République Algérienne Démocratique et Populaire

Ministère de L'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'informatique

Master 2 Systèmes Informatiques intelligents

Rapport des TPs Représentation des connaissance et raisonnement 2

Présenté par :

MOHAND LHADJ	Damia	191936003873
LOUNAS	Katia	191931012680

Année universitaire : 2023 / 2024

Table des matières

Table des figures i

Liste des tableaux iii

Introduction Générale 1

1 TP 1 : La théorie des fonctions de croyance 2
- 1.1 Introduction . . . 2
- 1.2 Théorie des fonctions de croyances . . . 2
- 1.3 Modélisations d'exemples de la théorie des fonctions de croyances . . . 3
  - 1.3.1 En utilisant le moteur Dempster-shafer . . . 3
  - 1.3.2 En utilisant l'outil Pyds . . . 8
- 1.4 Conclusion . . . 12

2 TP2 : Contrôleurs flous 13
- 2.1 Introduction . . . 13
- 2.2 La théorie des ensembles flous . . . 13
- 2.3 Conception et implémentation d'un contrôleur flou . . . 14
  - 2.3.1 En utilisant *Fuzzy Logic* de Matlab . . . 14
- 2.4 Conclusion . . . 18

3 TP3 : Réseaux causaux Bayésiens 19
- 3.1 Introduction . . . 19
- 3.2 Les réseaux causaux bayésiens . . . 19
- 3.3 Conception et implémentation des réseaux causaux bayésiens . . . 20
  - 3.3.1 En utilisant Bayes Net Toolbox (BNT) de Matlab . . . 20
  - 3.3.2 En utilisant Probabilistic Graphical Models de Python . . . 29
- 3.4 Conclusion . . . 35

4 TP4 : Logique possibiliste qualitative 36
- 4.1 Introduction . . . 36
- 4.2 La logique possibiliste qualitative . . . 36
- 4.3 Exploration de l'outil . . . 36
  - 4.3.1 Génération de la base de connaissance . . . 36
  - 4.3.2 Implémentation de l'algorithme d'inférence . . . 37

4.5 Conclusion 41

5 Inférence logique et propagation graphique en théorie des possibilités 42

5.1 Introduction 42
5.2 Théorie des possibilités et propagation graphique 42
5.3 Installation 43
5.3.1 Sous Windows 43
5.3.2 Sous Cygwin 43
5.4 Expérimentation 43
5.4.1 Une seule évidence 44
5.4.2 Deux évidence 44
5.5 Conclusion 45

Conclusion Générale 46

Table des figures

1.1 Dempster-Shafer engine . . . 3
1.2 Etape 1 : Insertions des sources de données . . . 6
1.3 Etape 2 : Insertions de la liste de suspects . . . 6
1.4 Etape 3 : Saisie des distribution de masse . . . 6
1.5 Diagramme circulaire de la distribution de masse . . . 7
1.6 Étape 4 : Combiner les sources d'expertise . . . 7
1.7 Diagramme représentant les masses combinées . . . 7
1.8 Graphique de la visualisation de la distribution de masse d'EGFF . . . 11
1.9 Graphique de fusion des experts . . . 12

2.1 Fuzzy Logic designer . . . 14
2.2 Ajout des paramètres d'entrées . . . 16
2.3 Ajout des fonctions d'appartenance . . . 17
2.4 Saisi de la base de règles . . . 17
2.5 Sorties flous . . . 18

3.1 Représentation du réseau Polyarbre . . . 20
3.2 Génération du polyarbre . . . 21
3.3 Les variables d'évidence et la variable d'intérêt du polyarbre . . . 21
3.4 Les distributions a priori pour les nœuds racine du polyarbre . . . 21
3.5 Les distributions conditionnelles pour les autres nœuds du polyarbre . . . 22
3.6 Calcule de p(variable d'intérêt | évidences) du polyarbre . . . 22
3.7 La distribution marginale de la variable T . . . 22
3.8 Affichage du polyarbre . . . 23
3.9 Représentation du graphe à connexions multiples . . . 23
3.10 Génération du graphe à connexions multiples . . . 24
3.11 Les variables d'évidence et la variable d'intérêt du graphe à connexions multiples . . . 24

3.12 Les distributions a priori pour les nœuds racine du graphe à connexions multiples . . . 24
3.13 Les distributions conditionnelles pour les autres nœuds du graphe à connexions multiples . . . 25
3.14 Calcule de p(variable d'intérêt | évidences) du graphe à connexions multiples 25
3.15 La distribution marginale de la variable PC . . . 26
3.16 Affichage du graphe à connexions multiples . . . 26
3.17 Génération de la structure du graphe . . . 27
3.18 Génération du réseau bayésien . . . 27
3.19 Génération des distributions des probabilités . . . 27
3.20 Affichage des distributions . . . 28
3.21 Affichage du graphe . . . 28
3.22 Resultat P(variable d'intérêt | évidences) . . . 30
3.23 Graphe polyarbre . . . 31
3.24 Resultat P(variable d'intérêt | évidences) . . . 32
3.25 Graphe muticonnected . . . 33
3.26 Graphe généré aléatoirement . . . 35
3.27 Distributions conditionnelles générées aléatoirements . . . 35

4.1 Résultat de l'exécution de l'algorithme d'inférence . . . 40
4.2 Graphique d'évolution des bornes inférieure et supérieure a travers les itérations . . . 41

5.1 Graphique des temps d'inférence et de propagation des expérimentations . 45

Liste des tableaux

1.1 Table des distribution des masses : Agent de sécurité 5
1.2 Table des distribution des masses : Log Journals 5
1.3 Table des distribution des masses : Charles 5
1.4 Table des distribution des masses : Caméras de surveillance 5
1.5 Table des distribution des masses : Clément 5
1.6 Table de distribution des masses : Experts en gestion des feux de forêt (EGFF) 9
1.7 Table de distribution des masses : Responsables protection de l'environnement (RPE) 9
1.8 Table de distribution des masses : Écologistes des forestière (EF) 9
1.9 Table de distribution des masses : Opinion publique (OP) 9
1.10 Table de distribution des masses : Expert en analyse des feux de foret (EAFF) 9

2.1 Paramètre d'entrée NM 15
2.2 Paramètre d'entrée PF 15
2.3 Paramètre d'entrée SP 15
2.4 Paramètre de sortie RF 16
2.5 La matrice d'inférence 16

5.1 Tbleau des expérimentations avec une seul évidence 44
5.2 Tbleau des expérimentations avec deux évidence 44

Introduction Générale

Le module que nous avons étudié propose une exploration approfondie des différentes théories et concepts liés à la gestion des connaissances imparfaites et incertaines. Ce domaine explore les méthodes et les modèles permettant de représenter efficacement des connaissances qui ne sont pas entièrement précises ou certaines, et de raisonner de manière logique en tenant compte de ces imperfections. À travers une série de Travaux Pratiques (TPs) éclairants, nous avons abordé des sujets fondamentaux tels que la théorie des fonctions de croyance, la logique floue, les Réseaux causaux Bayésiens, la logique possibiliste qualitative, l'inférence logique, et la propagation graphique.

Ces TPs visent à fournir une expérience pratique complète pour mieux comprendre et utiliser les outils disponibles dans le traitement des données incertaines.

Chapitre 1

TP 1 : La théorie des fonctions de croyance

1.1 Introduction

La théorie des fonctions de croyance constitue une approche formelle visant à modéliser l'incertitude et la gestion des croyances dans un contexte informatique. Elle propose un cadre pour représenter et raisonner sur des informations incertaines en utilisant des fonctions de masse de croyance. Au cours de ce TP, nous examinerons de manière approfondie des boîte à outils spécifiquement conçue pour cette théorie. Nous développerons des exemples concrets afin d'illustrer comment cette approche peut être mise en œuvre pour aborder l'incertitude dans divers domaines.

1.2 Théorie des fonctions de croyances

La théorie des fonctions de croyance, également appelée théorie de Dempster-Shafer, généralise l'inférence Bayésienne en l'absence d'apriori sur les paramètres. Shafer a formalisé cette approche en 1976, montrant son utilité pour modéliser des connaissances incertaines de manière plus naturelle que la théorie des probabilités, qui ne traite pas efficacement l'absence d'information.

Les distributions de probabilités sont construites non pas sur Ω mais sur l'ensemble des parties de Ω. Ω est dit cadre de discernement représentant l'ensemble exhaustif et exclusif des états connus du monde. 2^{Ω} est l'ensemble des parties de Ω. La distribution représente la masse de probabilités m définie par :

$$m : 2^{\Omega} \rightarrow [0, 1]$$

Les propriétés de la fonction de masse sont :

$$\sum_{A \subseteq \Omega} m(A) = 1$$

$$m(\varnothing) = 0$$

Tout sous-ensemble de Ω vérifiant $m(A) \neq 0$ est dit élément focal de m.

1.3 Modélisations d'exemples de la théorie des fonctions de croyances

1.3.1 En utilisant le moteur Dempster-shafer

Dans cette partie nous allons exploiter l'outil "Dampster-Shafer engine", disponible sur le lien suivant : *https://www.softpedia.com/get/Science-CAD/Dempster-Shafer-Engine.shtml* pour modéliser la théorie des fonctions de croyance.

1.3.1.1 Présentation de l'outil moteur Dempster-Shafer

Le moteur Dempster-Shafer est un logiciel qui permet d'analyser des situations en rassemblant des informations provenant de différentes sources, puis en combinant ces informations de manière statistiquement précise.

Il est basé sur la théorie des preuves de Dempster-Shafer, un cadre mathématique permettant de gérer l'incertitude et de combiner des données provenant de différentes sources.

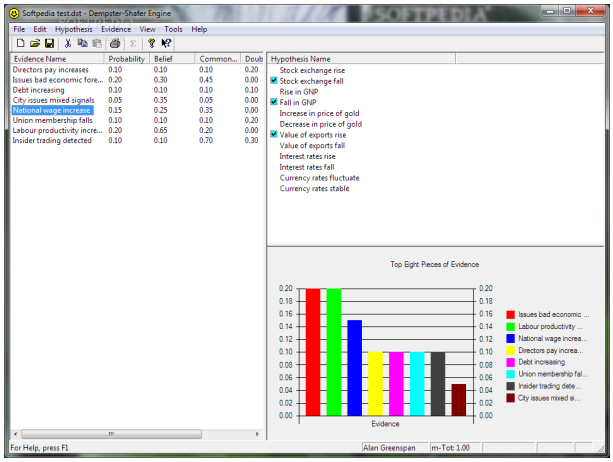


Figure 1.1: Dempster-Shafer engine

1.3.1.2 Exemple à modéliser :

Une cyberattaque a été lancée contre le système informatique d'une grande entreprise, après une enquête il s'est avéré que l'attaque a été lancée depuis l'intérieur du système et 4 employés sont suspectés : Maxime, Antoine, Jean et Sandrine. Les indices et témoignages recueillis sont comme suit :

1. L'agent de sécurité affirme que les quatre employés se trouvaient dans le département IT a l'heure de l'attaque et qu'ils ont donc des probabilités égales d'être à l'origine de l'attaque.
2. L'étude des log journals des suspects qui est fiable a 64% a relevée une activité suspecte sur les compte de Jean et Antoine.
3. Charles ; un employé a affirmé que Maxime et Jean avaient des raisons d'en vouloir à l'entreprise au point d'attaquer le système ; Maxime a 46% et Jean a 32%, mais il exclut totalement l'hypothèse que ce soit Antoine en lui offrant un alibi.
4. Les caméras de surveillance offrant une preuve tangible à 77% que Jean et Sandrine ont quitté leurs postes de travail au moment de l'attaque.
5. Clément ; le responsable de l'équipe IT affirme avoir eu un altercation avec Maxime au sujet d'une promotion qu'il n'a pas eu durant laquelle il a dit "Vous allez voir que cette entreprise va le payer. Vous allez voir que je mérite cette promotion". Ce témoignage est fiable a 54%.

