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 Informatique

Master Systèmes Informatiques intelligents

Module : Représentation de connaissances et raisonnement 2

Rapport des TPs RCR

Réalisé par :
FERKOUS Sarah, 191931043867
KHEMISSI Maroua, 191935007943

Année universitaire : 2023 / 2024

Table des matières

Introduction Generale

La représentation des connaissances consiste à organiser et structurer l'information de manière à faciliter sa compréhension et son utilisation. Elle peut prendre la forme de modèles, schémas ou graphes, facilitant la manipulation et l'analyse des données. Au cours de cette série de travaux pratiques, nous explorerons les méthodes permettant d'accomplir cela, tout en examinant les diverses logiques et les types de raisonneurs qui effectuent des déductions dans ce contexte.

TP1 : Logique des fonctions de croyances

1.1 Introduction :

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.

1.2 L'outil choisie

La bibliothèque Python utilisée est appelée "pyds" (pyds.py) et elle est utilisée pour effectuer des opérations sur les fonctions de masse de croyance (Belief Mass Functions - BMF) .

1.3 Example D'application

On a choisi ce exercise pour l'exploitation de cette bibliotheque : Trois experts tentent d'évaluer les causes du changement climatique dans une région donnée.

Le premier expert suggère que le changement climatique est attribuable à 30 aux activités humaines, à 40 aux variations naturelles, et à 30 à des facteurs inconnus.

Le deuxième expert affirme que le changement climatique est principalement dû à des facteurs humains à hauteur de 50, mais il pourrait également être causé par des variations naturelles à 20.

Le dernier expert déclare que le changement climatique est soit résultat d'activités humaines (comme l'industrialisation) à hauteur de 60, soit il est lié à des facteurs inconnue a 40.

1.3.1 Modélisation des connaissances

L'ensemble Ω est défini comme $\Omega = \{A, B\}$. tel que :

A : activités humaines

B : variations naturelles

Le premier expert (E1) croit en :

m(A) = 0.3 m(B) = 0.4 m(Ω) = 0.3

Le deuxième expert (E2) croit en :

m(A) = 0.5 m(B) = 0.2

Le dernier expert (E3) croit en :

$m(A) = 0.6$ $m(\Omega) = 0.4$

1.3.2 Modélisation avec pyds

```python
from pyds import MassFunction

# Expert 1
print("Expert 1")
m1 = MassFunction({'A': 0.3, 'B': 0.4, 'C': 0.3})
print("m_1 =", m1)

# Expert 2
print("\nExpert 2")
m2 = MassFunction({'A': 0.5, 'B': 0.2})
print("m_2 =", m2)

# Expert 3
print("\nExpert 3")
m3 = MassFunction({'A': 0.6, 'C': 0.4})
print("m_3 =", m3)
```

```
Expert 1
m_1 = {{'B'}:0.4; {'A'}:0.3; {'C'}:0.3}

Expert 2
m_2 = {{'A'}:0.5; {'B'}:0.2}

Expert 3
m_3 = {{'A'}:0.6; {'C'}:0.4}
```

FIGURE 1.1 – Modelisation avec pyds

1.4 Conclusion :

Dans nos résultats, on peut voir la croyance maximale associée à chaque combinaison. Regardons chaque cas :

Combinaison de m1 et m2 :

'A' : 0.6521739130434782 'B' : 0.3478260869565218

La proposition avec la plus grande croyance est 'A' (0.6521739130434782).

Combinaison de m2 et m3 :

'A' : 1.0 La proposition avec la plus grande croyance est 'A' (1.0).

Combinaison de m1 et m3 :

'A' : 0.6 'C' : 0.4 La proposition avec la plus grande croyance est 'A' (0.6).

Combinaison de m1, m2 et m3 :

```python
#affichage de croyance et plausibilité pour chaque expert
print("Croyance et possibilité pour l'expert 1")
print("bel_1 = ", m1.bel())
print("pl_1 = ", m1.pl())

print("Croyance et possibilité pour l'expert 2")
print("bel_2 = ", m2.bel())
print("pl_2 = ", m2.pl())

print("Croyance et possibilité pour l'expert 3")
print("bel_3 = ", m3.bel())
print("pl_3 = ", m3.pl())
```

```
Croyance et possibilité pour l'expert 1
bel_1 =  {frozenset(): 0.0, frozenset({'B'}): 0.4, frozenset({'A'}): 0.3, frozenset({'C'}): 0.3, frozenset({'B', 'A'}): 0.7,
pl_1 =  {frozenset(): 0.0, frozenset({'B'}): 0.4, frozenset({'A'}): 0.3, frozenset({'C'}): 0.3, frozenset({'B', 'A'}): 0.7,
Croyance et possibilité pour l'expert 2
bel_2 =  {frozenset(): 0.0, frozenset({'B'}): 0.2, frozenset({'A'}): 0.5, frozenset({'B', 'A'}): 0.7}
pl_2 =  {frozenset(): 0.0, frozenset({'B'}): 0.2, frozenset({'A'}): 0.5, frozenset({'B', 'A'}): 0.7}
Croyance et possibilité pour l'expert 3
bel_3 =  {frozenset(): 0.0, frozenset({'C'}): 0.4, frozenset({'A'}): 0.6, frozenset({'C', 'A'}): 1.0}
pl_3 =  {frozenset(): 0.0, frozenset({'C'}): 0.4, frozenset({'A'}): 0.6, frozenset({'C', 'A'}): 1.0}
```

FIGURE 1.2 – Croyance et plausibilité pour chaque expert

```python
#combinaisons des masses par fusion de Dempster-Shafer
print("Dempster-Shafer Combinaison rule")
print("Dempster-Shafer Combinaison rule for m_1 and m_2 = ", m1 & m2)
print("Dempster-Shafer Combinaison rule for m_2 and m_3 = ", m2 & m3)
print("Dempster-Shafer Combinaison rule for m_1 and m_3 = ", m1 & m3)

print("Dempster-Shafer Combinaison rule for m_1, m_2 and m_3 = ", m1.combine_conjunctive(m2, m3))
print("Dempster-Shafer Combinaison rule for m_2, m_1 and m_3 = ", m2.combine_conjunctive(m1, m3))
print("Dempster-Shafer Combinaison rule for m_3, m_1 and m_2 = ", m3.combine_conjunctive(m1, m2))
```

```
Dempster-Shafer Combinaison rule
Dempster-Shafer Combinaison rule for m_1 and m_2 =  {{'A'}:0.6521739130434782; {'B'}:0.3478260869565218}
Dempster-Shafer Combinaison rule for m_2 and m_3 =  {{'A'}:1.0}
Dempster-Shafer Combinaison rule for m_1 and m_3 =  {{'A'}:0.6; {'C'}:0.4}
Dempster-Shafer Combinaison rule for m_1, m_2 and m_3 =  {{'A'}:0.6521739130434782; {'B'}:0.3478260869565218}
Dempster-Shafer Combinaison rule for m_2, m_1 and m_3 =  {{'A'}:0.6521739130434782; {'B'}:0.3478260869565218}
Dempster-Shafer Combinaison rule for m_3, m_1 and m_2 =  {{'A'}:0.6; {'C'}:0.4}
```

FIGURE 1.3 – Combinaisons des masses par fusion de Dempster-Shafer

'A' : 0.6521739130434782 'B' : 0.3478260869565218

La proposition avec la plus grande croyance est 'A' (0.6521739130434782).

