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
Département d'Intelligence Artificielle et Sciences des Données (DIASD)

Master 2 Systèmes Informatiques Intelligents

Rapport TP RCR

Representation des Connaissances et Raisonnement

Théorie de Dempster-Shafer, Logique Floue, Reseau Bayesian et Logique Possibiliste

---

Présenté par :
Bokhtache Khawla
Anane Dalia Khadidja
Groupe 1

Responsable du TP :
H.F.Khellaf

Année universitaire 2024–2025

CONTENTS

CHAPTER 1
INTRODUCTION

1.1 Context

This document presents the work conducted as part of the project. The study addresses key challenges in the field and proposes solutions based on current research and methodologies.

1.2 Motivation

The motivation behind this project stems from the need to understand and solve practical problems encountered in real-world applications. This work aims to bridge the gap between theoretical concepts and their practical implementation.

1.3 Objectives

The main objectives of this project are:

- To analyze the current state of the art in the domain
- To develop and implement effective solutions
- To evaluate the performance and validate the results
- To provide recommendations for future work

1.4 Document Structure

This document is organized as follows:

- **Chapter 2** presents the theoretical background and related work
- **Chapter 3** describes the methodology and approach used
- **Chapter 4** details the implementation and experimental setup
- **Chapter 5** presents and discusses the results obtained
- **Chapter 6** concludes the document and outlines future perspectives

CHAPTER 2

LOGIQUE FLOUE

2.1 Introduction

Dans un système d'exploitation multitâche, plusieurs processus s'exécutent simultanément et se partagent les ressources matérielles, notamment le processeur (CPU). La gestion efficace du temps processeur est un problème fondamental des systèmes d'exploitation, généralement traité par des algorithmes d'ordonnancement classiques tels que Round-Robin ou la priorité fixe.

Cependant, ces méthodes reposent souvent sur des seuils stricts et des décisions binaires. Or, dans un environnement réel, la charge du système et l'importance des processus sont des notions imprécises et évolutives. La logique floue permet d'introduire une prise de décision progressive et plus proche du raisonnement humain.

Dans ce travail, nous proposons un **contrôleur flou** chargé de déterminer la **part de CPU à allouer à un processus** en fonction :

- de la charge globale du processeur,
- de l'importance du processus.

2.2 Présentation du contrôleur flou

Le contrôleur flou est de type **Mamdani**. Il repose sur trois variables linguistiques :

- deux variables d'entrée,
- une variable de sortie.

2.2.1 Entrée 1 : Charge CPU (CPULoad)

La variable *CPULoad* représente le taux d'occupation global du processeur.

- Univers de discours : $[0, 100]$ %
- Sous-ensembles flous :
  - **Low** (faible),
  - **Medium** (moyenne),
  - **High** (élevée).

Les fonctions d'appartenance utilisées sont de type trapézoïdal et triangulaire.