1.3.1.3 Distribution des masses :

Ω = { Maxime, Antoine, Jean, Sandrine }

Suspect	Maxime	Antoine	Jean	Sandrine
Masse	0.25	0.25	0.25	0.25

Tableau 1.1: Table des distribution des masses : Agent de sécurité

Suspect	{Antoine, Jean}	Ω
Masse	0.64	0.36

Tableau 1.2: Table des distribution des masses : Log Journals

Suspect	Maxime	Jean	{Maxime, Jean, Sandrine}
Masse	0.46	0.32	0.22

Tableau 1.3: Table des distribution des masses : Charles

Suspect	{Sandrine, Jean}	Ω
Masse	0.77	0.23

Tableau 1.4: Table des distribution des masses : Caméras de surveillance

Suspect	Maxime	Ω
Masse	0.54	0.46

Tableau 1.5: Table des distribution des masses : Clément

1.3.1.4 Étapes de modélisation :

1. **Insertions des sources de données :** On commence par insérer la liste des témoins et source des preuves recueillie

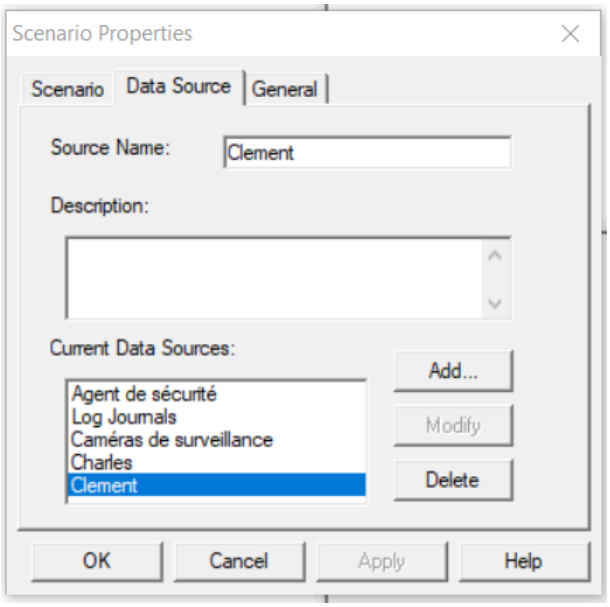


Figure 1.2: Etape 1 : Insertions des sources de données

2. **Insertions de la liste de suspects :** On saisi la liste des suspects en clinquant sur *"New hypothesis"*

Figure 1.3: Etape 2 : Insertions de la liste de suspects

3. **Saisie des distribution de masse :** Pour chaque source de données on saisi les masse de croyance attribuer au différant ensembles en clinquant sur *"New Evidence"*

Evidence Name	Probability	Belief	Common...	Doubt	UPF
Max	0.460	0.460	0.680	0.320	0.680
Jean	0.320	0.320	0.540	0.460	0.540
{Jean,Max,Sand}	0.220	1.000	0.220	0.000	1.000

Figure 1.4: Etape 3 : Saisie des distribution de masse

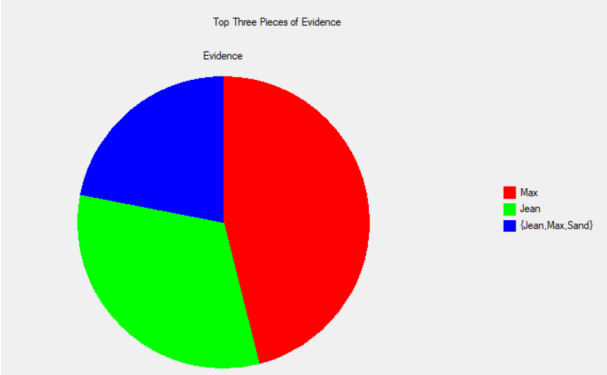


Figure 1.5: Diagramme circulaire de la distribution de masse

4. **Combiner les sources d'expertise :** On combine les sources de données qu'on a saisi dans le logiciel en clinquant sur *"Tools > Combine Evidence"*, On obtient alors les résultats représentant les suspects qui ont le plus de probabilité d'être a l'origine de l'attaque. On peut visualiser les résultats sous forme de tableau, diagramme circulaire ou diagramme de bars.

Evidence Name	Probability	Belief	Common...	Doubt	UPF
Sand	0.107	0.107	0.107	0.893	0.107
Jean	0.728	0.728	0.728	0.272	0.728
Max	0.165	0.165	0.165	0.835	0.165

Figure 1.6: Étape 4 : Combiner les sources d'expertise

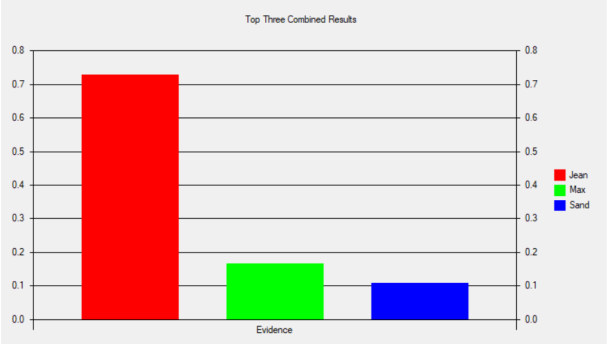


Figure 1.7: Diagramme représentant les masses combinées

1.3.1.5 Interprétation des résultats :

Les résultats de la combinaison des sources d'expertise et d'évidences recueillies montre que Jean a une plus grande probabilité d'être le coupable et se à 72.8% suivi de Maxime à 16.5% puis Sandrine à 10.7%.

1.3.2 En utilisant l'outil Pyds

Dans cette partie, nous allons vous présenter un outil puissant et convivial : Pyds (Python for Dempster-Shafer), disponible sur le lien suivant : *https://github.com/reineking/pyds*.

1.3.2.1 Présentation de la bibliothèque Pyds

Il s'agit d'une bibliothèque Python spécialisée qui simplifie la manipulation et la gestion des fonctions de croyance, un concept fondamental de la théorie de Dempster-Shafer. Pyds offre une approche plus expressive pour modéliser l'incertitude par rapport aux méthodes traditionnelles comme la probabilité. Cette bibliothèque a été spécialement conçue pour accompagner les chercheurs, les professionnels et les ingénieurs dans la gestion de systèmes d'information complexes, où l'incertitude et la croyance jouent un rôle crucial. Cette bibliothèque offre des fonctionnalités essentielles, parmi lesquelles la création de fonctions de croyance, combinaison de ces fonctions, le calcul de la croyance et de la plausibilité, ainsi que des outils de visualisation.

1.3.2.2 Exemple à modéliser

Suite à une série d'incendies de forêt récents dans une région forestière, une équipe d'enquête a été chargée de déterminer les causes probables de ces incendies. Voici les indices et les témoignages recueillis :

1. Les experts en gestion des feux de forêt estiment que l'origine des incendies est due :
   - À la foudre à 50%, en raison de conditions météorologiques extrêmement sèches,
   - À des activités humaines imprudentes (telles que des feux de camp non surveillés et des déchets brûlés) à 30%,
   - À des causes inconnues à 20%, car certaines zones ne présentaient pas de risques évidents.
2. Les responsables locaux de la protection de l'environnement pensent que les incendies sont principalement dus à des activités humaines imprudentes à 60%. Ils soulignent l'importance de la sensibilisation du public pour éviter de futurs incendies dus à la négligence humaine.
3. Les scientifiques en écologie forestière suggèrent que la combinaison de la foudre et de la végétation extrêmement sèche est responsable à 70%. Ils estiment que, même si des activités humaines ont pu contribuer, la nature aride de la région est le facteur déterminant.

4. L'opinion publique, pour l'instant, demeure divisée et incertaine quant aux véritables causes des incendies de forêt, ce qui a suscité un débat animé sur la nécessité de renforcer la prévention et la sensibilisation dans la région.
5. D'après l'expert en analyse des incendies de forêt qui a mené une étude détaillée des incendies récents dans la région. Les incendies sont dus à l'activité humaine et les conditions météorologiques extrêmement sèches qui favorisent ces feux de camp non surveillés à 60%.

1.3.2.3 Distribution des masses :

Ω = { FN , AH}
Soit : FN = facteurs-naturels, AH = activités-humaines

Cause	FN	AH	Ω
Masse	0.5	0.3	0.2

Tableau 1.6: Table de distribution des masses : Experts en gestion des feux de forêt (EGFF)

Cause	AH	Ω
Masse	0.6	0.4

Tableau 1.7: Table de distribution des masses : Responsables protection de l'environnement (RPE)

Cause	FN	Ω
Masse	0.7	0.3

Tableau 1.8: Table de distribution des masses : Écologistes des forestière (EF)

Cause	FN	AH
Masse	0.5	0.5

Tableau 1.9: Table de distribution des masses : Opinion publique (OP)

Cause	FN, AH	Ω
Masse	0.6	0.4

Tableau 1.10: Table de distribution des masses : Expert en analyse des feux de foret (EAFF)

1.3.2.4 Étapes de modélisation :

1. **Importation des librairies**

```
from pyds import MassFunction
```

2. **Création de la distribution de masse :**

```
# Utilisation d'une liste de tuples
EGFF = MassFunction([({'FN'}, 0.5), ({'AH'}, 0.3), ({'FN', 'AH
                                    '}, 0.2)])
print('EGFF =', EGFF)
EF = MassFunction([({'FN'}, 0.7), ({'FN', 'AH'}, 0.3)])
print('EF =', EF)
OP = MassFunction([({'FN'}, 0.5), ({'AH'}, 0.5)])
print('OP =', OP)
EAFF = MassFunction([({'FN', 'AH'}, 0.6), ({'FN', 'AH'}, 0.4)]
                                    )
print('EAFF =', EAFF)
RPE = MassFunction()  #
RPE[{'AH'}] = 0.6
RPE[{'FN', 'AH'}] = 0.4
print('RPE =', RPE)
```