Combinaison de m2, m1 et m3 :

'A' : 0.6521739130434782 'B' : 0.3478260869565218

La proposition avec la plus grande croyance est 'A' (0.6521739130434782).

Combinaison de m3, m1 et m2 :

'A' : 0.6 'C' : 0.4

La proposition avec la plus grande croyance est 'A' (0.6).

on peut deduire que A (activités humaines) est le raison le plus forte de changement de climate.

TP2 : Contrôleurs flous

2.1 Introduction

La vie courante est parsemée de concepts qui ne peuvent pas être décrits par des notions mathématiques simples. Chacun de nous manipule ainsi des notions qui sont soit imprécises (assez tôt, important, environ, ...) soit incertaines (il est possible que ...) : "Je cherche un appartement de taille moyenne, pas trop cher et proche du centre ville" ou "Il est possible que le ciel se couvre vers midi". Pour repondre a ces besoin on utilise un contrôleur flou.
Un "contrôleur flou" fait référence à un dispositif ou à un système qui utilise la logique floue pour prendre des décisions ou effectuer des contrôles.
Dans un contrôleur flou, les règles sont exprimées sous forme de propositions linguistiques. Ces propositions sont évaluées en utilisant des ensembles flous et des opérations sur ces ensembles. Le résultat est souvent une sortie également exprimée de manière floue.
La figure suivante montre le principe d'utilisation d'un contrôleur flou.

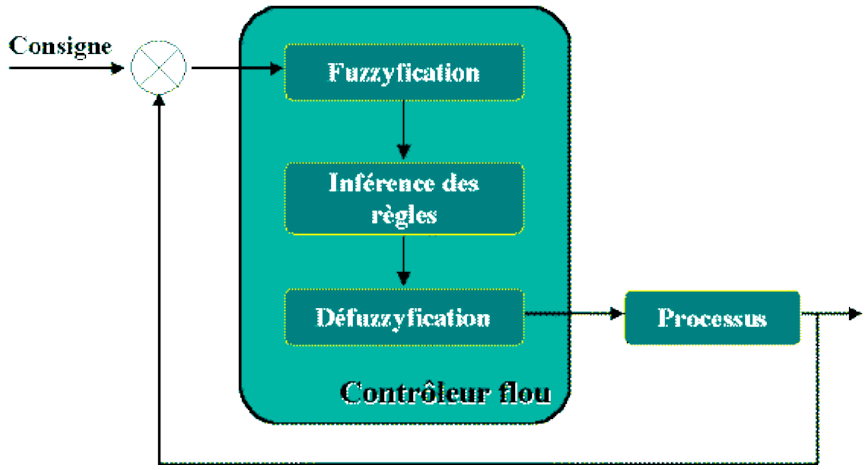

FIGURE 2.1 – Controleur Flou

1. **Fuzzyfication :** passage du monde réel (une tension, un nombre, ...) à la représentation floue.
2. **Inférence des règles :** déroulement de toutes les règles de description du processus.
3. **Défuzzyfication :** passage de la représentation floue à la représentation réelle.

2.2 Problème

Dans cette section, nous aborderons un problème spécifique et tenterons de le résoudre en utilisant un contrôleur flou afin de mieux appréhender son fonctionnement.

2.2.1 Enoncè

Les fonctions d'appartenance correspondantes aux paramètres d'entrée et de sortie sont définies par :

Negative Grande (NG)	(-100, -30, -10)
Negative Petite (PN)	(-30, -10, 0)
Envers Zero (EZ)	(-10, 0, 20)
Positive Petite (PP)	(0, 20, 50)
Positive Grande (PG)	(20, 50, 120)

TABLE 2.1 – Paramètre d'entrée puissance réactive mesurée (PM)

V min admissible (Vmiad)	(-13, -10, -8, -10)
V min desiree (Vmid)	(-8, -7 ,-3, -1)
Envers Zero (EZ)	(-3, -1, 2, 3)
V max desiree (Vmxd)	(2, 3, 5, 6)
V max admissible (Vmxad)	(5, 6, 12, 15)

TABLE 2.2 – Parametre d'entree Niveau de tension (NT)

V min admissible (Vmiad)	(-110, -50, -30)
V min desiree (Vmid)	(-50, -30 ,-10)
Envers Zero (EZ)	(-30, -10, 10)
V max desiree (Vmxd)	(-10, 10, 60)
V max admissible (Vmxad)	(10, 60, 90)

TABLE 2.3 – Parametre de sortie Puisance Reactive adaptee (PA)

PM / NT	PG	PP	EZ	NP	NG
Vmiad	PG	PG	PP	PG	PG
Vmin	PG	PP	EZ	PP	PG
EZ	PP	EZ	EZ	EZ	PP
Vmxd	NG	NP	EZ	NP	NG
Vmxad	NG	NG	NP	NG	NG

TABLE 2.4 – Les Règles d'inférence

Simuler le fonctionnement du contrôleur flou en explicitant les différentes étapes avec les paramètres suivants : PM = -25, NT= -2.5

2.3 Résolusion

Nous avons choisit l'outil fuzzy **fuzzylogic** de **Python** qui est particulièrement utile dans les situations où les règles du système ne peuvent pas être formulées de manière précise ou lorsque l'incertitude est présente. Elle offre une approche plus naturelle et humaine pour modéliser des systèmes complexes où la frontière entre les catégories n'est pas clairement définie.

