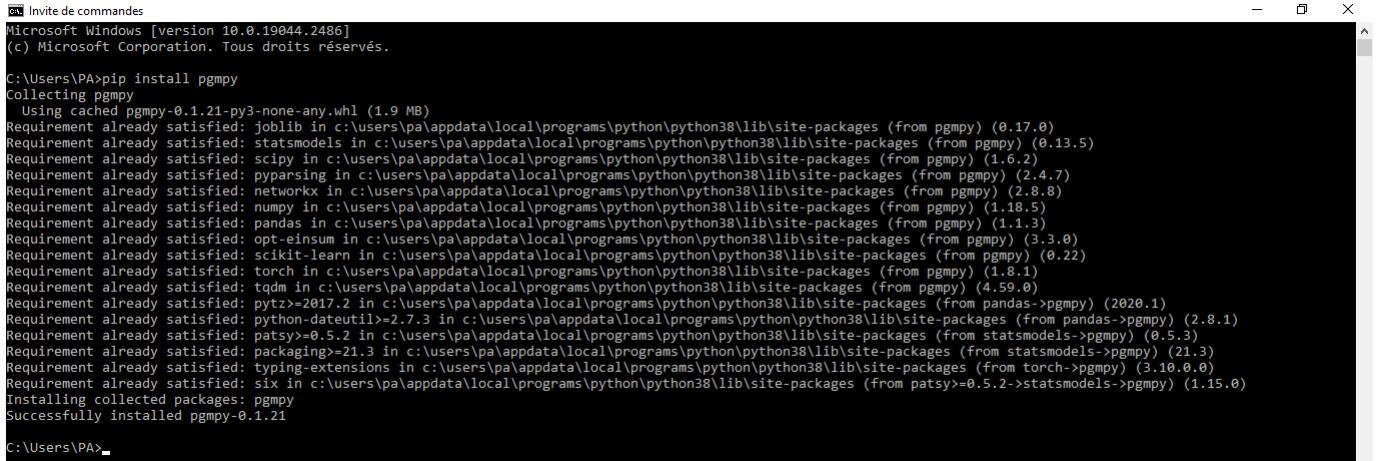
TP3 : Réseaux Bayésiens

1 Introduction

Le but de ce TP est d'implémenter des réseaux bayésiens en utilisant **PGMPY** : Une bibliothèque Python pour les réseaux Bayésiens. Cette librairie est disponible sur le repository GitHub suivant : PGMPY. Dans ce qui suit, nous allons implémenter un exemple de réseau bayésien avec une structure graphique de poly-arbre et aussi de multi-connected en suivant les étapes données dans l'énoncé du TP.

2 ÉTAPE 1 : Installation de PGMPY

Pour cela, il suffit de suivre les instructions indiquées dans la repository que nous avons partagé plutôt. Dans notre cas, nous avons juste exécuté la commande pip suivante au niveau de notre terminal.



```
Invité de commandes
Microsoft Windows [version 10.0.19044.2486]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\PA>pip install pgmpy
Collecting pgmpy
  Using cached pgmpy-0.1.21-py3-none-any.whl (1.9 MB)
Requirement already satisfied: joblib in c:\users\pa\appdata\local\programs\python\python38\lib\site-packages (from pgmpy) (0.17.0)
Requirement already satisfied: statsmodels in c:\users\pa\appdata\local\programs\python\python38\lib\site-packages (from pgmpy) (0.13.5)
Requirement already satisfied: scipy in c:\users\pa\appdata\local\programs\python\python38\lib\site-packages (from pgmpy) (1.6.2)
Requirement already satisfied: pyparsing in c:\users\pa\appdata\local\programs\python\python38\lib\site-packages (from pgmpy) (2.4.7)
Requirement already satisfied: networkx in c:\users\pa\appdata\local\programs\python\python38\lib\site-packages (from pgmpy) (2.8.8)
Requirement already satisfied: numpy in c:\users\pa\appdata\local\programs\python\python38\lib\site-packages (from pgmpy) (1.18.5)
Requirement already satisfied: pandas in c:\users\pa\appdata\local\programs\python\python38\lib\site-packages (from pgmpy) (1.1.3)
Requirement already satisfied: opt-einsum in c:\users\pa\appdata\local\programs\python\python38\lib\site-packages (from pgmpy) (3.3.0)
Requirement already satisfied: scikit-learn in c:\users\pa\appdata\local\programs\python\python38\lib\site-packages (from pgmpy) (0.22)
Requirement already satisfied: torch in c:\users\pa\appdata\local\programs\python\python38\lib\site-packages (from pgmpy) (1.8.1)
Requirement already satisfied: tqdm in c:\users\pa\appdata\local\programs\python\python38\lib\site-packages (from pgmpy) (4.59.0)
Requirement already satisfied: pytz>=2017.2 in c:\users\pa\appdata\local\programs\python\python38\lib\site-packages (from pandas->pgmpy) (2020.1)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\pa\appdata\local\programs\python\python38\lib\site-packages (from pandas->pgmpy) (2.8.1)
Requirement already satisfied: patsy>=0.5.2 in c:\users\pa\appdata\local\programs\python\python38\lib\site-packages (from statsmodels->pgmpy) (0.5.3)
Requirement already satisfied: packaging>=21.3 in c:\users\pa\appdata\local\programs\python\python38\lib\site-packages (from statsmodels->pgmpy) (21.3)
Requirement already satisfied: typing-extensions in c:\users\pa\appdata\local\programs\python\python38\lib\site-packages (from torch->pgmpy) (3.10.0.0)
Requirement already satisfied: six in c:\users\pa\appdata\local\programs\python\python38\lib\site-packages (from patsy>=0.5.2->statsmodels->pgmpy) (1.15.0)
Installing collected packages: pgmpy
Successfully installed pgmpy-0.1.21

C:\Users\PA>
```