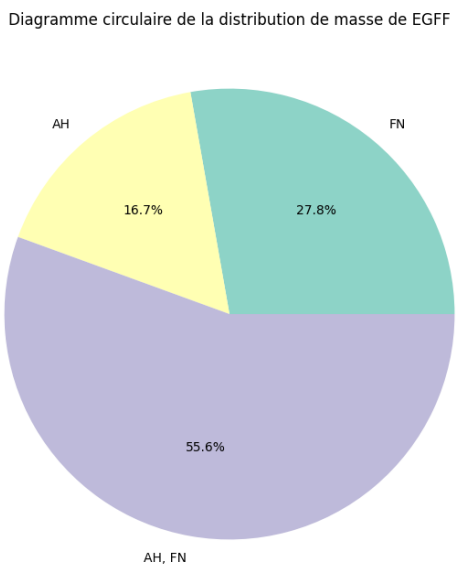
3. **Calcul de croyance et plausibilité pour chaque expert :**

```
# Affichage de croyance et plausibilit  pour chaque expert
print('\n=== La croyance et la plausibilite ===')
print('bel_1({FN, AH}) =', EGFF.bel({'FN', 'AH'}))
print('pl_1({FN, AH}) =', EGFF.pl({'FN', 'AH'}))
print('bel_1({FN, AH}) =', EF.bel({'FN'}))
print('pl_1({FN, AH}) =', EF.pl({'FN'}))
```

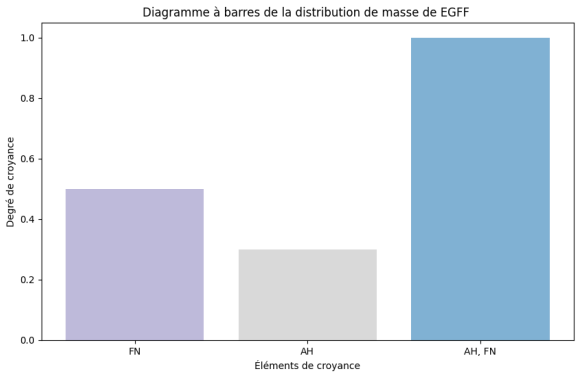
4. **Combinaisons les sources d'expertise :**

```
#combinaisons des masses par fusion de Dempster-Shafer
print('\n=== Combinaisons des masses par fusion de Dempster-
                                    Shafer ===')
print('Combinaisons des masses EGFF et EF =', EGFF & EF)
print('Combinaisons des masses EGFF et OP =', EGFF & OP)
print('Combinaisons des masses EGFF ET EAFF =', EGFF & EAFF)
print('Combinaisons des masses EGFF ET RPE =', EGFF & RPE)
print('\n=== Combinaisons des masses conjunctive de Dempster-
                                    Shafer ===')
print('Combinaisons des masses de EGFF, EAFF, and OP =', EGFF.
                                    combine_conjunctive(EAFF,
                                    OP))
```

```
print('Combinaisons des masses de EF, EAFF, and OP =', EF.
                                        combine_conjunctive(EAFF,
                                        OP))
comb1 = EGFF & EF
comb2 = RPE & OP
comb3 = EF & comb1
comb4 = comb2 & comb3
print('Combinaisons des masses de EF, EGFF, RPE, and OP =',
                                        comb4)
```



(a) Graphique circulaire

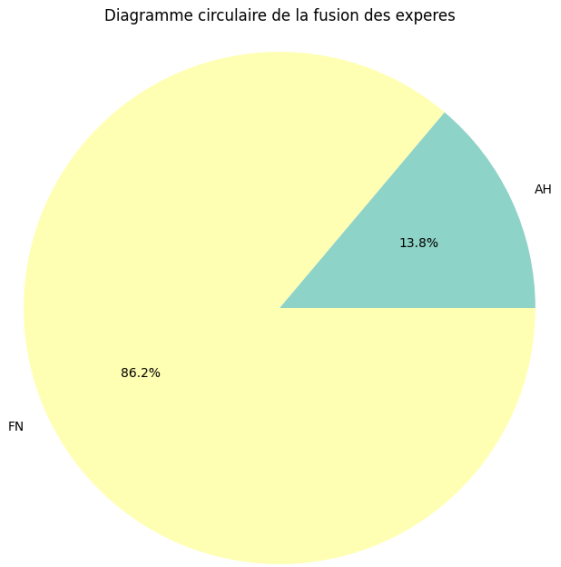


(b) Graphique de barres

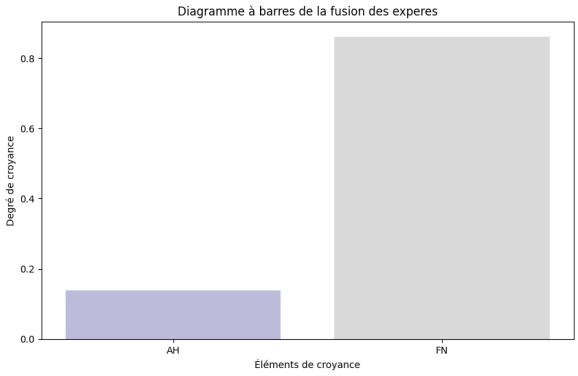
Figure 1.8: Graphique de la visualisation de la distribution de masse d'EGFF

Cette FIGURE 1.8 montre une visualisation de la distribution de masse des Experts en gestion des feux de forêt.

1.3.2.5 Fusion des Experts



(a) Graphique circulaire



(b) Graphique de barres

Figure 1.9: Graphique de fusion des experts

1.3.2.6 Interprétation des résultats :

Les résultats de la combinaison des sources d'expertise et d'évidences recueillies montre que la cause plus soutenue des incendies de foret et les facteurs naturels à 86.2% puis les activités humaines à 13.8%.

1.4 Conclusion

En résumé, la théorie des fonctions de croyance, explorée en profondeur au cours de ce TP, se révèle comme une approche formelle essentielle pour la modélisation de l'incertitude en informatique. Grâce à l'utilisation de deux outils clés : DSE (Dempster-Shafer Engine) et la bibliothèque Pyds et à travers des exemples concrets, nous avons mis en lumière sa capacité à représenter et raisonner sur des informations incertaines de manière robuste.

Chapitre 2

TP2 : Contrôleurs flous

2.1 Introduction

Ce chapitre explore la conception d'un contrôleur flou à l'aide de la "Fuzzy Toolbox" de MATLAB. La logique floue, basée sur la théorie des ensembles flous, répond à la nécessité de modéliser des connaissances imprécises et vagues. Cette approche est motivée par le constat que de nombreux objets physiques ne peuvent être catégorisés de manière nette. L'utilisation de la "Fuzzy Toolbox" permettra de concrétiser ces concepts dans le contexte de la conception d'un contrôleur flou.

2.2 La théorie des ensembles flous

La théorie des ensembles flous, développée par Lotfi Zadeh, est une extension de la théorie des ensembles traditionnels qui permet de représenter des concepts imprécis et vagues. Contrairement aux ensembles classiques où un élément est soit membre, soit non-membre, les ensembles flous autorisent des degrés graduels d'appartenance. Elle trouve des applications dans la modélisation de l'incertitude et de la subjectivité, en particulier dans des domaines où les frontières entre catégories sont difficiles à définir de manière précise.

2.3 Conception et implémentation d'un contrôleur flou

2.3.1 En utilisant *Fuzzy Logic* de Matlab

2.3.1.1 Présentation de l'outil *Fuzzy Logic* de Matlab

La Fuzzy Logic Toolbox de MATLAB est un ensemble d'outils logiciels spécialisés permettant la conception, la simulation et la mise en œuvre de systèmes basés sur la logique floue. Cet outil offre des fonctionnalités pour créer des systèmes flous, définir des règles d'inférence floue, effectuer des opérations de défuzzification, et simuler le comportement de systèmes complexes utilisant des concepts de logique floue. Il facilite ainsi la modélisation de l'incertitude et la prise de décisions dans des environnements où la logique traditionnelle pourrait être limitée.

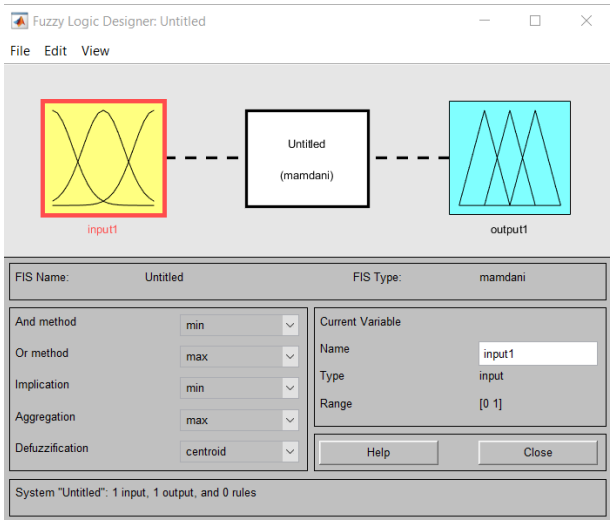


Figure 2.1: Fuzzy Logic designer

2.3.1.2 Exemple à modéliser :

Considérons un système de recommandation de films pour une plateforme de streaming basé sur un contrôleur flou. Les paramètres d'entrée incluent la note moyenne des utilisateurs pour un film donné (NM), la popularité du film (PF), et la similarité entre les préférences de l'utilisateur et celles d'autres utilisateurs (SP). Les paramètres sont spécifiés par des ensembles flous, et le paramètre de sortie est la recommandation du film (RF), exprimé à travers les ensembles flous suivants :

Médiocre (ME)	Triangle (1,2,4)
Moyenne (AV)	Triangle (3,5,7)
Excellente (EX)	Triangle (7,8,10)

Tableau 2.1: Paramètre d'entrée NM

Peu populaire (PP)	Triangle (1,20,40)
Modérément populaire (MP)	Triangle (20,40,60)
Très populaire (TP)	Triangle (40,60,80)

Tableau 2.2: Paramètre d'entrée PF

Faible (FL)	Triangle (0,0.2,0.4)
Moyenne (AV)	Triangle (0.2,0.4,0.6)
Élevée (EL)	Triangle (0.4,0.6,1)

Tableau 2.3: Paramètre d'entrée SP

À éviter (AE)	Triangle (0,20,40)
Option acceptable (OA)	Triangle (20,40,50)
Recommandé (RE)	Triangle (40,60,70)
À ne pas manquer (ANM)	Triangle (65,80,100)

Tableau 2.4: Paramètre de sortie RF

La matrice d'inférence est la suivante :

NM	ME			AV			EX		
SP\PF	PP	MP	TP	PP	MP	TP	PP	MP	TP
FL	AE	AE	AE	AE	OA	OA	RE	RE	RE
AV	AE	AE	OA	OA	OA	RE	RE	RE	ANM
EL	AE	OA	OA	OA	OA	RE	ANM	ANM	ANM

Tableau 2.5: La matrice d'inférence

2.3.1.3 Étapes de modélisation :

— **Ajouter les paramètres d'entrées flous :** On commence par ajouter nos parametres : NM, FP et SP en cliquant sur *Edit* > *Add variable* > *Input*

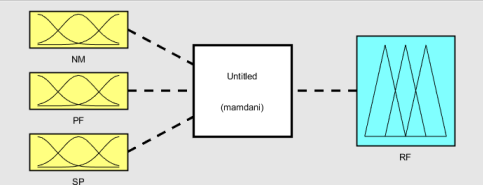


Figure 2.2: Ajout des paramètres d'entrées

— **Entrer les membership functions :** Pour chaque paramètre d'entrée et de sortie saisir les différentes fonctions d'appartenance.