2.3.1 Initialisation des E/S

Première étape, la création des ensembles flous. Dans note cas, il y a deux variables d'entree correspondant a :

1. Puissance reactive mesuree (PM))
2. Niveau de tension (NT)

Et un ensemble de sortie : Puisance Reactive adaptee (PA).

```
# Premier controleur d'entrée
PM = Domain('Puissance Reactive mesure', -100, 120)

# Deuxieme controleur d'entrée
NT = Domain('Niveau de Tension', -13, 15)

# Troisième controleur d'entrée
PA = Domain('Puissance Reactive adaptee', -110, 90)
```

FIGURE 2.2 – Entree/Sortie

2.3.2 Fuzification

```
# Paramètres d'entrée PM
PM.NG = trapezoid(-100, -30, -30, -10)
PM.NP = trapezoid(-30, -10, -10, 0)
PM.EZ = trapezoid(-10, 0, 0, 20)
PM.PP = trapezoid(0, 20, 20, 50)
PM.PG = trapezoid(20, 50, 50, 120)
PM.NG.plot()
PM.NP.plot()
PM.EZ.plot()
PM.PP.plot()
PM.PG.plot()
```

FIGURE 2.3 – initialisation de PM

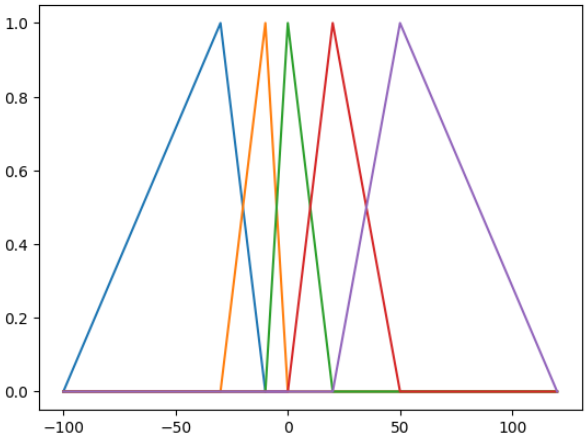

FIGURE 2.4 – initialisation de PM

```
# Paramètres d'entrée NT
NT.Vmiad = trapezoid(-13, -10, -8, -7)
NT.Vmid = trapezoid(-8, -7, -3, -1)
NT.EZ = trapezoid(-3, -1, 2, 3)
NT.Vmxd = trapezoid(2, 3, 5, 6)
NT.Vmxad = trapezoid(5, 6, 12, 15)
NT.Vmiad.plot()
NT.Vmid.plot()
NT.EZ.plot()
NT.Vmxd.plot()
NT.Vmxad.plot()
```

FIGURE 2.5 – initialisation de NT

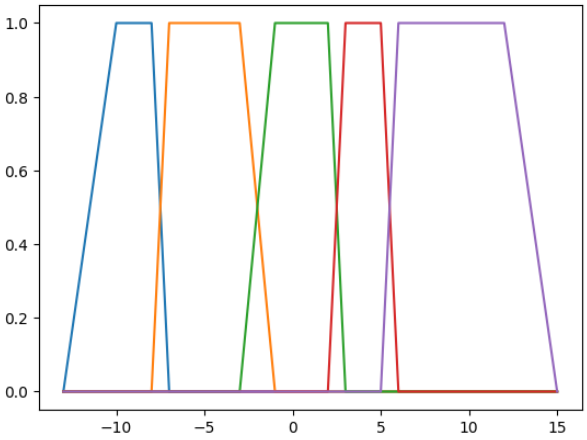

FIGURE 2.6 – initialisation de NT

```
# Paramètres de sortiee PA
PA.NG = trapezoid(-110, -50, -50, -30)
PA.NP = trapezoid(-50, -30, -30, -10)
PA.EZ = trapezoid(-30, -10, -10, 10)
PA.PP = trapezoid(-10, 10, 10, 60)
PA.PG = trapezoid(10, 60, 60, 90)
PA.NG.plot()
PA.NP.plot()
PA.EZ.plot()
PA.PP.plot()
PA.PG.plot()
```

FIGURE 2.7 – initialisation de PA

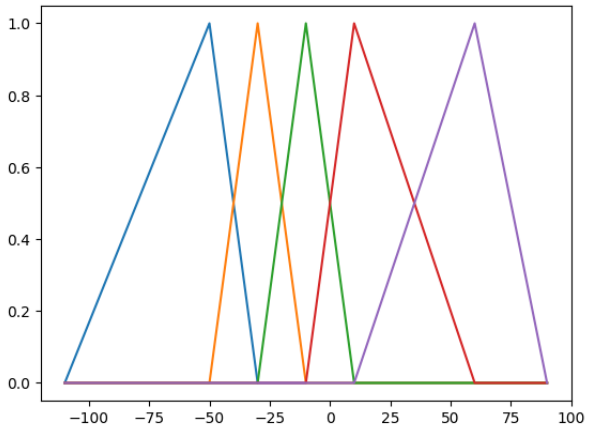

FIGURE 2.8 – initialisation de PA

2.3.3 Les Regles

```
rules = Rule({
    (NT.Vmiad, PM.PG): PA.PG,# R1
    (NT.Vmiad, PM.PP): PA.PG,# R2
    (NT.Vmiad, PM.EZ): PA.PP,# R3
    (NT.Vmiad, PM.NP): PA.PG,# R4
    (NT.Vmiad, PM.NG): PA.PG,# R5
    (NT.Vmid, PM.PG): PA.PG,# R6
    (NT.Vmid, PM.PP): PA.PP,# R7
    (NT.Vmid, PM.EZ): PA.EZ,# R8
    (NT.Vmid, PM.NP): PA.PP,# R9
    (NT.Vmid, PM.NG): PA.PG,# R10
    (NT.EZ, PM.PG): PA.PP,# R11
    (NT.EZ, PM.PP): PA.EZ,# R12
    (NT.EZ, PM.EZ): PA.EZ,# R13
    (NT.EZ, PM.NP): PA.EZ,# R14
    (NT.EZ, PM.NG): PA.PP,# R15
    (NT.Vmxd, PM.PG): PA.NG,# R16
    (NT.Vmxd, PM.PP): PA.NP,# R17
    (NT.Vmxd, PM.EZ): PA.EZ,# R18
    (NT.Vmxd, PM.NP): PA.NP,# R19
    (NT.Vmxd, PM.NG): PA.NG,# R20
    (NT.Vmxad, PM.PG): PA.NG,# R21
    (NT.Vmxad, PM.PP): PA.NG,# R22
    (NT.Vmxad, PM.EZ): PA.NP,# R23
    (NT.Vmxad, PM.NP): PA.NG, # R24
```

```
    (NT.Vmxad, PM.NG): PA.NG # R25
})
```