FIGURE 3.1 – Installation de PGMPY

À noter qu'on aurait pu utiliser d'autre méthode d'installation, mais nous avions jugé qu'il s'agissait de la plus simple.

3 ÉTAPE 2 : Poly Arbre

■ ETAPÉ2:

- Générez :
 - o un polyarbre,
 - o une (ou plusieurs) variable(s) d'évidence (avec son (leurs instance(s)),
 - o une variable d'intérêt (avec son instance),
 - o des distributions a priori pour les nœuds racine
 - o des distributions conditionnelles pour les autres nœuds
- Calculez $P(\text{variable d'intérêt} | \text{évidence(s)})$

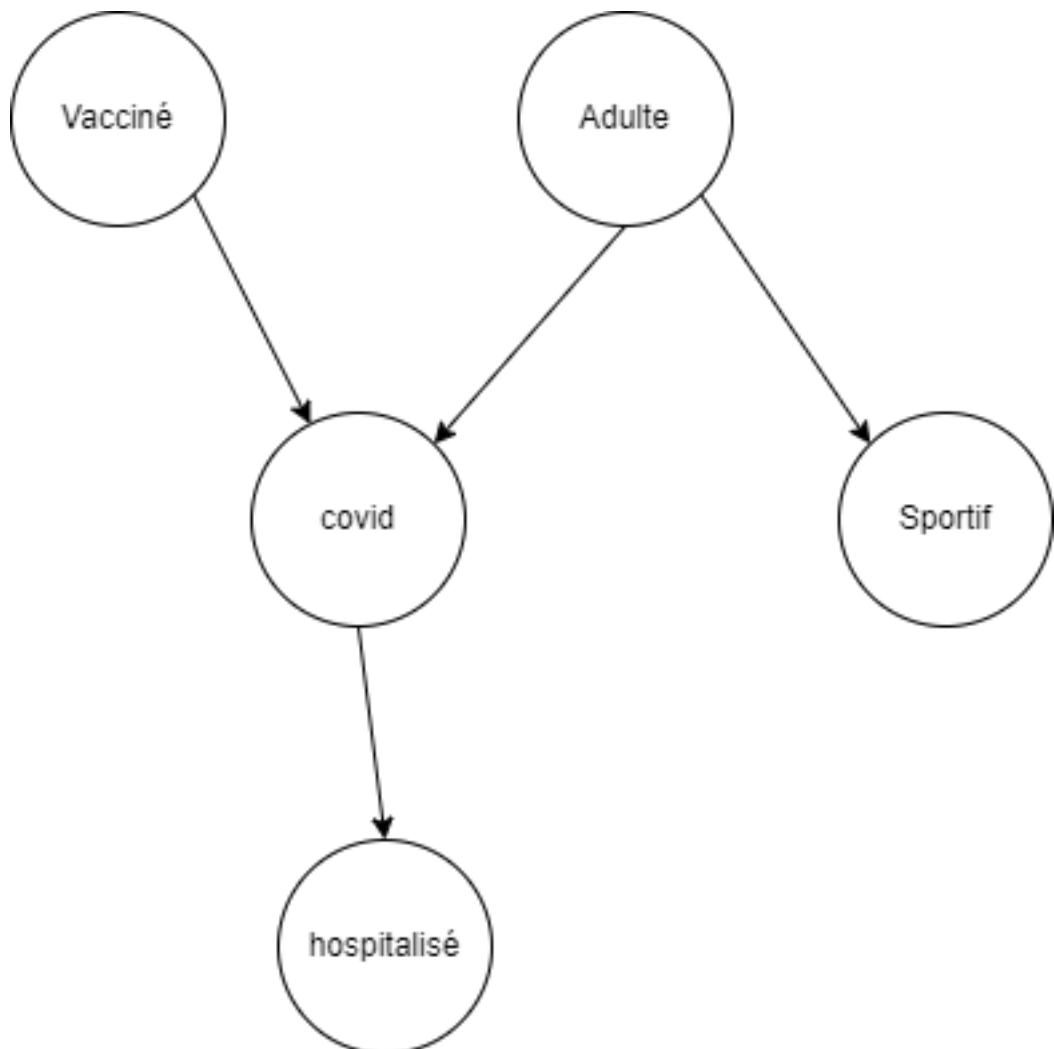
FIGURE 3.2 – Enoncé de l'etape 2

Nous avons implémenter un réseau bayésien pour modéliser des notions du domaine médical.