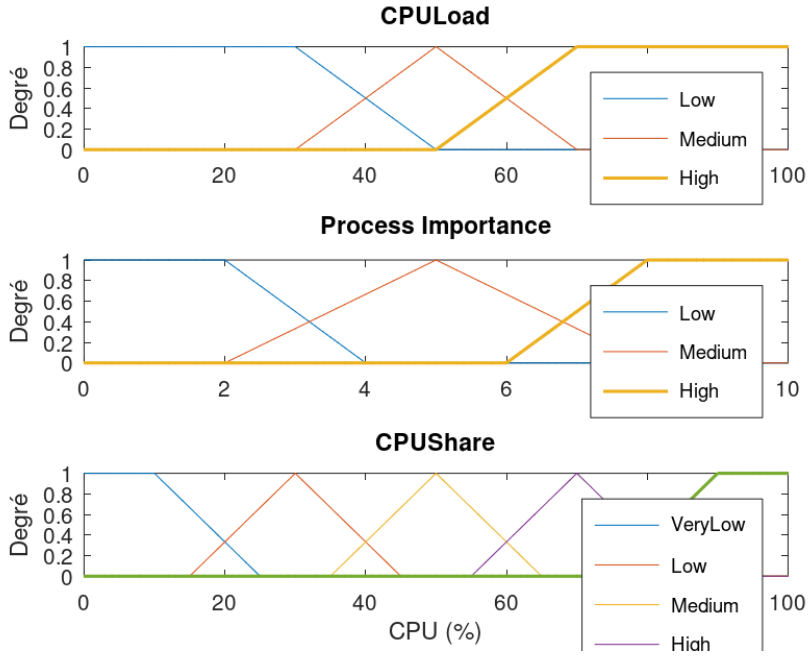

Figure 2.1: Fonctions d'appartenance de la variable CPULoad

2.2.2 Entrée 2 : Importance du processus (ProcessImportance)

Cette variable modélise l'importance relative d'un processus du point de vue du système d'exploitation.

- Univers de discours : $[0, 10]$
- Sous-ensembles flous :
  - **Low** (peu important),
  - **Medium** (importance moyenne),
  - **High** (processus critique).

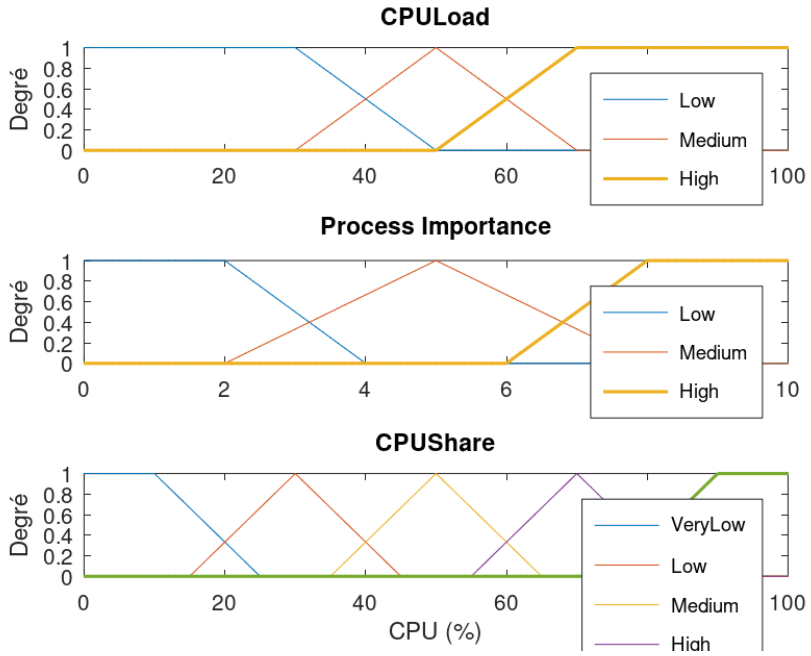

Figure 2.2: Fonctions d'appartenance de la variable ProcessImportance

2.2.3 Sortie : Part de CPU allouée (CPUShare)

La variable de sortie *CPUShare* représente le pourcentage de CPU attribué au processus.

- Univers de discours : $[0, 100]$ %
- Sous-ensembles flous :
  - **VeryLow**,
  - **Low**,
  - **Medium**,
  - **High**,
  - **VeryHigh**.