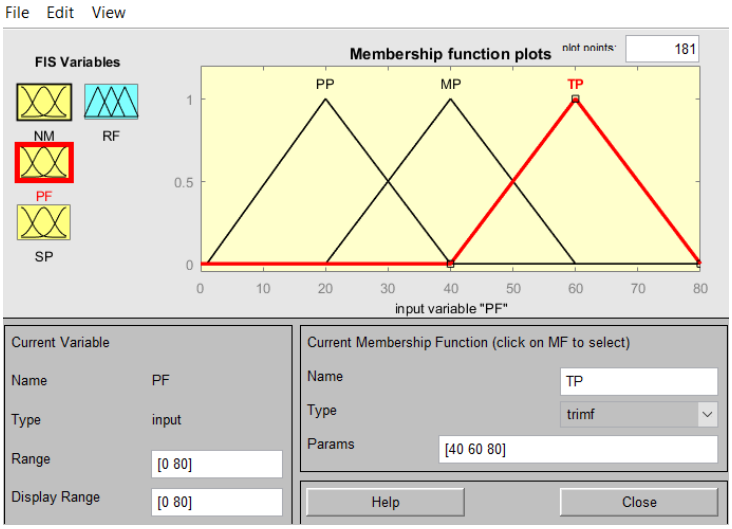


Figure 2.3: Ajout des fonctions d'appartenance

— **Saisir la base de règles :** Entrer notre base de règles en s'appuyant sur notre matrice d'inférence.

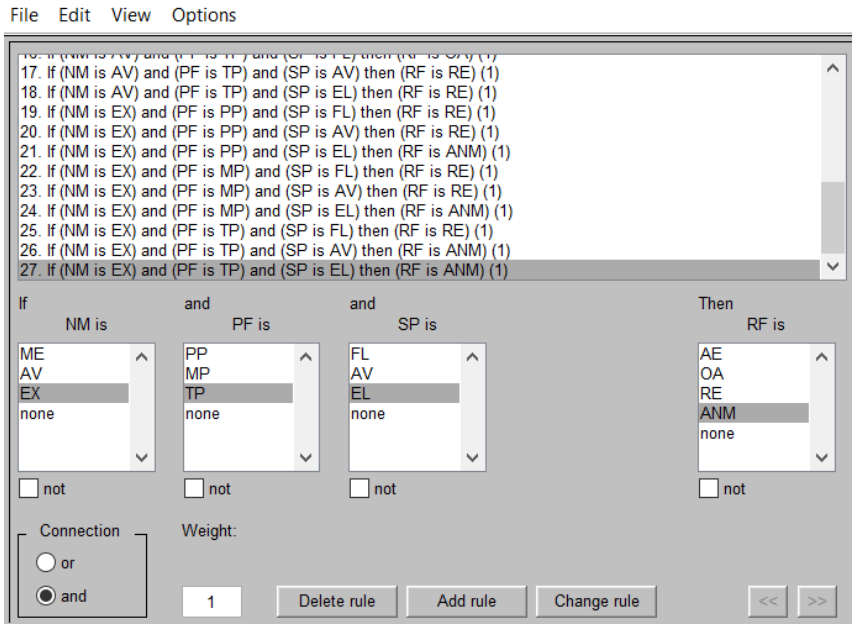


Figure 2.4: Saisi de la base de règles

— **Utiliser la fonction *evalfis* pour calculer les sorties flous :** On appel la fonction *evalfis* sur notre contrôleur et nos valeurs d'entrée flou pour obtenir nos sorties.

```
>> output = evalfis([8 25 0.53],TP2RCR)

output =

   72.5433

>> output = evalfis([1 44 0.3],TP2RCR)

output =

    50

>> output = evalfis([5 70 0.1],TP2RCR)

output =

   36.1111
```

Figure 2.5: Sorties flous

2.4 Conclusion

En conclusion, l'exploration de la conception d'un contrôleur flou avec la "Fuzzy Toolbox" de MATLAB met en lumière l'importance de la logique floue pour traiter des connaissances imprécises. La flexibilité démontrée par la "Fuzzy Toolbox" offre une approche novatrice pour modéliser des systèmes complexes, où la catégorisation nette est souvent limitée. Cette méthode, axée sur la théorie des ensembles flous, présente un potentiel significatif pour des applications plus adaptatives et robustes dans divers domaines de l'ingénierie.

Chapitre 3

TP3 : Réseaux causaux Bayésiens

3.1 Introduction

Les réseaux bayésiens causaux (RBC) constituent une approche fondamentale pour modéliser les relations de causalité au sein de systèmes complexes, offrant un cadre formel pour représenter et analyser les dépendances probabilistes entre différentes variables. Dans le cadre de ce travail pratique, notre objectif est d'explorer les boîtes à outils dédiées aux réseaux bayésiens : Bayes Net Toolbox (BNT) et le Probabilistic Graphical Models in Python (PGMPY). Nous débuterons par la construction de modèles causaux simples, évoluant progressivement vers des structures plus complexes, afin de comprendre la modélisation bayésienne et son application pour l'inférence probabiliste. Cette exploration permettra de mettre en lumière les capacités de ces outils dans la représentation et l'analyse de relations causales, tout en soulignant les défis associés à la modélisation de réseaux bayésiens de grande envergure.

3.2 Les réseaux causaux bayésiens

Les réseaux causaux bayésiens sont des modèles graphiques probabilistes qui servent à représenter et analyser les liens de cause à effet entre diverses variables. Fondés sur la théorie des probabilités bayésiennes, ces réseaux utilisent des graphes orientés sans cycle pour exprimer les dépendances causales entre les variables, offrant ainsi une représentation explicite des relations de cause à effet au sein d'un système.

3.3 Conception et implémentation des réseaux causaux bayésiens

3.3.1 En utilisant Bayes Net Toolbox (BNT) de Matlab

3.3.1.1 Présentation de l'outil BNT

BNT, la Bayes Net Toolbox, est un outil robuste pour la modélisation de réseaux bayésiens sur Matlab. Il propose diverses distributions de probabilité conditionnelle, notamment tabulaires, gaussiennes, softmax et perceptrons multi-couches. En plus des nœuds de probabilité, BNT prend en charge les nœuds de décision et d'utilité, permettant la création de diagrammes d'influence. Adapté aux réseaux bayésiens statiques et dynamiques, l'outil offre une large gamme d'algorithmes d'inférence exacte et approximative, ainsi que des méthodes pour l'apprentissage des paramètres et la régularisation. Grâce à sa documentation approfondie, son approche orientée objet, sa gratuité, BNT est idéal pour l'enseignement, la recherche et le prototypage rapide.

3.3.1.2 Etape 1 :

Considérons l'exemple suivant :

La reconnaissance vocale est une application très répondue dans le domaine de l'intelligence artificielle. Sa performance est influencée par plusieurs facteurs notamment la qualité de l'enregistrement audio, le modèle de langage utilisé et le traitement du signale audio. La reconnaissance vocale impact à son tour d'autres traitements tel que l'identification des personnes et la transcription.

Le texte précédant peut être modélisé par le réseau bayésien suivant :

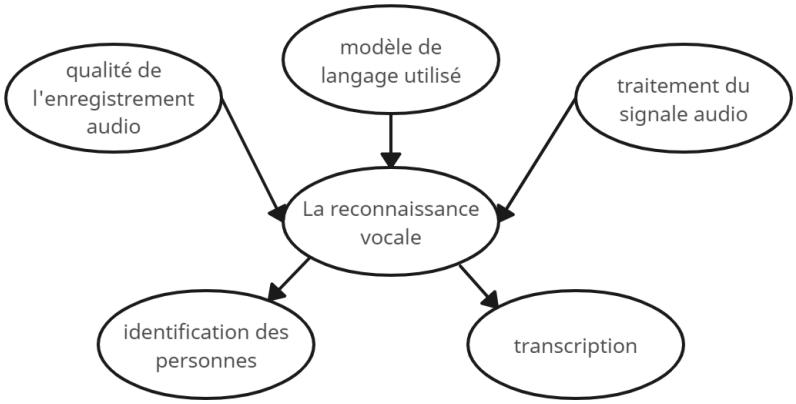


Figure 3.1: Représentation du réseau Polyarbre

1. **Génération du polyarbre :** En utilisant la notation suivante : QE : *"qualité de l'enregistrement audio"* , ML : *"modèle de langage utilisé"*, TS : *"traitement du signale audio"* , RV : *"La reconnaissance vocale"*, IP : *"identification des personnes"*, T : *"transcription"*. Nous générons le polyarbre en suivant les étapes suivantes :

```
>> N = 6;
QE = 1; ML = 2; TS = 3; RV = 4; IP = 5; T= 6;
dag = zeros(N,N);
dag(QE,RV)=1;
dag(ML,RV)=1;
dag(TS,RV)=1;
dag(RV,[IP T])=1;
discrete_nodes = 1:N;
node_sizes = 2*ones(1,N);
names = {'QE', 'ML', 'TS', 'RV', 'IP', 'T'};
bnet = mk_bnet(dag, node_sizes, 'names', names, 'discrete', 1:N);
```

Figure 3.2: Génération du polyarbre

2. **Les variables d'évidence et la variable d'intérêt :** Nous avons choisi les variables QE, ML et TS comme variables d'intérêt, la variable d'intérêt choisie est : T (La notation choisi est : 1 -> Vrai, 2-> Faux).

```
>> evidence = cell(1, N);
evidence{QE} = 1;
evidence{ML} = 2;
evidence{TS} = 1;
```

(a) Les variables d'évidence du polyarbre

```
>> interest = T;
interest_instance= 1;
```

(b) La variable d'intérêt du polyarbre

Figure 3.3: Les variables d'évidence et la variable d'intérêt du polyarbre

3. **Les distributions a priori pour les nœuds racine :** Les distributions pour chaque noeud sont de la forme [P(A=V) P(A=F)].

```
>> bnet.CPD{QE} = tabular_CPD(bnet, QE, [0.7 0.3]);
bnet.CPD{ML} = tabular_CPD(bnet, ML, [0.5 0.5]);
bnet.CPD{TS} = tabular CPD(bnet, TS, [0.6 0.4]);
```