3.1 Nos composantes

a - Graphique

Ce réseau graphique montre la causalité entre les concepts médicaux.



b - Calcul

Dans ce qui suit, nous allons présenter la distribution des probabilités conditionnelles de chaque concept médical.

Modélisation

- Vaccin : V
- Adulte : A
- Covid : C
- Sportif : S
- Hospitalisation : H

V	P(V)
Faux	0.6
Vrai	0.4

A	P(A)
Faux	0.7
Vrai	0.3

V	A	C	P(C V, A)
Faux	Faux	Faux	0.7
Faux	Faux	Vrai	0.3
Faux	Vrai	Faux	0.8
Faux	Vrai	Vrai	0.2
Vrai	Faux	Faux	0.95
Vrai	Faux	Vrai	0.05
Vrai	Vrai	Faux	0.8
Vrai	Vrai	Vrai	0.2

C	H	P(H C)
Faux	Faux	0.9
Faux	Vrai	0.1
Vrai	Faux	0.95
Vrai	Vrai	0.05

A	S	P(S A)
Faux	Faux	0.8
Faux	Vrai	0.2
Vrai	Faux	0.9
Vrai	Vrai	0.1

3.2 Solution

```

from pgmpy.factors.discrete import TabularCPD
from pgmpy.models import BayesianNetwork

#Etablissement de la structure
medical_model = BayesianNetwork([
    ('V', 'C'),
    ('A', 'C'),
    ('C', 'H'),
    ('A', 'S'),
])
#Définition des relations
vaccin_cpd= TabularCPD(
    variable='V',
    variable_card= 2,
    values=[[.6],[.4]]
)
adulte_cpd= TabularCPD(
    variable='A',
    variable_card= 2,
    values=[[.7],[.3]]
)

sportif_cpd= TabularCPD(
    variable='S',
    variable_card=2,
    values=[[.8,.9],
            [.2,.1]],
    evidence=[ 'A'],
    evidence_card=[2]
)
covid_cpd= TabularCPD(
    variable='C',
    variable_card=2,
    values=[
        [.7,.8,.95,.8],
        [.3,.2,.05,.2]],
    evidence=[ 'V', 'A'],
    evidence_card=[2,2]
)
hospital_cpd= TabularCPD(
    variable='H',
    variable_card=2,
    values=[[.9,.95],
            [.1,.05]],
    evidence=[ 'C'],

```

```

        evidence_card=[2]
    )

#on ajoute les relation à notre model
medical_model.add_cpds(vaccin_cpd,
sportif_cpd,covid_cpd,adulte_cpd,hospital_cpd)

#reverifier la structure de notre model
medical_model.get_cpds()

[<TabularCPD representing P(V:2) at 0x18407b40748>,
 <TabularCPD representing P(S:2 | A:2) at 0x18407b40808>,
 <TabularCPD representing P(C:2 | V:2, A:2) at 0x18407b40888>,
 <TabularCPD representing P(A:2) at 0x18407b40708>,
 <TabularCPD representing P(H:2 | C:2) at 0x18407b40948>]

#trouver les chemins des noeuds pour determiner qu'est ce qui donne des infos sur un noeud
medical_model.active_trail_nodes('A')

{'A': {'A', 'C', 'H', 'S'}}

medical_model.active_trail_nodes('H')

{'H': {'A', 'C', 'H', 'S', 'V'}}

medical_model.active_trail_nodes('C')

{'C': {'A', 'C', 'H', 'S', 'V'}}

#determinaison des independances locales
medical_model.local_independencies('S')

(S ⊥ H, V, C | A)

medical_model.local_independencies('H')

(H ⊥ V, S, A | C)

medical_model.local_independencies('A')

(A ⊥ V)

medical_model.get_independencies()

(H ⊥ V, S, A | C)
(H ⊥ S | A)
(H ⊥ S, A | V, C)
(H ⊥ S | V, A)
(H ⊥ V, A | S, C)
(H ⊥ V, S | C, A)
(H ⊥ A | V, S, C)
(H ⊥ S | V, C, A)
(H ⊥ V | S, C, A)

```

```

(V ⊥ S, A)
(V ⊥ A | S)
(V ⊥ H | C)
(V ⊥ S | A)
(V ⊥ S | H, A)
(V ⊥ H | S, C)
(V ⊥ H, S | C, A)
(V ⊥ S | H, C, A)
(V ⊥ H | S, C, A)
(S ⊥ V)
(S ⊥ H | C)
(S ⊥ H, V, C | A)
(S ⊥ V, C | H, A)
(S ⊥ H | V, C)
(S ⊥ H, C | V, A)
(S ⊥ H, V | C, A)
(S ⊥ C | H, V, A)
(S ⊥ V | H, C, A)
(S ⊥ H | V, C, A)
(A ⊥ V)
(A ⊥ V | S)
(A ⊥ H | C)
(A ⊥ H | V, C)
(A ⊥ H | S, C)
(A ⊥ H | V, S, C)
(C ⊥ S | A)
(C ⊥ S | H, A)
(C ⊥ S | V, A)
(C ⊥ S | H, V, A)

from pgmpy.inference import VariableElimination

medical_infer = VariableElimination(medical_model)

#calcul des proba existante pour vérifier que tout est correct
prob_vaccin = medical_infer.query(variables=['V'])
print(prob_vaccin)

+-----+
| V    |   phi(V) |
+=====+=====
| V(0) |   0.6000 |
+-----+
| V(1) |   0.4000 |
+-----+

#faire un autre calcul de proba
prob_hospital= medical_infer.query(variables=['H'])
print(prob_hospital)