2.3.4 Fonctionnement

```
pm_output = list(PM(-25).values())
pm_output = [float(x) for x in pm_output]

nt_output = list(NT(-2.5).values())
nt_output = [float(x) for x in nt_output]

pa_ng = max(
                min(pm_output[3], nt_output[3]),
                min(pm_output[4], nt_output[3]),
                min(pm_output[0], nt_output[4]),
                min(pm_output[1], nt_output[4]),
                min(pm_output[3], nt_output[4]),
                min(pm_output[4], nt_output[4])
                )
pa_np = max(
                min(pm_output[1], nt_output[3]),
                min(pm_output[3], nt_output[3]),
                min(pm_output[2], nt_output[4])
                )
pa_ez = max(
                min(pm_output[2], nt_output[1]),
                min(pm_output[1], nt_output[2]),
                min(pm_output[2], nt_output[2]),
                min(pm_output[3], nt_output[2]),
                min(pm_output[2], nt_output[3])
                )
pa_pp = max(
                min(pm_output[2], nt_output[0]),
                min(pm_output[2], nt_output[1]),
                min(pm_output[2], nt_output[0]),
                min(pm_output[3], nt_output[2]),
                min(pm_output[0], nt_output[2])
                )
pa_pg = max(
                min(pm_output[0], nt_output[0]),
```

```
                min(pm_output[1], nt_output[0]),
                min(pm_output[3], nt_output[0]),
                min(pm_output[4], nt_output[0]),
                min(pm_output[0], nt_output[1]),
                min(pm_output[4], nt_output[1])
                )
all_values = [pa_ng, pa_np, pa_ez, pa_pp, pa_pg]

print(f"Puissance reactive adapte Vegative Grande {pa_ng}\nNegative Petite {pa_np}\nEnvers Zero {pa_ez}\nPositive Petite {pa_pp}\nPositive Grande {pa_pg}")
```

2.3.5 Defuzzification

```
fig, axis = plt.subplots(figsize=(7, 5))

x_PA = PA.range

PA_0 = np.zeros_like(x_PA)
parametres_PA = [PA.NG, PA.NP, PA.EZ, PA.PP, PA.PG]
j=0
colors = ['y', 'b', 'g', 'm', 'r']
for each in parametres_PA:
    axis.plot(x_PA, each.array(), colors[j], linewidth=1)
    axis.fill_between(x_PA, PA_0, [min(all_values[j], x) for x in each.array()], facecolor=colors[j], alpha=0.7)
    j+=1
```

FIGURE 2.9 – Defuzzification

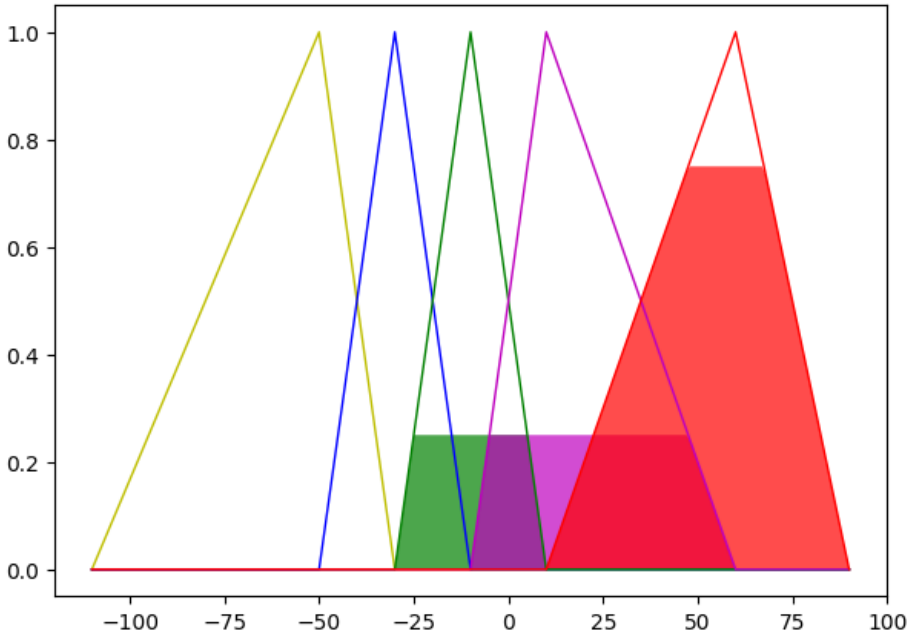

FIGURE 2.10 – Resultat finale

```
values = {PM: -25, NT: -2.5}
centre_de_gravite = rules(values)
print(f"Centre de gravite obtenu avec les valeurs PM = -25, NT = -2.5 est egale a : {centre_de_gravite}")
```

```
Centre de gravite obtenu avec les valeurs PM = -25, NT = -2.5 est egale a : 30.96185737976782
```

FIGURE 2.11 – Centre de Gravite

2.4 Conclusion

Dans cette section, nous nous sommes initiés aux contrôleurs flous. En mettant en pratique nos connaissances à travers une application concrète, nous avons amélioré notre compréhension de leur mécanisme, en particulier la manière de les concevoir et de les mettre en œuvre.

TP3 :Réseaux causaux Bayésiens

3.1 Introduction :

Les réseaux causaux bayésiens sont des outils puissants utilisés dans le domaine de la modélisation probabiliste pour représenter et analyser les relations de cause à effet entre différentes variables. Ces réseaux combinent les concepts de la théorie des probabilités bayésiennes avec la représentation graphique des relations causales.

3.2 Outil choisie :

PGMPY : Une bibliothèque Python pour les réseaux Bayésiens,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.

3.3 ÉTAPE 1 : Installation de PGMPY

```
pip install pgmpy

Collecting pgmpy
  Downloading pgmpy-0.1.24-py3-none-any.whl (2.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 21.4 MB/s eta 0:00:00
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from pgmpy) (3.2.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.11.4)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.5.3)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.10/dist-packages (from pgmpy) (3.1.1)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from pgmpy) (2.1.0+cu118)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-packages (from pgmpy) (0.14.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from pgmpy) (4.66.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.3.2)
Requirement already satisfied: opt-einsum in /usr/local/lib/python3.10/dist-packages (from pgmpy) (3.3.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas->pgmpy) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->pgmpy) (2023.3.post1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->pgmpy) (3.2.0)
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.10/dist-packages (from statsmodels->pgmpy) (0.5.3)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels->pgmpy) (23.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (3.13.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (1.12)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (3.1.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (2.1.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.2->statsmodels->pgmpy) (1.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch->pgmpy) (2.1.3)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch->pgmpy) (1.3.0)
Installing collected packages: pgmpy
Successfully installed pgmpy-0.1.24
```

FIGURE 3.1 – Installation de PGMPY

3.4 ETAPE2 : Poly-Arbre

Nous avons implémenter un réseau bayésien pour modéliser des notions du domaine changement de climat.