Figure 3.4: Les distributions a priori pour les nœuds racine du polyarbre

4. **Les distributions conditionnelles pour les autres nœuds :** Les distributions pour les noeud IP et T sont de la forme [P(A=V|B) P(A=F|B)] et pour le noeud RV de la forme [P(A=V|B, C, D) P(A=F|B, C, D)]

```
>> bnet.CPD{RV} = tabular_CPD(bnet, RV, [0.1 0.9
                                         0.2 0.8
                                         0.3 0.7
                                         0.4 0.6
                                         0.5 0.5
                                         0.6 0.4
                                         0.7 0.3
                                         0.8 0.2]);
bnet.CPD{IP} = tabular_CPD(bnet, IP, [0.2 0.8
                                      0.4 0.6]);
bnet.CPD{T} = tabular_CPD(bnet, T, [0.3 0.7
                                    0.5 0.5] );
```

Figure 3.5: Les distributions conditionnelles pour les autres nœuds du polyarbre

5. **Calcule de P(variable d'intérêt | évidences) :**

```
>> engine = jtree_inf_engine(bnet);
engine = enter_evidence(engine, evidence);

marg_interest = marginal_nodes(engine, interest);
prob_interest_given_evidence = marg_interest.T(interest_instance);
disp(['Probabilité de la variable T (', names{interest}, '=', num2str(interest_instance), ') given evide
Probabilité de la variable T (T=1) given evidence: 0.44
```

Figure 3.6: Calcule de p(variable d'intérêt | évidences) du polyarbre

Nous pouvons afficher la distribution pour le noeud variable d'intérêt. En utilisant la commande suivante : BAR(MARG_INTEREST.T)

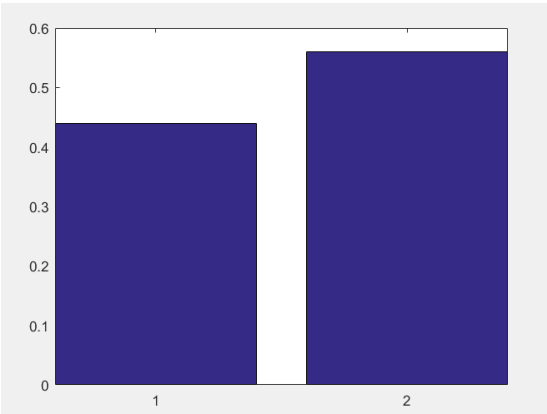


Figure 3.7: La distribution marginale de la variable T

6. **Afficher le graphe :** Nous pouvons afficher le graphe en utilisant la commande suivante G= BNET.DAG DRAW_GRAPH(G) ;

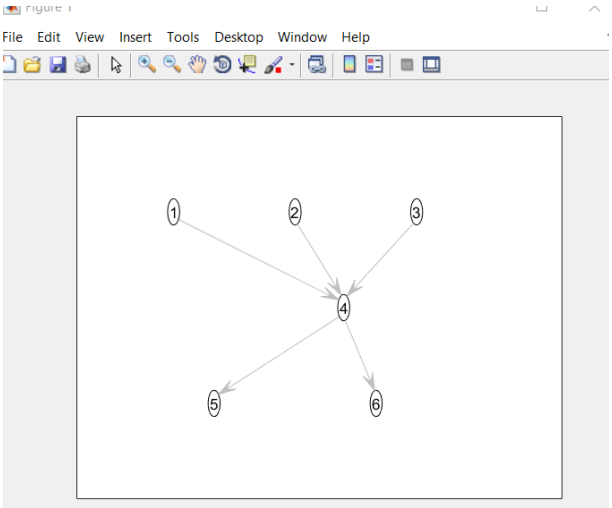


Figure 3.8: Affichage du polyarbre

3.3.1.3 Etape 2 :

Considérons l'exemple suivant :

Les problèmes de classification sont des thématiques très répandues dans le domaine de l'intelligence artificielle. Le type d'algorithmes choisi et l'affinement des paramètres d'entrée influent fortement sur la performance de l'algorithme de classification. Cette dernière, associée au prétraitement du dataset et à la distribution des exemples des différentes classes, détermine la précision du classificateur. Il est aussi important de préciser que l'approche utilisée lors du prétraitement peut affecter la distribution des exemples des classes.

Le texte précédent peut être modélisé par le réseau bayésien suivant :

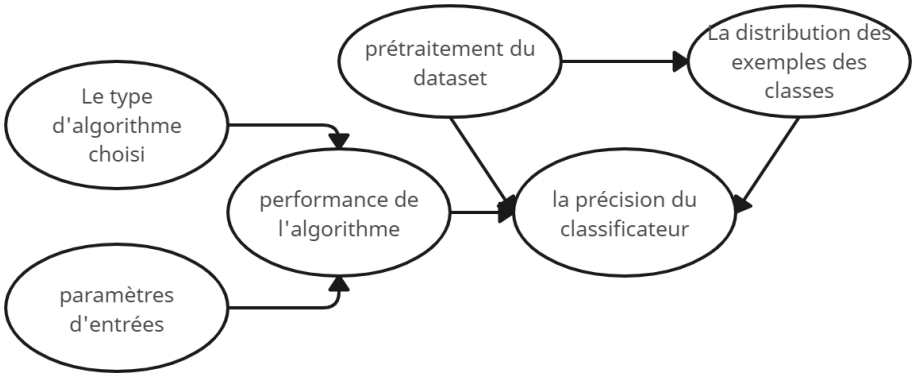


Figure 3.9: Représentation du graphe à connexions multiples

1. **Génération du graphe à connexions multiples :** En utilisant la notation suivante : AC : *"algorithme choisi"* , AP : *"paramètres d'entrées"*, PA : *"performance de l'algorithme de classification"* , PRE : *prétraitement du dataset"*, DC : *"distribution des exemples des différentes classes"*, PC : *"précision du classificateur"*. Nous générons le graphe à connexions multiples en suivant les étapes suivantes :

```
>> N = 6;
AC =1; AP=2; PA=3; PRE=4; DC=5; PC=6;
dag = zeros(N,N);
dag(AC , PA) = 1;
dag(AP, PA) =1;
dag(PA, PC) = 1;
dag(PRE, PC) =1;
dag(PRE, DC) =1;
dag(DC,PC) =1;
discrete_nodes = 1:N;
node_sizes = 2*ones(1,N);
>> names = {'AC', 'AP', 'PA', 'PRE', 'DC', 'PC'};
bnet = mk_bnet(dag, node_sizes, 'names', names, 'discrete', 1:N);
```

Figure 3.10: Génération du graphe à connexions multiples

2. **Les variables d'évidence et la variable d'intérêt :** Nous avons choisi les variables AC et AP comme variables d'intérêt, la variable d'intérêt choisie est : PC.

```
>> evidence = cell(1, N);
evidence{AC} = 2;
evidence{AP} = 1;
```

(a) Les variables d'évidence du graphe à connexions multiples

```
>> interest = PC;
interest_instance= 1;
```

(b) La variable d'intérêt du graphe à connexions multiples

Figure 3.11: Les variables d'évidence et la variable d'intérêt du graphe à connexions multiples

3. **Les distributions a priori pour les nœuds racine :** Les distributions pour chaque noeud sont de la forme [P(A=V) P(A=F)].

```
>> bnet.CPD{AC} = tabular_CPD(bnet, AC, [0.5 0.5]);
bnet.CPD{AP} = tabular_CPD(bnet, AP, [0.6 0.4]);
bnet.CPD{PRE} = tabular_CPD(bnet, PRE, [0.7 0.3]);
```

Figure 3.12: Les distributions a priori pour les nœuds racine du graphe à connexions multiples

4. **Les distributions conditionnelles pour les autres nœuds :** Les distributions pour le noeud DC sont de la forme [P(A=V|B) P(A=F|B)], pour le noeud PA de la forme [P(A=V|B, C) P(A=F|B, C)] et pour le noeud PC de la forme [P(A=V|B,C, D) P(A=F|B, C, D)].

```
>> bnet.CPD{AC} = tabular_CPD(bnet, AC, [0.5 0.5]);
bnet.CPD{AP} = tabular_CPD(bnet, AP, [0.6 0.4]);
bnet.CPD{PRE} = tabular_CPD(bnet, PRE, [0.7 0.3]);

bnet.CPD{PC} = tabular_CPD(bnet, PC, [0.1 0.9
                                      0.15 0.85
                                      0.4 0.6
                                      0.45 0.55
                                      0.6 0.4
                                      0.78 0.22
                                      0.8 0.2
                                      0.2 0.8]);
bnet.CPD{DC} = tabular_CPD(bnet, DC, [0.6 0.4
                                      0.35 0.65]);
bnet.CPD{PA} = tabular_CPD(bnet, PA, [0.9 0.1
                                      0.7 0.3
                                      0.88 0.12
                                      0.05 0.95]);
```

Figure 3.13: Les distributions conditionnelles pour les autres nœuds du graphe à connexions multiples

5. **Calcule de P(variable d'intérêt | évidences) :**

```
>> engine = jtree_inf_engine(bnet);
engine = enter_evidence(engine, evidence);
marg_interest = marginal_nodes(engine, interest);
prob_interest_given_evidence = marg_interest.T(interest_instance);
disp(['Probabilité de la variable PC (', names{interest}, '=', num2str(interest_instance), ') given
Probabilité de la variable PC (PC=1) given evidence: 0.3959
```

Figure 3.14: Calcule de p(variable d'intérêt | évidences) du graphe à connexions multiples

Nous pouvons afficher la distribution pour le noeud variable d'intérêt. En utilisant la commande suivante : BAR(MARG_INTEREST.T)

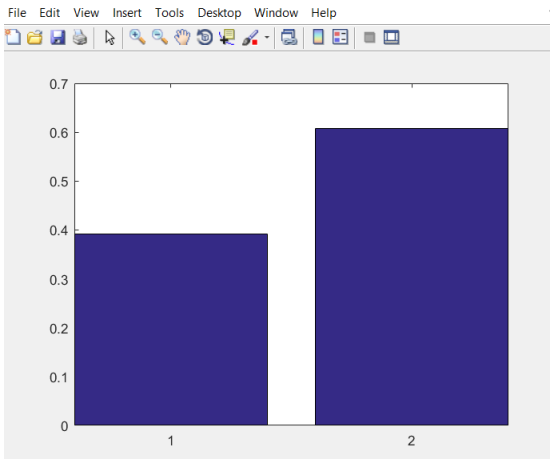


Figure 3.15: La distribution marginale de la variable PC

6. **Afficher le graphe :** Nous pouvons afficher le graphe en utilisant la commande suivante G= BNET.DAG DRAW_GRAPH(G) ;

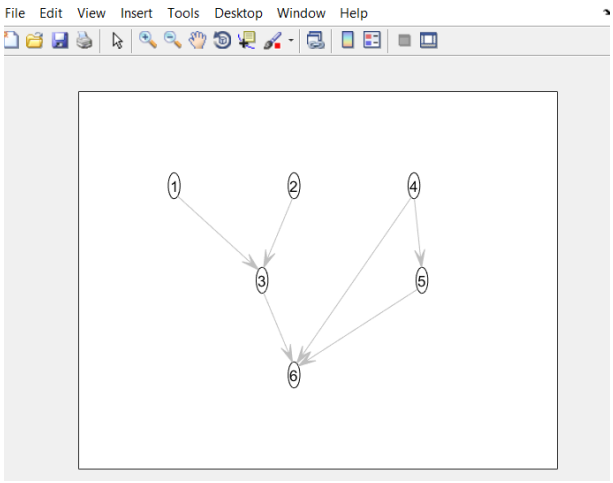


Figure 3.16: Affichage du graphe à connexions multiples

3.3.1.4 Etape 3 :

Génération d'un graphe à connexions multiples tels que le nombre de variables et le nombre de parents max sont très grands. Pour créer ce graphe nous avons suivi les étapes suivantes :