```

H	phi(H)
H(0)	0.9100
H(1)	0.0900

#probabilité d'avoir le covid si on est sportif

```
proba_covid_sportif = medical_infer.query(
    variables=['C'],
    evidence= {'S':1}
)
print(proba_covid_sportif)
```

C	phi(C)
C(0)	0.8000
C(1)	0.2000

#probabilité d'être hospitalisé si on est adulte non vacciné

```
proba_hospital_adulteNonvaccin = medical_infer.query(
    variables=['H'],
    evidence= {'V':0, 'A':1}
)
print(proba_hospital_adulteNonvaccin)
```

H	phi(H)
H(0)	0.9100
H(1)	0.0900

4 ETAPE 3 : Conception d'un arbre multi-connected

Nous avons conçu un réseau bayésien pour modéliser les conditions météorologiques.

4.1 Nos composantes

a - Graphique

Ce réseau graphique montre la causalité entre les concepts médicaux.

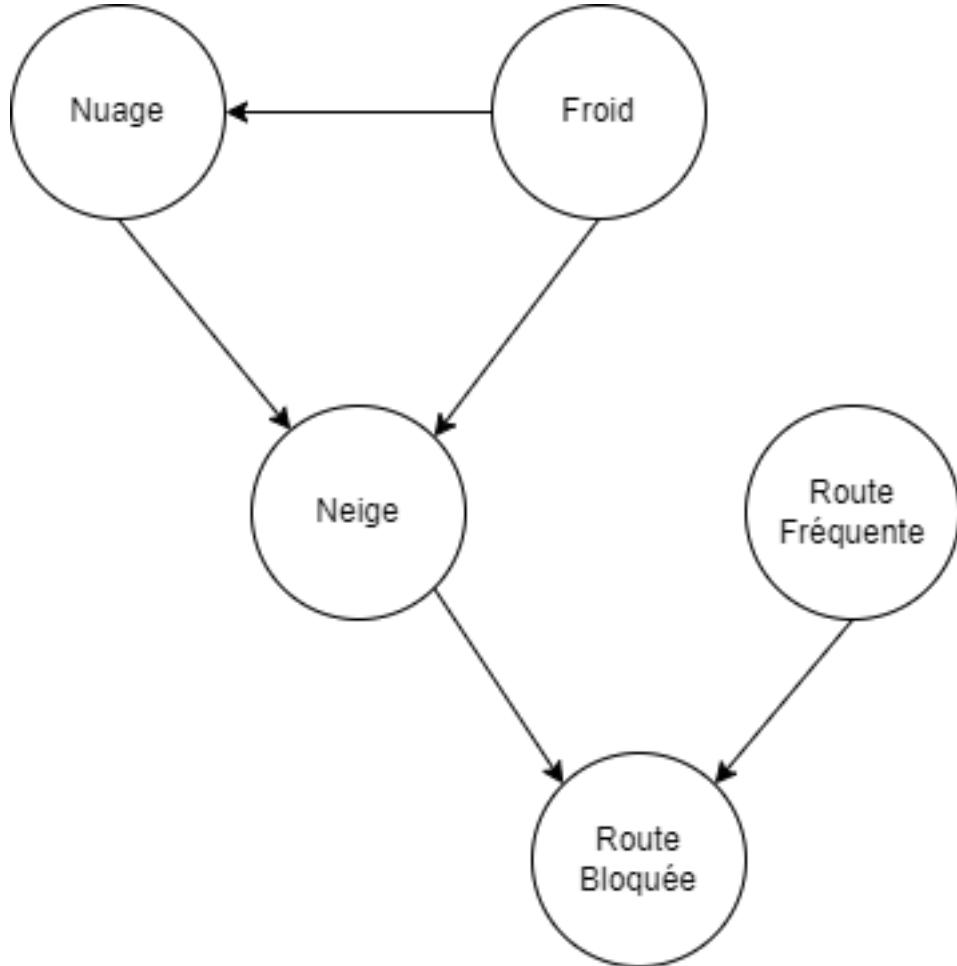


FIGURE 3.3 – Résultats : Calcul de croyance et de plausibilité de l'expert 2

b - Calcul

Dans ce qui suit, nous allons présenter la distribution des probabilités conditionnelles.