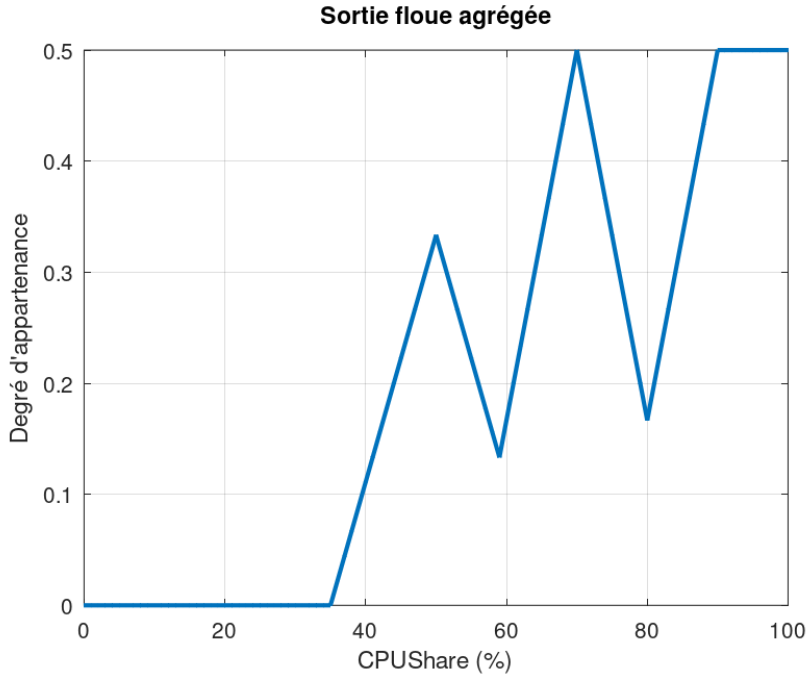

Figure 2.3: Fonctions d'appartenance de la variable CPUShare

2.3 Base de règles floues

La base de connaissances est constituée de règles de la forme :

Si CPULoad est X et ProcessImportance est Y alors CPUShare est Z

Les règles principales sont :

- Si la charge CPU est faible et le processus peu important, alors l'allocation est moyenne.
- Si la charge CPU est faible et le processus critique, alors l'allocation est très élevée.
- Si la charge CPU est moyenne, l'allocation dépend directement de l'importance du processus.
- Si la charge CPU est élevée, l'allocation est réduite, sauf pour les processus critiques.

Ces règles traduisent une politique réaliste de gestion du CPU visant à préserver les performances globales du système tout en favorisant les processus importants.

2.4 Exemple de fonctionnement du contrôleur

On considère les valeurs suivantes :

- Charge CPU : $CPULoad = 40\ \%$
- Importance du processus : $ProcessImportance = 7$

2.4.1 Étape 1 : Fuzzification

Après évaluation des fonctions d'appartenance :

- CPULoad est **Low** à 50% et **Medium** à 50%.
- ProcessImportance est **Medium** à environ 33% et **High** à 50%.

2.4.2 Étape 2 : Inférence floue

Quatre règles sont activées. L'opérateur logique utilisé est :

- ET : minimum,
- implication floue : minimum,
- agrégation des règles : maximum.

Après agrégation :

- CPUShare est **Medium** à 33%,
- CPUShare est **High** à 50%.

2.4.3 Étape 3 : Défuzzification

La méthode du **centre de gravité (COG)** est utilisée pour obtenir une sortie nette.

2.5 Résultat et analyse

Le contrôleur flou fournit la valeur suivante :

CPUShare allouée = 74,37 %

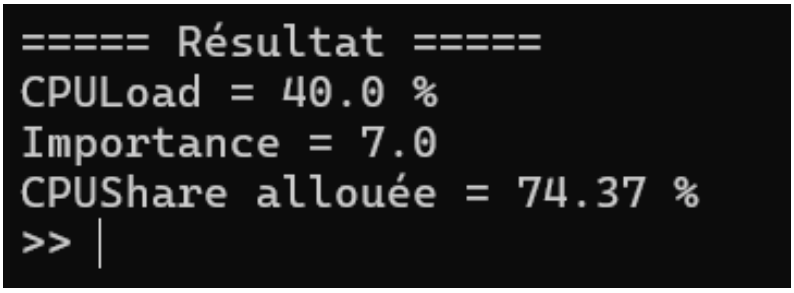

Figure 2.4: Résultat de la simulation du contrôleur flou

Cette valeur relativement élevée s'explique par :

- une charge CPU modérée (40%), laissant une marge d'allocation,
- une importance élevée du processus (7/10),
- l'activation dominante des règles associées aux ensembles *High*.

Le contrôleur flou adopte donc un comportement cohérent : malgré une charge non négligeable, il privilégie un processus important afin de garantir sa bonne exécution.

2.6 Conclusion

Ce travail a montré l'intérêt de la logique floue pour la gestion des ressources dans les systèmes d'exploitation. Contrairement aux méthodes classiques à seuils fixes, le contrôleur flou permet une prise de décision progressive, robuste et plus réaliste.

L'approche proposée peut être étendue à d'autres problématiques telles que l'ordonnancement dynamique, la gestion de la mémoire ou la qualité de service réseau.