1. **Générer la structure du graphe :** Nous avons tout d'abord fixer le nombre de noeuds du graphe et le nombre maximal de parents, puis nous avons généré aléatoirement la matrice d'adjacence *dag*

```
>> num_variables = 50;
>> max_parents = 5;
>> dag = zeros(num_variables, num_variables);
for i = 1:num_variables
    num_parents = min(randi([1, max_parents]), i-1);
    if num_parents > 0
        parents = randperm(i-1, num_parents);
        dag(i, parents) = 1;
    end
end
```

Figure 3.17: Génération de la structure du graphe

2. **Générer le réseau bayésien :** Créer le réseau bayésien en précisant le nombre de variables discrètes, chacune étant binaire.

```
>> node_sizes = 2 * ones(1, num_variables);
names = cell(1, num_variables);
for i = 1:num_variables
    names{i} = ['Var' num2str(i)];
end
>> bnet = mk_bnet(dag, node_sizes, 'names', names, 'discrete', 1:num_variables);
```

Figure 3.18: Génération du réseau bayésien

3. **Attribuer les distributions des probabilités :** Attribuer a chaque noeud une distribution aléatoire selon le nombre de noeuds parents.

```
for i = 1:num_variables
    parents = find(dag(:, i));
    num_combinations = prod(node_sizes(parents));

    CPT = rand(num_combinations, node_sizes(i)-1);
    CPT = [CPT, 1 - sum(CPT, 2)];

    bnet.CPD{i} = tabular_CPD(bnet, i, 'CPT', CPT);
end
```

Figure 3.19: Génération des distributions des probabilités

4. **Afficher les distributions :**

```
>> CPTs = cell(1, num_variables);
for i = 1:num_variables
    s = struct(bnet.CPD{i});
    CPTs{i} = s.CPT;
end
for i = 1:num_variables
    disp(['Distribution de probabilité pour le nœud ' names{i} ':']);
    disp(CPTs{i});
end
```

(a) Code

```
Distribution de probabilité pour le nœud Var45:
    0.7314    0.2686
    0.3199    0.6801

Distribution de probabilité pour le nœud Var46:
    0.7305
    0.2695

Distribution de probabilité pour le nœud Var47:
    0.2095
    0.7905
```

(b) Résultats

Figure 3.20: Affichage des distributions

5. **Afficher le graphe :** En utilisant la commande suivante G= BNET.DAG DRAW_GRAPH(G) ;

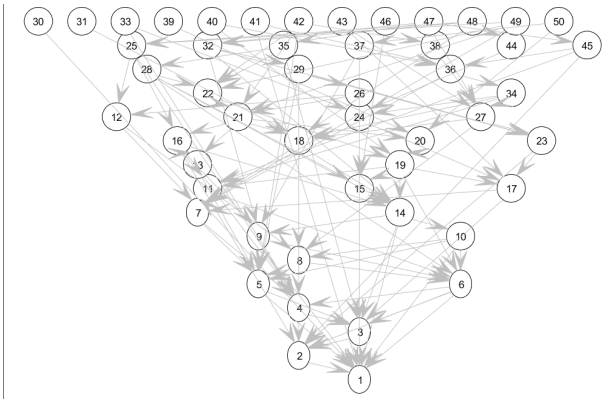


Figure 3.21: Affichage du graphe

3.3.2 En utilisant Probabilistic Graphical Models de Python

3.3.2.1 Présentation de l'outil PGMPY

PGMPy est une bibliothèque open-source en Python spécialisée dans la modélisation, l'inférence et l'apprentissage des modèles graphiques probabilistes, tels que les réseaux bayésiens et les champs aléatoires de Markov. Elle propose une interface conviviale pour la création et la manipulation de ces modèles, ainsi que des fonctionnalités d'inférence probabiliste pour estimer des valeurs manquantes ou prédire de nouvelles observations. En outre, la bibliothèque prend en charge des opérations d'apprentissage, permettant d'ajuster les paramètres du modèle à partir de données observées, ce qui facilite l'analyse des relations complexes entre variables aléatoires.

3.3.2.2 Etape 1 :

Considérons le même exemple utilise pour cette étape avec l'outil BNT.

1. **Génération du polyarbre :** En utilisant la notation suivante : QE : *"qualité de l'enregistrement audio"* , ML : *"modèle de langage utilisé"*, TS : *"traitement du signale audio"* , RV : *"La reconnaissance vocale"*, IP : *"identification des personnes"*, T : *"transcription"*.

```
polyarbre_model = BayesianNetwork([
('QE', 'RV'),
('ML', 'RV'),
('TS', 'RV'),
('RV', 'IP'),
('RV', 'T'),])
```

2. **Les variables d'évidence et la variable d'intérêt :** Tout comme pour le premier outils nous avons choisi les variables QE, ML et TS comme variables d'intérêt, la variable d'intérêt choisie est : T.

```
evidence={'QE': 0, 'ML': 1, 'TS': 0}
var_interest=['T']
```

3. **Les distributions a priori pour les nœuds racine :**

```
QE_cpd = TabularCPD( variable='QE', variable_card=2, values=[[
                                    0.7], [0.3]])
ML_cpd = TabularCPD( variable='ML', variable_card=2, values=[[
                                    0.5], [0.5]])
TS_cpd = TabularCPD( variable='TS', variable_card=2, values=[[
                                    0.6],[0.4]],)
```

4. **Les distributions conditionnelles pour les autres nœuds :**

```
RV_cpd = TabularCPD( variable='RV', variable_card=2,
values=[[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8],
        [0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2]],
evidence=['QE', 'ML', 'TS'], evidence_card=[2, 2, 2])
IP_cpd = TabularCPD( variable='IP', variable_card=2,
    values=[[0.2,0.4],[0.8,0.6]],
    evidence=['RV'],evidence_card=[2])
T_cpd = TabularCPD( variable='T', variable_card=2,
    values=[[0.3,0.5],[0.7,0.5]],
    evidence=['RV'], evidence_card=[2])
polyarbre_model.add_cpds( QE_cpd, ML_cpd, TS_cpd, RV_cpd,
                                    IP_cpd,T_cpd)
```

5. **Calcule de P(variable d'intérêt | évidences) :**

```
from pgmpy.inference import VariableElimination
livraison_inference = VariableElimination(polyarbre_model)
print("P(T|QE=0, ML=1, TS=0)")
print(livraison_inference.query(variables=var_interest,
                                evidence=evidence))
```

Resultat :

```
P(T|QE=0, ML=1, TS=0)
+------+----------+
| T    |   phi(T) |
+======+==========+
| T(0) |   0.4400 |
+------+----------+
| T(1) |   0.5600 |
+------+----------+
```

Figure 3.22: Resultat P(variable d'intérêt | évidences)

6. **Afficher le graphe :**

```
import networkx as nx
import matplotlib.pyplot as plt
poly_graph = nx.DiGraph(polyarbre_model.edges())
pos = nx.spring_layout(poly_graph)
node_labels = {node: node for node in polyarbre_model.nodes()}
plt.figure(figsize=(3, 3))
nx.draw(poly_graph, pos, with_labels=True, labels=node_labels,
                                    node_size=1000,
                                    node_color='green',
```

```
                                                font_size=10, font_color='
                                                black', arrowsize=20)
    plt.show()
```

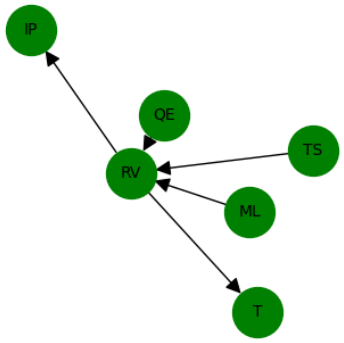


Figure 3.23: Graphe polyarbre

3.3.2.3 Etape 2 :

Considérons le même exemple utilise pour cette étape avec l'outil BNT.

1. **Génération du graphe à connexions multiples :** En utilisant la notation suivante : AC : *"algorithme choisi"* , AP : *"paramètres d'entrées"*, PA : *"performance de l'algorithme de classification"* , PRE : *prétraitement du dataset"*, DC : *"distribution des exemples des différentes classes"*, PC : *"précision du classificateur"*.

```
    multiconect_model = BayesianNetwork([
    ('AC', 'PA'),
    ('AP', 'PA'),
    ('PA', 'PC'),
    ('PRE', 'DC'),
    ('PRE', 'PC'),
    ('DC', 'PC'),])
```

2. **Les variables d'évidence et la variable d'intérêt :**

```
    evidence_multi={'AC': 1, 'AP': 0}
    var_interest_multi=['PC']
```

3. **Les distributions a priori pour les nœuds racine :**

```
    AC_cpd = TabularCPD(variable='AC',variable_card=2,values=[[0.5
                                                ], [0.5]])
    AP_cpd = TabularCPD(variable='AP', variable_card=2,values=[[0.
                                                6], [0.4]])
```

```
PRE_cpd = TabularCPD(variable='PRE', variable_card=2, values=[
                                        [0.7],[0.3]],)
```

4. **Les distributions conditionnelles pour les autres nœuds :**

```
DC_cpd = TabularCPD( variable='DC', variable_card=2,
values=[[0.6,0.35],
        [0.4,0.65]],
evidence=['PRE'],evidence_card=[2])
PA_cpd = TabularCPD(
variable='PA',variable_card=2,
values=[[0.9,0.7,0.88,0.05],
        [0.1,0.3,0.12,0.95]],
evidence=['AC','AP'],evidence_card=[2,2])
PC_cpd = TabularCPD(
variable='PC', variable_card=2,
values=[[0.1, 0.15, 0.4, 0.45, 0.6, 0.78, 0.8, 0.2],
        [0.9, 0.85, 0.6, 0.55, 0.4, 0.22, 0.2, 0.8]],
evidence=['PRE', 'DC', 'PA'],evidence_card=[2, 2, 2])
```

5. **Calcule de P(variable d'intérêt | évidences) :**

```
from pgmpy.inference import VariableElimination
livraison_inference = VariableElimination(multiconect_model)
print("P(PC|AC=1, AP=0)")
print(livraison_inference.query(variables=var_interest,
                                evidence=evidence))
```

Resultat :

```
P(PC|AC=1, AP=0)
+-------+-----------+
| PC    |   phi(PC) |
+=======+===========+
| PC(0) |    0.3654 |
+-------+-----------+
| PC(1) |    0.6346 |
+-------+-----------+
```