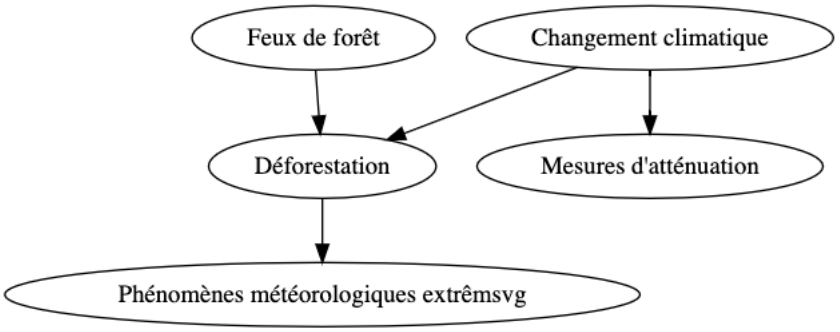

FIGURE 3.2 – Reseau bayesien des notions de changement de climat

Ce réseau graphique montre la causalité entre les concepts de changement de climat. Dans ce qui suit, nous allons présenter la distribution des probabilités conditionnelles de chaque concept :

Modélisation

— Deforestation : D

— Feux de foret : F

— Phénomènes météorologiques extrêmes : P

— Mesures d'atténuation : M

— Changement Climatique : C

La representation numerique des noeuds :

on est dans le domaine probabiliste donc on peut faire la representation numerique respectons :

- Si $(U_A) = \emptyset$ (A est un nœud racine) alors, il s'agit de spécifier les probabilités a priori relatives aux différentes instances de la variable A, tout en respectant la condition de normalisation qui stipule que :

$$\sum_a P(a) = 1$$

- Si $(U_A) \neq \emptyset$ alors il faudrait spécifier les probabilités conditionnelles des différentes instances a de A dans le contexte des différentes instances de ses parents u_A telles que :

$$\sum_a P(a \mid u_A) = 1$$

FIGURE 3.3 – Distribution conditionnels

F	P(F)
Faux	0.6
Vrai	0.4

C	P(C)
Faux	0.7
Vrai	0.3

F	C	D	P(D|F, C)
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

D	P	P(P|D)
Faux	Faux	0.9
Faux	Vrai	0.1
Vrai	Faux	0.95
Vrai	Vrai	0.05

C	M	P(M|C)
Faux	Faux	0.8
Faux	Vrai	0.2
Vrai	Faux	0.9
Vrai	Vrai	0.1

3.4.1 Modelisation avec PGMPY

```
from pgmpy.factors.discrete import TabularCPD
from pgmpy.models import BayesianNetwork
# Etablissement de la structure
climate_model = BayesianNetwork([
    ('F', 'D'),
    ('C', 'D'),
    ('D', 'P'),
    ('C', 'M'),
])
# Définition des relations
fire_cpd = TabularCPD(
    variable='F',
    variable_card=2,
    values=[[.6], [.4]]
)

climate_cpd = TabularCPD(
    variable='C',
    variable_card=2,
    values=[[.7], [.3]]
)
mitigation_cpd = TabularCPD(
    variable='M',
    variable_card=2,
    values=[[.8, .9],
            [.2, .1]],
    evidence=['C'],
    evidence_card=[2]
)

deforestation_cpd = TabularCPD(
    variable='D',
    variable_card=2,
    values=[[.7, .8, .95, .8], [.3, .2, .05, .2]],
    evidence=['F', 'C'],
    evidence_card=[2, 2]
```

```
    )

    extreme_weather_cpd = TabularCPD(
        variable='P',
        variable_card=2,
        values=[[.9, .95], [.1, .05]],
        evidence=['D'],
        evidence_card=[2]
    )



    # Ajout des relations à notre modèle
    climate_model.add_cpds(deforestation_cpd, fire_cpd, climate_cpd, mitigation_cpd,␣
     ↪extreme_weather_cpd)

    # Re-vérification de la structure de notre modèle
    climate_model.get_cpds()
```

```
    [<TabularCPD representing P(D:2 | F:2, C:2) at 0x7a4d26c08f10>,
     <TabularCPD representing P(F:2) at 0x7a4de0d46e60>,
     <TabularCPD representing P(C:2) at 0x7a4de0d460b0>,
     <TabularCPD representing P(M:2 | C:2) at 0x7a4de0d448b0>,
     <TabularCPD representing P(P:2 | D:2) at 0x7a4de0d45ae0>]
```

[4]:
```
    from pgmpy.inference import VariableElimination
    # Calcul des probabilités existantes pour vérifier que tout est correct
    prob_deforestation = VariableElimination(climate_model).query(variables=['D'])
    print(prob_deforestation)
```

```
    +------+----------+
    | D    |   phi(D) |
    +======+==========+
    | D(0) |   0.8000 |
    +------+----------+
    | D(1) |   0.2000 |
    +------+----------+
```

[5]:

```
# Faire un autre calcul de probabilité
prob_extreme_weather = VariableElimination(climate_model).query(variables=['P'])
print(prob_extreme_weather)
```

```
+------+----------+
| P    |   phi(P) |
+======+==========+
| P(0) |   0.9100 |
+------+----------+
| P(1) |   0.0900 |
+------+----------+
```

[6]:
```
# Probabilité d'avoir le changement climatique si on a la déforestation
proba_climate_deforestation = VariableElimination(climate_model).
 ↪query(variables=['C'], evidence={'D': 1})
print(proba_climate_deforestation)
```

```
+------+----------+
| C    |   phi(C) |
+======+==========+
| C(0) |   0.7000 |
+------+----------+
| C(1) |   0.3000 |
+------+----------+
```

[7]:
```
# Probabilité d'avoir des mesures d'atténuation si on a le changement climatique
proba_mitigation_climate = VariableElimination(climate_model).
 ↪query(variables=['M'], evidence={'C': 1})
print(proba_mitigation_climate)
```

```
+------+----------+
| M    |   phi(M) |
+======+==========+
| M(0) |   0.9000 |
+------+----------+
| M(1) |   0.1000 |
+------+----------+
```

3.5 ETAPE3 : Multi-Arbre

Nous avons conçu un réseau bayésien pour modéliser la causalite entre les concepts de changement de climate.