Notations

- Nuage : \mathbf{N}
- Froid : \mathbf{F}
- Neige : \mathbf{Ne}
- Route fréquente : \mathbf{R}
- Route Bloquée : \mathbf{B}

N	F	P(N)
Faux	Faux	0.6
Faux	Vrai	0.4
Vrai	Faux	0.3
Vrai	Vrai	0.7

F	P(F)
Faux	0.75
Vrai	0.25

F	N	Ne	P(Ne F, N)
Faux	Faux	Faux	0.99
Faux	Faux	Vrai	0.01
Faux	Vrai	Faux	0.8
Faux	Vrai	Vrai	0.2
Vrai	Faux	Faux	0.7
Vrai	Faux	Vrai	0.3
Vrai	Vrai	Faux	0.4
Vrai	Vrai	Vrai	0.6

R	P(R)
Faux	0.6
Vrai	0.4

N	R	B	P(B N, R)
Faux	Faux	Faux	0.9
Faux	Faux	Vrai	0.1
Faux	Vrai	Faux	0.4
Faux	Vrai	Vrai	0.6
Vrai	Faux	Faux	0.2
Vrai	Faux	Vrai	0.8
Vrai	Vrai	Faux	0.3
Vrai	Vrai	Vrai	0.7

4.2 Solution

```

from pgmpy.factors.discrete import TabularCPD
from pgmpy.models import BayesianNetwork

#Etablissement de la structure
weather_model = BayesianNetwork([
    ('F', 'N'),
    ('F', 'Ne'),
    ('N', 'Ne'),
    ('Ne', 'B'),
    ('R', 'B'),
])
#Définition des relations
froid_cpd= TabularCPD(
    variable='F',
    variable_card= 2,
    values=[[.75],[.25]]
)

nuage_cpd= TabularCPD(
    variable='N',
    variable_card=2,
    values=[[.6,.3],
            [.4,.7]],
    evidence=['F'],
    evidence_card=[2]
)

neige_cpd= TabularCPD(
    variable='Ne',
    variable_card=2,
    values=[
        [.99,.8,.7,.4],
        [.01,.2,.3,.6]],
    evidence=['F','N'],
    evidence_card=[2,2]
)

route_frequente_cpd= TabularCPD(
    variable='R',
    variable_card=2,
    values=[[.6],[.4]]
)

route_bloque_cpd= TabularCPD(
    variable='B',
    variable_card=2,
    values=[[.9,.4,.2,.3],
            [.1,.6,.8,.7]],
)

```

```

evidence=['R','Ne'],
evidence_card=[2,2]
)

#on ajoute les relation à notre model
weather_model.add_cpds(froid_cpd,
nuage_cpd,neige_cpd,route_bloque_cpd,route_frequente_cpd)

#reverifier la structure de notre model
weather_model.get_cpds()

[<TabularCPD representing P(F:2) at 0x1e947e544c8>,
 <TabularCPD representing P(N:2 | F:2) at 0x1e947e8cf8c8>,
 <TabularCPD representing P(Ne:2 | F:2, N:2) at 0x1e947e8cf88>,
 <TabularCPD representing P(B:2 | R:2, Ne:2) at 0x1e947e641c8>,
 <TabularCPD representing P(R:2) at 0x1e947e640c8>]

#trouver les chemins des noeuds pour determiner qu'est ce qui donne des infos sur un noeud
weather_model.active_trail_nodes('F')

{'F': {'B', 'F', 'N', 'Ne'}}

weather_model.active_trail_nodes('B')

{'B': {'B', 'F', 'N', 'Ne', 'R'}}

weather_model.active_trail_nodes('Ne')

{'Ne': {'B', 'F', 'N', 'Ne'}}

#determinaison des independances locales
weather_model.local_independencies('F')

(F ⊥ R)

weather_model.local_independencies('Ne')

(Ne ⊥ R | N, F)

weather_model.local_independencies('B')

(B ⊥ N, F | Ne, R)

weather_model.get_independencies()

(F ⊥ R)
(F ⊥ R | N)
(F ⊥ R, B | Ne)
(F ⊥ R, B | N, Ne)
(F ⊥ B | Ne, R)
(F ⊥ R | Ne, B)
(F ⊥ B | N, Ne, R)
(F ⊥ R | N, Ne, B)

```

```

(N ⊥ R)
(N ⊥ R, B | Ne)
(N ⊥ R | F)
(N ⊥ R, B | Ne, F)
(N ⊥ B | Ne, R)
(N ⊥ R | Ne, B)
(N ⊥ B | Ne, R, F)
(N ⊥ R | Ne, F, B)
(Ne ⊥ R)
(Ne ⊥ R | N)
(Ne ⊥ R | F)
(Ne ⊥ R | N, F)
(R ⊥ N, F, Ne)
(R ⊥ Ne, F | N)
(R ⊥ N, Ne | F)
(R ⊥ N, F | Ne)
(R ⊥ Ne | N, F)
(R ⊥ F | N, Ne)
(R ⊥ N | Ne, F)
(R ⊥ N, F | Ne, B)
(R ⊥ F | N, Ne, B)
(R ⊥ N | Ne, F, B)
(B ⊥ N, F | Ne)
(B ⊥ F | N, Ne)
(B ⊥ N | Ne, F)
(B ⊥ N, F | Ne, R)
(B ⊥ F | N, Ne, R)
(B ⊥ N | Ne, F, R)

from pgmpy.inference import VariableElimination

weather_infer = VariableElimination(weather_model)

#calcul des proba existante pour vérifier que tout est correct
prob_froid = weather_infer.query(variables=['F'])
print(prob_froid)

+-----+-----+
| F    |   phi(F) |
+=====+=====+
| F(0) |   0.7500 |
+-----+-----+
| F(1) |   0.2500 |
+-----+-----+

#faire un autre calcul de proba
prob_bloque= weather_infer.query(variables=['B'])
print(prob_bloque)

+-----+-----+
| B    |   phi(B) |
+-----+-----+