CHAPTER 3

LOGIQUE POSSIBILISTE

Ce chapitre présente l'application de la logique possibiliste combinée aux solveurs SAT pour le raisonnement sous incertitude. Nous détaillons les fondements théoriques, le mécanisme de fusion entre ces deux approches, et présentons les résultats expérimentaux obtenus.

3.1 Fondements Théoriques Détaillés

La logique possibiliste est une extension de la logique classique permettant de gérer des connaissances incertaines ou prioritaires. Contrairement aux approches probabilistes, elle est purement ordinale dans sa version qualitative.

3.1.1 Mesures de Possibilité et de Nécessité

Elle repose sur deux mesures duales définies sur une échelle de priorité (généralement $[0, 1]$ ou un ensemble fini de labels) :

- **Possibilité (Π)** : Représente le degré de compatibilité d'une information avec les connaissances. $\Pi(\varphi) = 1$ signifie que φ est totalement possible.
- **Nécessité (N)** : Représente le degré de certitude ou de priorité. Elle est définie par la dualité **?** : $N(\varphi) = 1 - \Pi(\neg\varphi)$.

3.1.2 Base de Connaissances Stratifiée

Une base de connaissances possibiliste Σ est un ensemble de formules pondérées (φ_i, α_i), où α_i exprime le niveau de nécessité (priorité) de φ_i. La base est vue comme une pile de strates :

$$\Sigma = \{(\phi_1, \alpha_1), (\phi_2, \alpha_2), \ldots, (\phi_n, \alpha_n)\}$$

avec $1 = \alpha_1 > \alpha_2 > \cdots > \alpha_n > 0$. Chaque strate Σ_{α_i} contient les formules ayant une priorité supérieure ou égale à α_i **??**.

3.2 Le Solveur SAT : Définition et Rôle

Un solveur SAT est un outil algorithmique conçu pour résoudre le **Problème de Satisfaisabilité Booléenne**.

3.2.1 Fonctionnement

Étant donnée une formule logique en Forme Normale Conjonctive (CNF), le solveur doit déterminer s'il existe une affectation de valeurs de vérité (Vrai/Faux) aux variables qui rend la formule globale vraie.

- **SAT (Satisfiable)** : Il existe au moins une interprétation qui satisfait toutes les clauses.
- **UNSAT (Insatisfiable)** : Aucune affectation ne peut satisfaire la formule. Il existe une contradiction logique.

Dans ce TP, nous utilisons le solveur `Glucose 4`, basé sur l'algorithme CDCL (Conflict-Driven Clause Learning), extrêmement efficace pour traiter des milliers de clauses en un temps réduit **?**.

3.3 Fusion : Théorie des Possibilités et SAT

La fusion de ces deux domaines repose sur le **principe de réfutation** appliqué aux strates de la base.

3.3.1 Le Mécanisme de Projection

Pour vérifier si une variable I est déduite avec une certitude α, on ne considère pas toute la base, mais sa **coupe de niveau** α (notée Σ^*_α). Cette coupe est la projection de la base ne contenant que les formules dont le poids est $\geq \alpha$ **??**.

3.3.2 L'Inconsistance comme Preuve

La fusion s'opère par l'équivalence suivante :

$$\Sigma \vdash (\varphi, \alpha) \iff \Sigma^*_\alpha \cup \{\neg\varphi\} \text{ est INCONSISTANT (UNSAT)}$$

Le solveur SAT agit comme un "moteur de preuve" :

1. On "ignore" temporairement les poids et on ne donne au solveur SAT que les clauses logiques des strates supérieures.
2. On ajoute la négation de notre objectif ($\neg\varphi$) avec une priorité maximale (1).
3. Si le solveur répond **UNSAT**, cela signifie que même au niveau de priorité α, φ est inévitable (sa négation crée une contradiction) **??**.

3.4 Méthode de Dichotomie

Plutôt que de tester chaque strate séquentiellement (ce qui serait $O(n)$), nous utilisons la recherche binaire (dichotomie) pour trouver le niveau de coupure critique **??**.