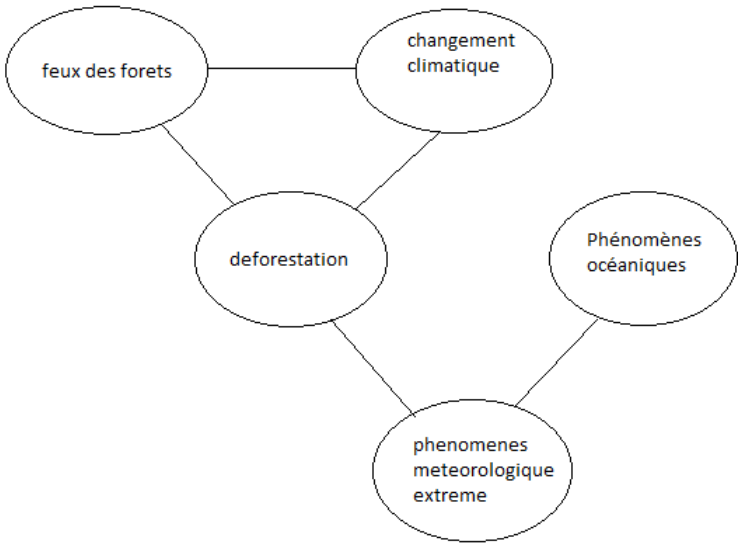

FIGURE 3.4 – Reseau bayesien des notions de changement de climat

La representation numerique des noeud :
on est dans le domaine probabiliste donc on peut faire la representation numerique respectons :

F	C	$P(F)$
Faux	Faux	0.6
Faux	Vrai	0.4
Vrai	Faux	0.3
Vrai	Vrai	0.7

(a)

C	$P(C)$
Faux	0.75
Vrai	0.25

(b)

C	F	D	$P(D\|C,F)$
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

(c)

O	$P(O)$
Faux	0.6
Vrai	0.4

(d)

P	O	P	$P(P\|D,O)$
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

(e)

FIGURE 3.5 – Description des tableaux

3.5.1 Modelisation avec PGMPY

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

# Establishing the structure
climate_model = BayesianNetwork([
    ('C', 'F'),
    ('C', 'D'),
    ('F', 'D'),
    ('D', 'P'),
    ('O', 'P'),
])
```

```
12
13 # Defining the relationships
14 changement_climate_cpd = TabularCPD(
15     variable='C',
16     variable_card=2,
17     values=[[.75], [.25]]
18 )
19
20 feu_cpd = TabularCPD(
21     variable='F',
22     variable_card=2,
23     values=[[.6, .3],
24             [.4, .7]],
25     evidence=['C'],
26     evidence_card=[2]
27 )
28
29 defostration_cpd = TabularCPD(
30     variable='D',
31     variable_card=2,
32     values=[
33         [.99, .8, .7, .4], [.01, .2, .3, .6]],
34     evidence=['C', 'F'],
35     evidence_card=[2, 2]
36 )
37
38 ocean_cpd = TabularCPD(
39     variable='O',
40     variable_card=2,
41     values=[[.6], [.4]]
42 )
43
44 phenomenes_cpd = TabularCPD(
45     variable='P',
46     variable_card=2,
47     values=[[.9, .4, .2, .3], [.1, .6, .8, .7]],
48     evidence=['O', 'D'],
49     evidence_card=[2, 2]
50 )
51
```

```
# Adding the relationships to our model
climate_model.add_cpds(changement_climate_cpd,
                       feu_cpd, defostration_cpd, ocean_cpd,
   phenomenes_cpd)

# Checking the structure of our model
climate_model.get_cpds()
```

```
[11] from pgmpy.inference import VariableElimination
     climate_infer = VariableElimination(climate_model)
     #calcul des proba existante pour vérifier que tout est correct
     prob_changement = climate_infer.query(variables=['C'])
     print(prob_changement)
```

```
+------+----------+
| C    |   phi(C) |
+======+==========+
| C(0) |   0.7500 |
+------+----------+
| C(1) |   0.2500 |
+------+----------+
```

```
#probabilité d'avoir une phenomene meteorologique quand il ya une defostration
proba_phenomene_def = climate_infer.query(variables=['P'], evidence= {'D':1})
print(proba_phenomene_def)
```

```
+------+----------+
| P    |   phi(P) |
+======+==========+
| P(0) |   0.3600 |
+------+----------+
| P(1) |   0.6400 |
+------+----------+
```

FIGURE 3.6 – Performance de notre modele multi-connected

3.6 ETAPE4 : Générez un graphe à connexions multiples avec un Grand Nombre de Variables et de Parents

```
import networkx as nx
import random

```

```python
def generate_random_bayesian_network(num_nodes, max_parents_per_node):
    # Cr ation d'un graphe dirige
    G = nx.DiGraph()

    # Ajout des noueds au graphe
    for i in range(num_nodes):
        G.add_node(f'X{i}')

    # Ajout des arcs entre les noeuds
    for i in range(num_nodes):
        num_parents = random.randint(0, min(i, max_parents_per_node))
        parents = random.sample(range(i), num_parents)
        for parent in parents:
            G.add_edge(f'X{parent}', f'X{i}')

    # Attribution de probabilites aleatoires aux CPDs
    for node in G.nodes():
        num_states = random.randint(2, 5)  # Nombre al atoire d' tats pour chaque variable
        states = [f'State{i}' for i in range(num_states)]
        cpd_values = [random.random() for _ in range(num_states)]
        cpd_values = [val / sum(cpd_values) for val in cpd_values]  # Normalisation des probabilit s
        G.nodes[node]['states'] = states
        G.nodes[node]['probabilities'] = cpd_values

    return G

# Param tres
num_nodes = 10  # Nombre total de variables
max_parents_per_node = 3  # Nombre maximal de parents par variable

# G n ration du r seau bay sien
random_bayesian_network = generate_random_bayesian_network(num_nodes, max_parents_per_node)

# Affichage du graphe
pos = nx.spring_layout(random_bayesian_network)
nx.draw(random_bayesian_network, pos, with_labels=True, font_weight='bold', node_size=700, node_color='skyblue', font_size=8)
```