```

```

+=====+=====
| B(0) | 0.5701 |
+-----+
| B(1) | 0.4299 |
+-----+

#probabilité d'avoir une route bloqué quand il fait froid (proba conditionnelle)

proba_bloque_froid = weather_infer.query(
    variables=['B'],
    evidence= {'F':1}
)
print(proba_bloque_froid)

+-----+-----
| B    | phi(B) |
+=====+=====
| B(0) | 0.4874 |
+-----+
| B(1) | 0.5126 |
+-----+


#probabilité d'avoir de la neige quand il fait pas froid mais quand il y a des nuages

proba_neige_nonFroidetNuage = weather_infer.query(
    variables=['Ne'],
    evidence= {'F':0, 'N':1}
)
print(proba_neige_nonFroidetNuage)

+-----+-----
| Ne   | phi(Ne) |
+=====+=====
| Ne(0) | 0.8000 |
+-----+
| Ne(1) | 0.2000 |
+-----+

```

TP4 : Théorie des possibiliste

1 introduction

La logique possibiliste est une logique de représentation et de traitement de l'incertitude issue de la théorie des possibilités. Dans ce TP, nous allons nous attarder sur le processus d'inférence à calculer la plus grande valeur de α telle que (φ, α) (Avec φ une formule, d'un ensemble de formules, α un poids, d'un ensemble de poids). Ceci revient à prouver que $(\Sigma \cup (\neg\varphi, 1))$ infère \perp en utilisant le principe de la réfutation qui est équivalent à : $\text{Val}(\varphi, \Sigma) = \text{Incons}(\Sigma \cup (\neg\varphi, 1))$. Pour cela, on utilisera l'algorithme d'inférence qui est basé sur le principe de dichotomie comme le montre l'algorithme ci-dessous.

```
begin
    l := 0 ;
    u := n ;
    while l < u do
        r := [(l + u) / 2] ;
        if  $\Sigma^* \wedge \neg\varphi$  consistent then
            u := r-1
        else
            l := r
    { $\text{Val}(\varphi, \Sigma) = \alpha_r$ }
end
```

FIGURE 4.1 – Algorithme d'inférence

Pour la réalisation de ce TP, nous utiliserons le langage Python ainsi que la librairie pycosat qui fera office de prouveurs SAT.

2 Création de notre base de connaissance

La première étape avant de réaliser notre algorithme est de créer notre base de connaissance que nous créons en utilisant le code suivant :



```
1  class BaseConnaissance:
2      number_of_clause    = 0
3      number_of_variable = 0
4      evidence_length   = 0
5      var_list = []
6      max_clause_length = 0
7
8      def __init__(self,nb_clause,nb_variable,length,max_number):
9          self.number_of_clause = nb_clause
10         self.number_of_variable = nb_variable
11         self.evidence_length = length
12         self.max_clause_length = max_number
13         for i in range(self.number_of_variable):
14             self.var_list.append(i+1)
15             self.var_list.append(-(i+1))
16
17     def get_evidence(self):
18         return random.sample(self.var_list, self.evidence_length)
19
20     def get_sigma(self):
21         os.system("touch temp.txt")
22         sigma_file = open("temp.txt","w")
23         for i in range(self.number_of_clause):
24             ran = random.randrange(1,self.max_clause_length + 1,1)
25             temp_list = random.sample(self.var_list, ran)
26             weight = random.random()
27             weight = float(int(weight*100))/100
28             chaine = "" + str(weight) + " "
29             for i in range(len(temp_list)):
30                 chaine = chaine + str(temp_list[i]) + " "
31             sigma_file.write(chaine + "\n")
32             temp_list=[]
33         sigma_file.close()
```

FIGURE 4.2 – Classe python représentant notre base de connaissance

Cette classe nous permet de générer de manière aléatoire toutes les informations nécessaires à notre base de connaissance ainsi qu'une évidence.

3 Traitement de notre base de connaissance

Une fois notre base de connaissance créée, de nombreux traitements sont nécessaires avant d'utiliser notre algorithme. Tel qu'extraire la liste des poids et des formules, calculer le nombre des strates, celons les valeurs telles qu'une strate regroupe les formules ayant le même poids, et calculer le nombre des strates dans notre base de connaissance.