Figure 3.24: Resultat P(variable d'intérêt | évidences)

6. **Afficher le graphe :**

```
import networkx as nx
import matplotlib.pyplot as plt
multi_graph = nx.DiGraph(multiconect_model.edges())
pos = nx.spring_layout(multi_graph)
```

```
    node_labels = {node: node for node in multiconect_model.nodes
                                        ()}
    plt.figure(figsize=(3, 3))
    nx.draw(multi_graph, pos, with_labels=True, labels=node_labels
                                        , node_size=1000,
                                        node_color='green',
                                        font_size=10, font_color='
                                        black', arrowsize=20)
    plt.show()
```

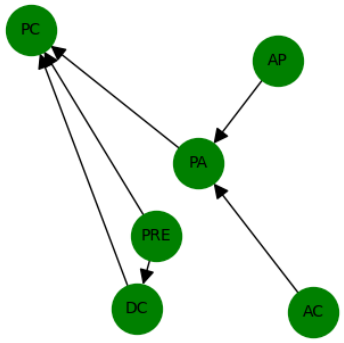


Figure 3.25: Graphe muticonnected

3.3.2.4 Etape 3 :

Génération d'un graphe à connexions multiples tels que le nombre de variables et le nombre de parents max sont très grands. Pour créer ce graphe nous avons utilise le code suivant :

```
import numpy as np
import networkx as nx
from pgmpy.models import BayesianNetwork
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination
import matplotlib.pyplot as plt

num_variables = 50
max_parents = 5

dag = np.zeros((num_variables, num_variables))
for i in range(num_variables):
    num_parents = min(np.random.randint(1, max_parents + 1), i)
    if num_parents > 0:
        parents = np.random.choice(np.arange(i), num_parents, replace=
                                            False)
```

```
        dag[i, parents] = 1

G = nx.DiGraph(dag)

pos = nx.spring_layout(G, seed=42)
nx.draw(G, pos, with_labels=True, node_color='g', font_weight='bold')
plt.show()

node_sizes = 2 * np.ones(num_variables, dtype=int)
names = [f'Var{i}' for i in range(1, num_variables + 1)]

bnet = BayesianNetwork()

for i in range(num_variables):
    parents = np.where(dag[:, i] == 1)[0]
    num_combinations = np.prod(node_sizes[parents])

    CPT = np.random.rand(num_combinations, node_sizes[i] - 1)
    CPT = np.hstack((CPT, 1 - np.sum(CPT, axis=1, keepdims=True)))

    CPT = CPT.T

    bnet.add_node(f'Var{i+1}')

    bnet.add_edges_from([(f'Var{j+1}', f'Var{i+1}') for j in parents])

    cpd = TabularCPD(variable=f'Var{i+1}', variable_card=node_sizes[i],
                     evidence=[f'Var{j+1}' for j in parents],
                     evidence_card=node_sizes[parents],
                     values=CPT)

    bnet.add_cpds(cpd)
np.set_printoptions(precision=3, suppress=True)

CPTs = []
for i in range(num_variables):
    cpd_values = bnet.get_cpds(f'Var{i+1}').values
    CPTs.append(cpd_values)

for i, cpt in enumerate(CPTs, start=1):
    print(f'CPD for Var{i}:\n', cpt)
```

Afficher le graphe :

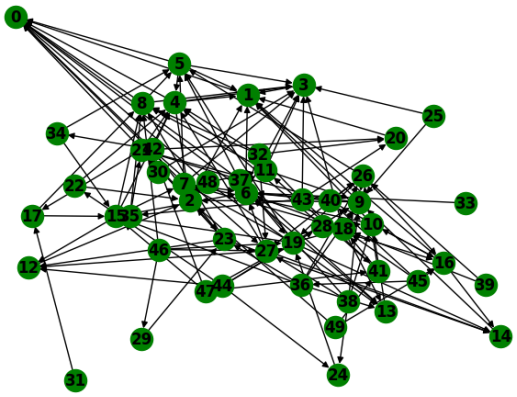


Figure 3.26: Graphe généré aléatoirement

Afficher les distributions conditionnelles :

```
CPD for Var40:
 [0.543 0.457]
CPD for Var41:
 [[0.066 0.496]
 [0.934 0.504]]
CPD for Var42:
 [[[[0.273 0.685]
   [0.763 0.618]]

  [[0.944 0.603]
   [0.413 0.619]]]


 [[[0.727 0.315]
   [0.237 0.382]]

  [[0.056 0.397]
   [0.587 0.381]]]]
```

Figure 3.27: Distributions conditionnelles générées aléatoirements

3.4 Conclusion

Cette exploration des Réseaux Bayésiens Causaux à travers les outils BNT et PGMPY a révélé leur efficacité pour modéliser des relations causales, offrant une compréhension approfondie de la modélisation bayésienne et de son application à l'inférence probabiliste. Toutefois, la gestion des défis liés à la modélisation de réseaux bayésiens complexes reste une considération cruciale pour une utilisation étendue.

Chapitre 4

TP4 : Logique possibiliste qualitative

4.1 Introduction

L'objectif principal de ce TP est triple : premièrement, implémenter l'algorithme à l'aide d'un solveur SAT afin de manipuler des formules logiques complexes. Deuxièmement, générer des bases de connaissances pondérées ainsi que des variables d'intérêt qui sont cruciales pour représenter les incertitudes dans le raisonnement logique. Enfin, troisièmement, calculer Π(variable d'intérêt) en utilisant l'algorithme spécifié dans le cadre de la logique possibiliste qualitative.

4.2 La logique possibiliste qualitative

La logique possibiliste qualitative est une approche de la modélisation logique qui permet de traiter l'incertitude de manière qualitative. Elle s'attache à représenter les degrés de croyance ou de possibilité dans la vérité des propositions plutôt que de fournir des réponses binaires. Cette logique offre un cadre permettant de raisonner sur des situations où l'information est imprécise, en attribuant des degrés de crédibilité aux différentes hypothèses plutôt que de se limiter à des certitudes catégoriques.

4.3 Exploration de l'outil

4.3.1 Génération de la base de connaissance

Pour mettre en œuvre notre algorithme, il est nécessaire de commencer par établir une base de connaissances. Cela se fait à travers un processus algorithmique (voir 4.3.1 qui

génère de manière aléatoire toutes les informations requises pour notre base de connaissances, y compris une évidence.

```
    def getPreProcessedFormulas(self, subFormulas):
        dictCor = {}
        returnedFormulas = []
        cpt = 1
        i = 0
        for form in subFormulas:
            j = 0
            returnedFormulas.insert(i, [])
            for pred in form:
                if pred not in dictCor.keys():
                    if int(pred) > 0:
                        dictCor[int(pred)] = cpt
                        dictCor[-int(pred)] = -cpt
                        cpt += 1
                    else:
                        dictCor[-int(pred)] = cpt
                        dictCor[int(pred)] = -cpt
                        cpt += 1
                returnedFormulas[i].insert(j, dictCor[pred])
                j += 1
            i += 1
        return returnedFormulas
```

4.3.2 Implémentation de l'algorithme d'inférence

```
    while baseConnaissance.lower < baseConnaissance.upper:
    print("iteration = ", iteration)
    #on calcul r
    r = int((baseConnaissance.lower + baseConnaissance.upper + 1) / 2)
    # Detect all formulas of r strate
    liste = baseConnaissance.getWeights()
    valueR = -1

    # On d tecte les deux indices des formules associ  aux strates r
                                        et u.
    for i in range(len(liste)):
        if baseConnaissance.getStratesWeights()[r - 1] > liste[i]:
            valueR = i
            break
    if valueR == -1:
        valueR = len(liste) - 1
```

```
    # Detect formulas of u strate
    for j in range(len(liste)):
        if baseConnaissance.getStratesWeights()[baseConnaissance.upper -
                                                1] == liste[j]:
            valueU = j
            break
    #cnf contient les formules situ e entre les strates u et r pour
                                        optimiser l'algo de recherche
    cnf = baseConnaissance.formulas[valueU:valueR]

    # Generate negation of evidence absurde
    for i in range(len(gen_base)):
        literal = -1 * gen_base[i]
        cnf.append([literal])

    cnf = baseConnaissance.getPreProcessedFormulas(cnf)
    print("cnf =  ", cnf)



    result = pycosat.solve(cnf) #tester la consistence
    # Si le r sultat est consistant alors on met    jours upper    r-1
                                        sinon on met lower    r.
    if type(result) == type([]):  # Consistence Test : pycosat return
                                        liste if result is satisfiable
                                        else return string "UNSAT" or "
                                        UNKNOWN"
        baseConnaissance.upper = r - 1
        print("upper u : ", baseConnaissance.upper)
    else:
        baseConnaissance.lower = r
        print("lower l : ", baseConnaissance.lower)
    iteration = iteration + 1
    print("")

#la valeur associ e    la strate correspondant    la borne sup rieure
Val = baseConnaissance.getStratesWeights()[r - 1]
print("Val(" + str(gen_base) + ",BC) = " + str(Val))
```

1. **Initialisation des Paramètres :**
   - `nClause` : Le nombre de clauses dans chaque formule.
   - `nVariable` : Le nombre de variables dans chaque clause.
   - `length` : La longueur des clauses générées.

- `maxNumber` : La valeur maximale possible pour une variable.

2. **Génération de la Base de Connaissance**
3. **Traitement de la Base de Connaissance :**
   - `Process` est classe qui prend en charge le traitement de la base de connaissance.
   - `baseConnaissance.evidence = gen_base` : Définit l'évidence de la base de connaissance à partir de ce qui a été généré précédemment.
   - `baseConnaissance.sortedWeight()` : Trie les poids des formules dans la base de connaissance.
   - `baseConnaissance.calculateStrates()` : Calcule les strates (plages de poids) dans la base de connaissance.
4. **Boucle d'Itération :**
   - La boucle `while` s'exécute tant que la borne inférieure de la strate est inférieure à la borne supérieure.
   - `r = int((baseConnaissance.lower + baseConnaissance.upper + 1) / 2)` : Calcule le milieu entre la borne inférieure et la borne supérieure.
   - La boucle recherche des formules dans des strates spécifiques (`u` et `r`) pour optimiser l'algorithme de recherche.
5. **Test de Consistance :**
   - `result = pycosat.solve(cnf)` : Utilise le solveur `pycosat` pour tester la consistence des formules générées.
   - Si le résultat est une liste, cela signifie que les formules sont satisfiables, et la borne supérieure est mise à jour, sinon la borne inférieure est mise à jour.
6. **Affichage des Résultats :**
   - Le code affiche les itérations, les valeurs de la borne supérieure et inférieure, ainsi que la valeur associée à la strate correspondant à la borne supérieure.