```
40 nx.draw_networkx_edge_labels(random_bayesian_network, pos, edge_labels={(
       i, j): '' for i, j in random_bayesian_network.edges()})
```

3.7 Conclusion

En conclusion, l'étude approfondie des réseaux bayésiens polyarbres et multiconnectés a permis de modéliser avec précision les dépendances complexes entre variables. L'analyse des distributions conditionnelles des probabilités des nœuds a enrichi notre compréhension des relations probabilistes.

TP4 : Logique possibiliste qualitative

4.1 Introduction

La logique possibiliste qualitative vise à traiter l'incertitude et l'imprécision dans le raisonnement logique en utilisant des degrés de certitude plutôt que des valeurs binaires (vrai/faux). Dans ce contexte, le problème de déduction se pose comme suit :

1. **Objectif de la déduction possibiliste :** Calculer le degré d'inconsistance (Incons) d'un ensemble de connaissances Σ étendu par la négation d'une proposition ϕ (notée ici comme $\neg\phi$).
2. **Mesure d'inconsistance :** La mesure d'inconsistance est définie comme Incons($\Sigma \cup \{\neg\phi\}$) = val(ϕ, Σ), où val(ϕ, Σ) représente le degré de certitude que la proposition ϕ est vraie dans le contexte de connaissances Σ.
3. **Utilisation du principe de réfutation :** La déduction se base sur le principe de réfutation. Prouver que Σ infère (ϕ, α) est équivalent à prouver que l'ensemble étendu $\Sigma \cup \{\neg\phi\}$ est inconsistant, c'est-à-dire qu'il conduit à une contradiction.
4. **Test d'inconsistance :** Le test d'inconsistance est matérialisé par l'utilisation de prouvers SAT (*satisfiability*) qui sont des outils algorithmiques utilisés pour déterminer la satisfaisabilité d'une formule logique. Si l'ensemble étendu $\Sigma \cup \{\neg\phi\}$ est satisfaisable, alors $\Sigma \wedge \neg\phi$ est consistant, et le processus de dichotomie ajuste la borne supérieure (u) en conséquence. Sinon, si $\Sigma \wedge \neg\phi$ est inconsistant, le processus ajuste la borne inférieure (l).
5. **Principe de dichotomie :** L'algorithme d'inférence utilise le principe de dichotomie pour réduire l'intervalle $[l, u]$ jusqu'à ce que la borne inférieure l soit égale à la borne supérieure u. À chaque étape, l'algorithme divise l'intervalle en deux, teste l'inconsistance de $\Sigma^* \wedge \neg\phi$, et ajuste les bornes en conséquence.
   En résumé, l'algorithme cherche à trouver le degré de certitude (α) pour lequel l'ensemble étendu $\Sigma \cup \{\neg\phi\}$ devient inconsistant, en utilisant une approche de recherche dichotomique pour réduire l'intervalle des valeurs possibles. L'utilisation de prouvers SAT facilite le test d'inconsistance dans ce contexte de logique possibiliste qualitative.

Voici le pseudo code de l'algorithme :

Algorithme 1 : Algorithme de Dichotomie pour l'Inférence en Logique Possibiliste

Input : n, Σ, ϕ
Output : l, u
Initialization : $l := 0$, $u := n$
tant que $l < u$ **faire**
  $r := \lfloor \frac{l+u}{2} \rfloor$
  si $\Sigma^* \wedge \neg\phi$ *consistent* **alors**
    $u := r - 1$
  fin si
  sinon
    $l := r$ $\{\text{Val}(\phi, \Sigma) = \alpha_r\}$
  fin si
fin tq

4.2 La création de la base de connaissances pondérée

```python
import random
import os

class BaseConnaissance:
    number_of_clause = 0
    number_of_variable = 0
    evidence_length = 0
    var_list = []
    max_clause_length = 0

    def __init__(self, nb_clause, nb_variable, length, max_number):
        self.number_of_clause = nb_clause
        self.number_of_variable = nb_variable
        self.evidence_length = length
        self.max_clause_length = max_number
        for i in range(self.number_of_variable):
            self.var_list.append(i + 1)
            self.var_list.append(-(i + 1))

    def get_evidence(self):
        return random.sample(self.var_list, self.evidence_length)

    def get_sigma(self):
        with open("temp.txt", "w") as sigma_file:
```

```
            for i in range(self.number_of_clause):
                ran = random.randrange(1, self.max_clause_length + 1, 1)
                temp_list = random.sample(self.var_list, ran)
                weight = random.random()
                weight = float(int(weight * 100)) / 100
                chaine = "" + str(weight) + " "
                for j in range(len(temp_list)):
                    chaine = chaine + str(temp_list[j]) + " "
                sigma_file.write(chaine + "\n")
                temp_list = []
```

4.3 L'implémentation de l'algorithme en utilisant un prouver SAT

```
class Sigma:
    lower = 0
    upper = 0
    weights = []
    formulas = []
    strates_weights = []

    def __init__(self, path_file):
        with open(path_file) as file:
            for line in file:
                information = line.split()
                self.weights.append(float(information[0]))
                self.formulas.append([int(information[i]) for i in range
   (1, len(information))])
        self.length = len(self.weights)

    def get_length(self):
        return self.length

    def get_weights(self):
        return self.weights

    def get_formulas(self):
        return self.formulas
```

```
24
25      def sort_weights(self):
26          sorted_data = sorted(zip(self.weights, self.formulas), reverse=
    True)
27          self.weights, self.formulas = zip(*sorted_data)
28
29      def compute_strates(self):
30          self.upper = len(set(self.weights))
31          self.strates_weights = sorted(set(self.weights))
32
33      def get_strates_number(self):
34          return self.upper
35
36      def get_strates_weights(self):
37          return self.strates_weights
38
39      def get_preprocessed_formulas(self, sub_formulas):
40          dict_cor = {}
41          returned_formulas = []
42          cpt = 1
43
44          for form in sub_formulas:
45              temp_form = []
46              for pred in form:
47                  if pred not in dict_cor:
48                      dict_cor[pred] = cpt
49                      dict_cor[-pred] = -cpt
50                      cpt += 1
51                  temp_form.append(dict_cor[pred])
52              returned_formulas.append(temp_form)
53
54          return returned_formulas
```

4.4 L'algorithme calcule la variable d'intérêt Π, qui correspond à $val(\phi, \Sigma)$