Pour réaliser tout cela, nous utilisons le code suivant :



```
1  class Sigma:
2      lower = 0
3      upper = 0
4      weights = []
5      formulas = []
6      strates_weights = []
7      def __init__(self,path_file):
8          file = open(path_file)
9          for line in file:
10              information = line.split()
11              size = len(information)
12              self.weights.append(float(information[0]))
13              self.formulas.append([int(information[i]) for i in range(size) if i != 0])
14          self.length = len(self.weights)
15          file.close()
16
17      def get_length(self):
18          return self.length
19
20      def get_weights(self):
21          return self.weights
22
23      def get_formulas(self):
24          return self.formulas
25
26      def sort_weights(self):
27          for i in range(len(self.weights)-1, 0, -1):
28              for j in range(i):
29                  if self.weights[j] < self.weights[j+1]:
30                      self.weights[j], self.weights[j+1] = self.weights[j+1], self.weights[j]
31                      self.formulas[j], self.formulas[j+1] = self.formulas[j+1], self.formulas[j]
32
33      def compute_strates(self):
34          no_doubles = set(self.weights)
35          self.upper = len(no_doubles)
36          self.strates_weights = sorted(no_doubles)
37
38      def get_strates_number(self):
39          return self.upper
40
41      def get_strates_weights(self):
42          return self.strates_weights
43
44      def get_preprocessed_formulas(self,sub_formulas):
45          dict_cor={}
46          returned_formulas=[]
47          cpt=1
48          i=0
49          for form in sub_formulas:
50              j=0
51              returned_formulas.insert(i,[])
52              for pred in form:
53                  if(pred not in dict_cor.keys()):
54                      if(int(pred)>0):
55                          dict_cor[int(pred)]=cpt
56                          dict_cor[-int(pred)]=-cpt
57                          cpt+=1
58                      else:
59                          dict_cor[-int(pred)]=cpt
60                          dict_cor[int(pred)]=-cpt
61                          cpt+=1
62                  returned_formulas[i].insert(j,dict_cor[pred])
63                  j+=1
64              i+=1
65          return returned_formulas
66
```

FIGURE 4.3 – Classe python permettant de traiter notre base de connaissance

4 Création et exécution de notre algorithme

Maintenant que tout est prêt, nous pouvons enfin s'attaquer à notre algorithme dont le fonctionnement et la réalisation est assez simple comme montrer ci-dessous :

```

1 base_de_connaissance.sort_weights()
2 base_de_connaissance.compute_strates()
3 iteration = 1
4 while base_de_connaissance.lower < base_de_connaissance.upper:
5     r = int((base_de_connaissance.lower + base_de_connaissance.upper + 1)/2)
6
7     liste = base_de_connaissance.get_weights()
8     value_of_r = -1
9     for i in range(len(liste)):
10         if(base_de_connaissance.get_strates_weights()[r-1]>liste[i]):
11             value_of_r = i
12             break
13     if value_of_r == -1:
14         value_of_r = len(liste)-1
15
16     for j in range(len(liste)):
17         if base_de_connaissance.get_strates_weights()[base_de_connaissance.upper-1] == liste[j] :
18             valueU = j
19             break
20
21     cnf = base_de_connaissance.formulas[valueU:value_of_r]
22
23     for i in range(len(evidence)):
24         liste = []
25         liste.append(-1 * evidence[i])
26         cnf.append(liste)
27
28     cnf = base_de_connaissance.get_preprocessed_formulas(cnf)
29
30     result = pycosat.solve(cnf)
31     if type(result) == type([]):
32         base_de_connaissance.upper = r - 1
33     else:
34         base_de_connaissance.lower = r
35
36     iteration = iteration + 1
37
38 Val = base_de_connaissance.get_strates_weights()[r-1]

```

FIGURE 4.4 – Classe python représentant notre algorithme d’inférence

Notre programme ci-dessus peut être facilement expliqué, car il se contente de suivre les étapes suivantes :

- Créeation d'une base de connaissance aléatoire.
 - Récupérer la liste des poids et des formules.
 - Trier nos formules dans un ordre décroissant celons les valeurs des poids.
 - Affecter le upper avec le nombre de strates en utilisant la méthode compute_strates().
 - Tant que le lower est plus petit que l'upper, on exécute en boucle les étapes suivantes :
 - On calcule la valeur de r celons la formule suivante : $r = \frac{l+u+1}{2}$
 - On détecte les deux indices des formules associés aux strates r et u.
 - Notre solveur pycosat se charge par la suite de réaliser test de consistance sur les formules de strates r jusqu'à la strate u.
 - Une fois le teste effectué, nous recevons différent type de valeurs possible selon le résultat du test. Dans le cas ou le résultat du test affirme que $\Sigma * r \wedge \neg\varphi$ est consistant, nous recevons une liste des variables dans le cas de consistance. Au contraire, dans le cas d'une inconsistance, nous recevons une chaine « UNSAT » et dans le cas d'une insuffisance d'itération pour la résolution une chaine « UNKNOW ».
 - Si le résultat est consistant, alors, on met à jour upper à r-1, sinon on met lower à r.
- Lorsqu'on sort de la boucle, la valeur de Val (φ, Σ) s'égale à αr .

Une fois notre programme exécuté, nous avons obtenu les résultats suivants :

```
L'exécution d'une base de connaissance qui contient :  
0.6 1 2  
0.55 2 3  
0.5 -2  
0.4 4  
0.3 5  
0.2 6  
Et avec une évidence de : [1, -2, -3]  
Donne le résultat suivant : Val([1, -2, -3], Sigma) = 0.5
```

FIGURE 4.5 – Résultat de notre algorithme

5 Conclusion

La réalisation de ce TP nous aura permis de nous intéresser au problème de la déduction en logique possibiliste qualitative en calculer un degré d'incohérence. L'implémentation de cet algorithme nous aura donné un meilleur aspect des différentes étapes de ce processus.