- On initialise deux indices : $low = 1$ (la plus haute priorité) et $high = n$ (la plus basse).
- On calcule le milieu $mid = \lfloor (low + high)/2 \rfloor$.
- On teste la consistance de $\Sigma^*_{\alpha_{mid}} \cup \{\neg\varphi\}$ avec le solveur SAT.
- Si le résultat est **UNSAT**, on sait que φ est déductible à ce niveau ou supérieur, donc on ajuste $low = mid + 1$.
- Si le résultat est **SAT**, on ajuste $high = mid - 1$.
- On répète jusqu'à ce que $low > high$. Le niveau critique est alors α_{high}.

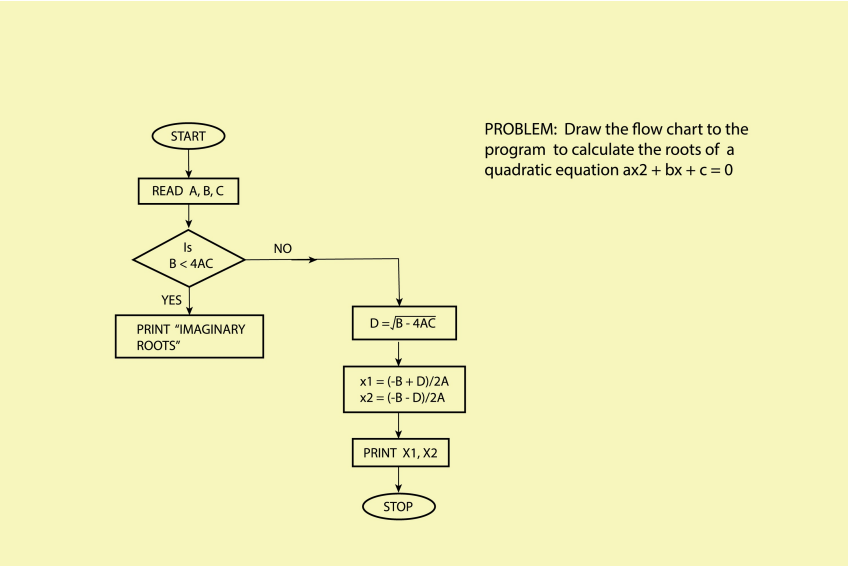

Figure 3.1: Représentation de la base de connaissances stratifiée (Source : Sujet de TP)

3.5 Résultats Expérimentaux

L'application de cette fusion sur la base fournie **?** a permis d'identifier les niveaux de nécessité exacts pour chaque variable.

Variable	$Val(\varphi, \Sigma)$	Observation SAT
b (2)	0.6	Inconsistant à partir de $r = 4$.
c (3)	0.6	Inconsistant à partir de $r = 4$.
d (4)	0.14	Consistant jusqu'à l'ajout de la dernière strate **?**.
a, e, f	0	Toujours Consistant (Satisfiable).

3.6 Conclusion

La force de cette approche réside dans la capacité du solveur SAT à gérer la complexité combinatoire du raisonnement logique, tandis que la théorie des possibilités fournit la structure hiérarchique nécessaire pour traiter l'incertitude qualitative. La méthode de dichotomie rend ce processus extrêmement performant, même pour des bases de connaissances volumineuses.

CHAPTER 4

CONCLUSION

Conclusion

Ce travail a illustré deux approches complémentaires du raisonnement sous incertitude appliquées à des problématiques informatiques concrètes.

La logique floue, à travers le contrôleur de type Mamdani, a permis de modéliser l'allocation de ressources CPU de manière progressive et intuitive, en s'appuyant sur des règles linguistiques traduisant l'expertise humaine. Cette approche offre une alternative flexible aux méthodes d'ordonnancement classiques basées sur des seuils rigides.

La logique possibiliste, combinée aux solveurs SAT, a démontré sa capacité à gérer des connaissances stratifiées par niveau de priorité. L'utilisation de la méthode de dichotomie avec le solveur Glucose 4 a permis d'identifier efficacement les niveaux de nécessité des propositions, illustrant ainsi la puissance de la fusion entre raisonnement qualitatif et techniques algorithmiques modernes.

Ces deux paradigmes montrent que l'intégration de l'incertitude et de l'imprécision dans les systèmes informatiques peut améliorer significativement leur adaptabilité et leur performance face à des environnements complexes et dynamiques.