4.4 Résultat

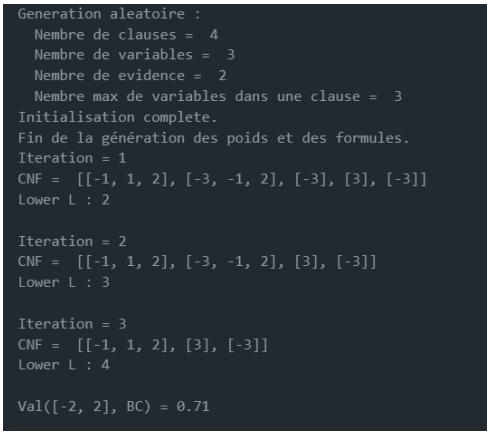


Figure 4.1: Résultat de l'exécution de l'algorithme d'inférence

À chaque itération, la CNF générée est adaptée en fonction de la consistence des formules. La valeur calculée pour la variable d'intérêt [-2, 2] dans la base de connaissances (BC) est 0.71.
Cette valeur peut être interprétée comme un degré de croyance associé à la variable d'intérêt, indiquant dans quelle mesure cette variable est considérée comme plausible ou probable compte tenu des connaissances logiques et des incertitudes intégrées dans le raisonnement.

A travers la FIGURE 4.2, il est clair que tout au long des itérations, la borne inférieure est ajustée jusqu'à ce qu'elle concorde avec la borne supérieure. **En résumé**
L'algorithme vise à évaluer les degrés de croyance associés à des variables d'intérêt en tenant compte de l'incertitude dans les connaissances logiques.
Les itérations successives ajustent la borne inférieure en fonction de la consistence des formules générées, ce qui reflète une démarche d'adaptation aux incertitudes dans le raisonnement logique.
La valeur calculée pour la variable d'intérêt représente un indice de confiance, indiquant la force de la croyance associée à cette variable.

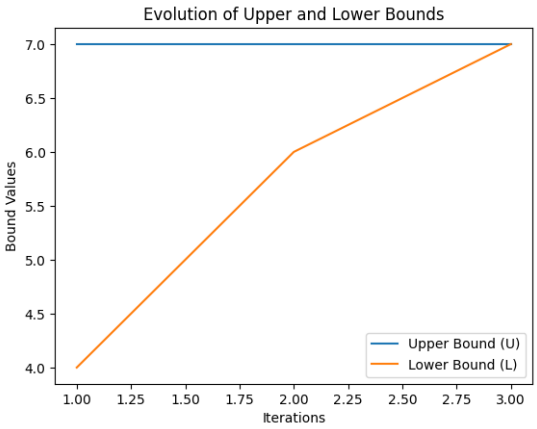


Figure 4.2: Graphique d'évolution des bornes inférieure et supérieure a travers les itérations

4.5 Conclusion

Ce TP nous a permis d'explorer la génération aléatoire de bases de connaissances, en utilisant un algorithme de recherche binaire pour résoudre des problèmes de satisfaction de contraintes. De plus, il nous a plongé dans la déduction en logique possibiliste qualitative, calculant un degré d'incohérence. L'implémentation de l'algorithme a souligné l'importance de la méthodologie et de l'approche adoptée, renforçant ainsi la compréhension des mécanismes de la logique possibiliste pour évaluer l'incertitude et l'incohérence dans les bases de connaissances générées.

Chapitre 5

Inférence logique et propagation graphique en théorie des possibilités

5.1 Introduction

Dans ce TP final, nous allons mettre en œuvre l'inférence et la propagation graphique dans la théorie des possibilités en utilisant l'outil PNT. Pour cela, nous allons utiliser Cygwin 32bit, un émulateur Unix pour Windows, afin d'exécuter les programmes passage.exe et inference.exe. Une fois l'outil en cours d'exécution, nous allons évaluer les temps d'exécution variés de la propagation et de l'inférence sur des réseaux de possibilités correspondant à des polyarbres, des graphes faiblement connectés, des graphes moyennement connectés et des graphes fortement connectés.

5.2 Théorie des possibilités et propagation graphique

L'inférence logique et la propagation graphique en théorie des possibilités sont des concepts clés dans la modélisation de l'incertitude. L'inférence logique s'appuie sur la logique floue pour tirer des conclusions à partir d'informations incertaines, tandis que la propagation graphique analyse la manière dont l'incertitude se propage à travers un réseau de relations représenté sous forme de graphe. Ensemble, ces concepts offrent une approche puissante pour représenter, raisonner et propager l'incertitude de manière plus avancée, jouant un rôle crucial dans des domaines tels que l'intelligence artificielle et la prise de décision sous incertitude.

5.3 Installation

5.3.1 Sous Windows

1. Création du répertoire anhla avec les fichiers Matlab a utiliser et l'outil PNT.
2. Execution du fichier **add_pnt_to_path_possmin** sous matlab d'inférence.

5.3.2 Sous Cygwin

1. Execution du fichier **prop1evid.m** (pour une évidence) **prop2evid.m** (pour deuxévidences)
2. Execution de **passage.exe** qui associe au graphe de possibilité basé sur le produit, la base de connaissance possibilité quantitative correspondante.
3. Execution de **inference.exe** afin de lancer processus d'inférence qui consiste à calculer le degré de possibilité de 'instance de la variable d'intérêt ainsi que le temps

5.4 Expérimentation

1. Polyarbres :
   - Nombre de nœuds : Choisir un nombre élevé, par exemple, `nb_nodes = 15`.
   - Nombre maximal de parents : Choisir un faible nombre, par exemple, `nb_parent_max = 2`.
2. Graphes faiblement connectés :
   - Nombre de nœuds : Choisir un nombre moyen, par exemple, `nb_nodes = 20`.
   - Nombre maximal de parents : Choisir un faible nombre, par exemple, `nb_parent_max = 2`.
3. Graphes moyennement connectés :
   - Nombre de nœuds : Choisir un nombre moyen à élevé, par exemple, `nb_nodes = 25`.
   - Nombre maximal de parents : Choisir un nombre moyen, par exemple, `nb_parent_max = 3`.
4. Graphes fortement connectés :
   - Nombre de nœuds : Choisir un nombre élevé, par exemple, `nb_nodes = 30`.
   - Nombre maximal de parents : Choisir un nombre élevé, par exemple, `nb_parent_max = 4`.

5.4.1 Une seule évidence

Graphe	Nbr de noeud	nbr max parent	Temps d'exécution d'inference (ms)	Temps de propagation (S)	possibilite conditionnelle de l'interet | evidences
Polyarbre	15	2	87944	0	0
Faiblement connectés	20	3	112937	0.027459	1
Myennement connecté	25	4	128987	0.020417	1
Fortement connecté	30	5	111857	0.033943	1

Tableau 5.1: Tbleau des expérimentations avec une seul évidence

5.4.1.1 Interprétation

- Pour le Polyarbre, le temps d'exécution est très faible (87,944 ms) avec un temps de propagation nul, indiquant probablement un modèle simple sans nécessité de propagation.
- Les graphes faiblement, moyennement et fortement connectés ont des temps d'exécution plus élevés, ce qui peut être dû à la complexité des relations entre les nœuds.
- La possibilité conditionnelle de l'intérêt, étant 1 pour les graphes connectés, indique que l'évidence fournie influence effectivement le nœud d'intérêt dans ces cas.

5.4.2 Deux évidence

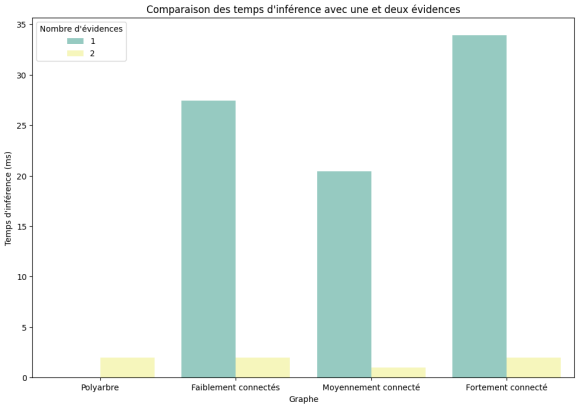
Graphe	Nbr de noeud	nbr max parent	Temps d'exécution d'inference (ms)	Temps de propagation (S)	possibilite conditionnelle de l'interet | evidences
Polyarbre	15	2	99655	2	5
Faiblement connectés	20	3	97533	2	17
Myennement connecté	25	4	105535	1	14
Fortement connecté	30	5	107434	2	21

Tableau 5.2: Tbleau des expérimentations avec deux évidence

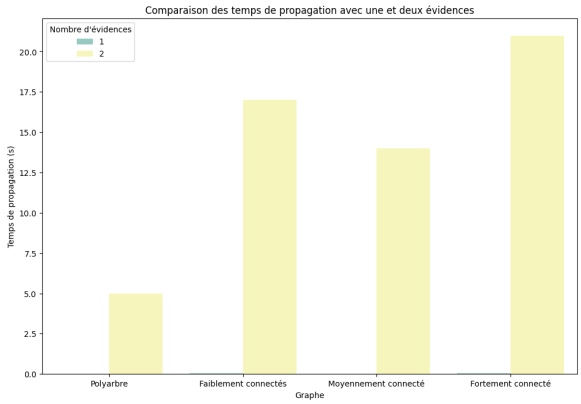
5.4.2.1 Interprétation

- Le temps d'exécution de l'inférence a augmenté par rapport aux expérimentations avec une seule évidence, ce qui est attendu car l'inclusion d'une deuxième évidence peut rendre le calcul plus complexe.
- Le temps de propagation, bien que relativement court, a augmenté, ce qui indique une augmentation de la complexité dans la prise en compte de deux évidences.
- La possibilité conditionnelle de l'intérêt est également plus élevée dans ces cas, indiquant que les évidences fournies ont une influence significative sur le nœud d'intérêt.

5.5 Conclusion



(a) Graphique des temps d'inférence avec une et deux évidences



(b) Graphique des temps de propagation avec une et deux évidences

Figure 5.1: Graphique des temps d'inférence et de propagation des expérimentations

Les expérimentations ont permis d'évaluer les performances d'un modèle graphique probabiliste dans divers scénarios, en variant le nombre de nœuds et le nombre maximal de parents. L'ajout d'une deuxième évidence a généralement entraîné une augmentation du temps d'exécution de l'inférence (voir FIGURE 5.1, soulignant l'impact de la complexité des calculs. Les polyarbres ont affiché des temps d'exécution relativement courts, tandis que les graphes fortement connectés ont exigé davantage de ressources. Les résultats mettent également en lumière l'influence significative des évidences sur la possibilité conditionnelle de l'intérêt. Ces observations fournissent des informations précieuses pour comprendre comment configurer efficacement les modèles graphiques probabilistes en fonction des caractéristiques spécifiques des graphes et des exigences d'une application donnée.

Conclusion Générale

En conclusion, cette étude a élargi nos compétences dans le domaine de la gestion des connaissances incertaines, fournissant des bases solides pour aborder des problèmes du monde réel qui impliquent des données sujettes à l'incertitude. Les compétences acquises au cours de ces Travaux Pratiques sont précieuses dans des contextes tels que l'analyse de données complexes, la prise de décision sous incertitude, et la modélisation de systèmes où la certitude complète n'est pas toujours réalisable.