TP5 : Inférence logique et propagation graphique en théorie des possibilités

Dans ce dernier TP, nous allons exécuter l'inférence et la propagation graphique en théorie des possibilités en utilisant l'outil PNT. Pour exécuter ce dernier, nous allons utiliser Cygwin un émulateur Unix pour Windows pour exécuter les deux programmes `passage.exe` et `inference.exe`. Après avoir fait fonctionner l'outil, nous allons tester les différents temps d'exécution de la propagation et l'inférence sur les réseaux possibilités qui correspondent à polyarbre, Graphes faiblement connectés, Graphes moyennement connectés et Graphes fortement connectés.

1 Utilisation de la Toolbox

Ce TP vise à accomplir quelques objectifs dont une comparaison qui mettra en avant les temps de propagation et de l'inférence pour différents scénarios, et une Conclusion sur les différents résultats obtenus.

1.1 L'utilisation de Cygwin

Le premier programme associe au graphe possibiliste basé sur le produit la base de connaissance possibiliste quantitative correspondante. Lorsque nous exécutons le programme `inference.exe` afin de lancer processus d'inférence qui consiste à calculer le degré de possibilité de l'instance de la variable d'intérêt ainsi que le temps d'inférence.

Nous utiliserons Cygwin pour exécuter les deux programmes `passage.exe` et `inference.exe`,

* le premier programme associe au graphe de possibilité basé sur le produit, la base de connaissance possibilité quantitative correspondante.

* Lorsque nous exécutons le programme `inference.exe` afin de lancer processus d'inférence qui consiste à calculer le degré de possibilité de l'instance de la variable d'intérêt ainsi que le temps d'inférence.

a - ÉTAPE 1 : Utilisation d'un algorithme de propagation pour les réseaux possibilistes basés sur le produit

Avant d'exécuter les deux programmes, nous devons d'abord exécuter les scripts MATLAB 'prop1evid' ou 'prop2evid' pour générer le graphe de possibilité.

b - ÉTAPE 2 : Exécution des fichiers

```
PA@DESKTOP-HP9BBE2 /home/licence
$ ./passage.exe

PA@DESKTOP-HP9BBE2 /home/licence
$ ./inference.exe
1156560
PA@DESKTOP-HP9BBE2 /home/licence
$ cat resultats
*****R  sultats de la propagation graphique*****
nombre de variables: 20
nombre de parentsmax: 20
l'evidence: 1
variable d'interet: -12
possibilit   conditionnelle de l'interet | evidences 18.000000
temps de propagation 1.000000 secondes

*****R  sultats de l'in  ference logique*****
le nombre de clauses dans les bases est de: 286245
la variable d'int  ret est inf  r  e | partir de la base de p  nalit  s
le co  t de p  nalit   est de 10
avec un degr   de possibilit   correspondant est   gale | :0.000045
la dur  e de l'in  ference est: 1156560 millisecondes
PA@DESKTOP-HP9BBE2 /home/licence
$
```

FIGURE 5.1 – Résultats de l'exécution du fichier 1

c - ÉTAPE 3 :Évaluations

Nous avons effectué des évaluations sur les nombres de noeuds pour chaque scénario (polyarbre, Graphes faiblement connectés, Graphes moyennement connectés et Graphes fortement connectés) pour une évidence puis pour deux évidences.

Nous avons pris 1 parent pour avoir un polyarbre, 3 parents pour avoir un Graphe faiblement connectés, 10 parents pour un Graphe moyennement connectés et enfin 20 parents pour avoir un Graphe fortement connectés.

Pour le nombre de noeuds, nous avons testé les valeurs suivantes : [5, 10, 15, 20]. les résultats obtenus pour l'inférence et la propagation pour une évidence puis 2 évidences sont présentées ci-dessous.

1.2 Une évidence

nb nodes / nb de parents	1	3	10	20
5	76800	79556	86009	70713
10	80881	70009	86301	71781
15	69980	71530	78269	95093
20	68135	91594	85614	83655

TABLE 5.1 – Temps d'exécution de l'inférence en ms pour une évidence

nb nodes / nb de parents	1	3	10	20
5	0	0.032381	0.010385	0.012787
10	0	0.033178	0.018347	0.018947
15	0	0.049536	0.036939	0.036784
20	0	0.067523	0.054596	0.105959

TABLE 5.2 – Temps d'exécution de la propagation en secondes pour une évidence

1.3 Deux évidences

nb nodes / nb de parents	1	3	10	20
5	69612	68500	67124	89463
10	70760	70313	83714	77229
15	68381	69963	73974	134646
20	83144	87290	90105	1156560

TABLE 5.3 – Temps d'exécution de la propagation en secondes pour deux évidences

nb nodes / nb de parents	1	3	10	20
5	2	2	1	2
10	0	1	1	2
15	0	1	1	2
20	0	1	2	1

TABLE 5.4 – Temps d'exécution de la propagation en secondes pour deux évidences

Conclusion Générale

Ces TPs nous ont fait découvrir les différentes logiques sur lesquelles un système informatique peut opérer. Nous avons appris à modéliser des problèmes de la vie réelle, générer des inférences, utiliser des raisonneurs pour étudier la véracité de formule, représenter graphiquement des informations et finalement déduire des faits.