```
1 import pycosat
2 from prettytable import PrettyTable
```

```
3
4  choice = 1
5  if choice == 1:
6      base_de_connaissance = Sigma("/content/drive/MyDrive/data/BD.wcnf")
7      evidence = [3, 4, -7]
8  else:
9      nb_clause = 8
10     nb_variable = 5
11     length = 3
12     max_number = 4
13     gen = BaseConnaissance(nb_clause, nb_variable, length, max_number)
14     gen.get_sigma()
15     base_de_connaissance = Sigma("temp.txt")
16     evidence = gen.get_evidence()
17
18 base_de_connaissance.sort_weights()
19 base_de_connaissance.compute_strates()
20 iteration = 1
21
22 while base_de_connaissance.lower < base_de_connaissance.upper:
23     r = int((base_de_connaissance.lower + base_de_connaissance.upper + 1)
       / 2)
24
25     liste = base_de_connaissance.get_weights()
26     value_of_r = -1
27     for i in range(len(liste)):
28         if base_de_connaissance.get_strates_weights()[r - 1] > liste[i]:
29             value_of_r = i
30             break
31     if value_of_r == -1:
32         value_of_r = len(liste) - 1
33
34     for j in range(len(liste)):
35         if base_de_connaissance.get_strates_weights()[
     base_de_connaissance.upper - 1] == liste[j]:
36             valueU = j
37             break
38
39     cnf = base_de_connaissance.formulas[valueU:value_of_r]
40
```

```
    for i in range(len(evidence)):
        liste = []
        liste.append(-1 * evidence[i])
        cnf = list(base_de_connaissance.get_preprocessed_formulas(cnf))

    cnf = base_de_connaissance.get_preprocessed_formulas(cnf)

    result = pycosat.solve(cnf)
    if type(result) == type([]):
        base_de_connaissance.upper = r - 1
    else:
        base_de_connaissance.lower = r
    iteration = iteration + 1

Val = base_de_connaissance.get_strates_weights()[r - 1]

with open("/content/drive/MyDrive/data/BD.wcnf") as f:
    content = f.read().splitlines()
table = PrettyTable()
table.field_names = ["Poids", "PropositionP1", "PropositionP2"]
for line in content:
    row_values = line.split()
    while len(row_values) < len(table.field_names):
        row_values.append('')
    table.add_row(row_values)

print(table)
print("**L evidence de : ", evidence)
print("**Le resultat : Val(" + str(evidence) + ", Sigma) = " + str(Val))
```

Le résultat est le suivant :

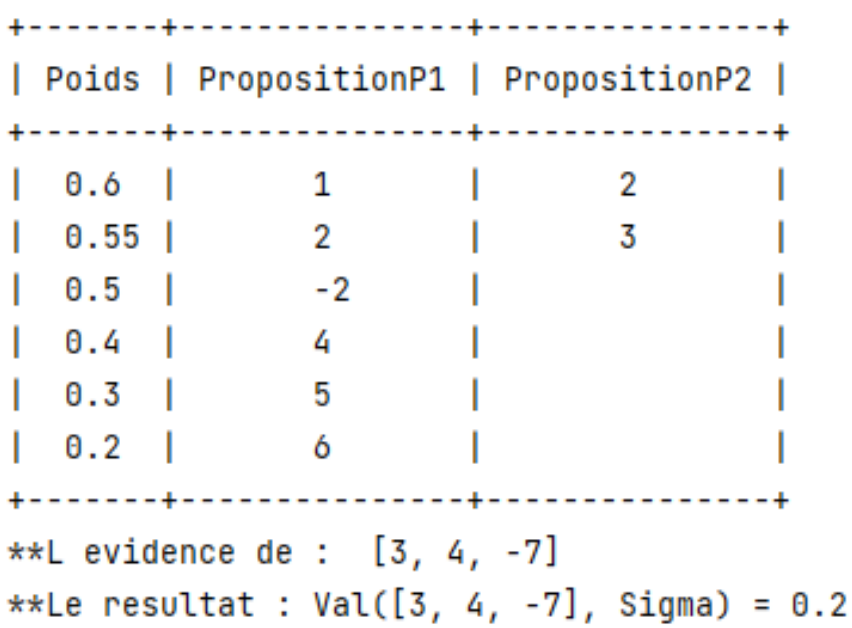

FIGURE 4.1 – Resultat d'inerence

4.5 Conclusion

Le but de ce TP en logique possibiliste qualitative est de traiter le problème de déduction dans ce contexte. Plus précisément, le TP se focalise sur l'algorithme d'inférence qui utilise le principe de dichotomie pour calculer le degré d'inconsistance d'une base de connaissances pondérées Σ.

TP 5 : Théorie des possibilités : Inférence logique et propagation graphique

5.1 Introduction

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 faiblemen connectés, Graphes moyennement connectés et Graphes fortement connectés.

5.2 Utilisation de Cygwin

Le premier programme est lié au graphe possibiliste basé sur le produit et à la base de connaissance possibiliste quantitative correspondante. Lorsque nous exécutons le programme "inference.exe" pour initier le processus d'inférence, cela implique le calcul du degré de possibilité de l'instance de la variable d'intérêt, ainsi que la mesure du temps d'inférence. Pour exécuter ces deux programmes, "passage.exe" et "inference.exe", nous utiliserons Cygwin.

5.2.1 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.

5.2.2 ÉTAPE 2 : Exécution des fichiers

```
VLADE@DESKTOP-HUHORPD /home
$ cd licence

VLADE@DESKTOP-HUHORPD /home/licence
$ ./passage.exe

VLADE@DESKTOP-HUHORPD /home/licence
$ ./inference.exe
236975
VLADE@DESKTOP-HUHORPD /home/licence
$ cat resultats
********R□sultats de la propagation graphique********
nombre de variables: 38
nombre de parentsmax: 3
l'evidence: -31
variable d'interet: 25
possibilit□ conditionnelle de l'interet | evidences 0.018316
temps de propagation 0.220853  secondes

********R□sultats de l'inf□rence logique********
le nombre de clauses dans les bases est de: 146
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 4
 avec un degr□ de possibilit□ correspondant est □gale □ :0.018316
la dur□e de l'inf□rence est: 236975      millisecondes
VLADE@DESKTOP-HUHORPD /home/licence
$
```

FIGURE 5.1 – Résultats de l'exécution du fichier 1

5.2.3 Evaluation

Nous avons effectué des évaluations sur les nombres de nœuds 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 nœuds, 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.

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

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 d'inference 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 Generale

Ces travaux pratiques nous ont permis d'explorer les diverses logiques sur lesquelles un système informatique peut fonctionner. Nous avons acquis des compétences en modélisation de problèmes réels, en génération d'inférences, en utilisation de raisonneurs pour évaluer la validité de propositions, en représentation graphique d'informations, et en déduction de faits.