Université Larbi Ben M'hidi Oum El Bouaghi
Faculté des Sciences Exactes & S.N.V
Département de Mathématiques et Informatique

Support de Cours

Outils d'Intelligence Artificielle

Destiné aux étudiants de Master -1- Informatique
parcours vision artificielle

Préparé par Dr. Tahar Guerram
MCA, Enseignant Chercheur.

- Février 2023 -

Table des matières

Introduction générale

Les premiers programmes intelligents commercialisés s'appellent les systèmes experts qui permettent d'automatiser le processus cognitif chez les experts humains. Bien que ces systèmes aient connu un succès extravagant depuis leur apparition au début des années soixante-dix jusqu'à la fin des années quatre- vingt, leurs limites, se résument en la difficulté de modéliser parfois les aspects intuitifs du raisonnement humain, en plus de la difficulté de prendre en charge aussi d'autres tâches complexes telles la vision artificielle et la traduction automatique des textes. Ces difficultés, ont poussé, plus tard, les développeurs et les chercheurs en systèmes experts à développer des systèmes experts hybrides utilisant des techniques un peu plus avancées de l'Intelligence Artificielle, telle la théorie de l'incertitude et la logique floue.

Dans le même ordre d'idée et pour pallier à la complexité de certains problèmes (tels les problèmes à explosion combinatoire comme le fameux problème du voyageur de commerce ou bien un problème de planification de tâches), des systèmes d'Intelligence Artificielle inspirés de la nature et de la biologie ont été développés. Ces derniers copient la façon dont les systèmes biologiques et naturels se comportent pour résoudre les problèmes auxquels ils doivent faire face quotidiennement, par exemple les algorithmes génétiques en intelligence artificielle sont inspirés du principe de la sélection naturelle qui stipule, que les individus d'une population qui s'adaptent le mieux à leur environnement, ont de fortes chances de survivre et de se reproduire pour donner naissance à d'autres populations. Aussi, les algorithmes de colonies de fourmis en intelligence artificielle, est un modèle artificiel qui s'inspire du comportement d'une colonie de fourmis lors de la recherche collective d'une source de nourriture.

Les individus appartenant à la population d'un système naturel ou biologique communiquent d'une manière directe ou indirecte (par exemple, les fourmis communiquent indirectement par dépôt d'une matière chimique dans l'environnement appelée phéromone et ressentie par les autres fourmis. Par contre dans un réseaux de neurones, les cellules neuronales communiquent directement par envoi de signaux) dans le but de de trouver une solution à un problème auquel la population est confrontée. Par exemple, résoudre un problème d'approvisionnement en alimentation ou résoudre un problème de sécurité de la population contre des ennemis, des intrus, ou des sources virales. La résolution de tels problèmes se fait d'une manière collective par la coopération des individus de la population et dont la finalité est de mettre le système en entier dans un état de confort ou un état désiré dit objectif de la population..

Inspirés notamment par ces concepts de comportements collectifs intelligents émanant de systèmes naturels et biologiques, les chercheurs en intelligence artificielle ont réussi par la suite à mettre en œuvre une nouvelle approche pour l'intelligence artificielle distribuée connue sous le nom de « systèmes multi agents ». Selon cette nouvelle approche, un système multi agents est composé d'un ensemble d'agents autonomes qui coopèrent et interagissent pour atteindre un but global du système.

Ce document est le fruit de l'enseignement durant plusieurs années de la matière Intelligence Artificielle au profit des étudiants de troisième année palier licence et des étudiants du palier Master. L'objectif

escompté est de mettre à la disposition de ces étudiants une référence bibliographique leur serviront en tant que support de cours unifié de la matière Intelligence Artificielle.

Ce document sera organisé de la manière suivante :

Le premier chapitre sera consacré à l'étude de l'historique de l'Intelligence Artificielle, le type de problèmes qu'elle est en mesure de résoudre ainsi que ses domaines d'application. Dans Le deuxième chapitre, seront présentés les différents formalismes de représentation de connaissances. Le troisième chapitre donnera une définition de l'Intelligence Artificielle, présentera l'architecture générale des systèmes experts et mettra l'accent sur l'importance de développement de tels systèmes. Dans le quatrième, chapitre il sera discuté du fonctionnement des systèmes experts notamment de la logique menée par le module de raisonnement, appelé moteur d'inférence. Le cinquième chapitre traitera du processus de développement des système experts.. Nous verrons dans le sixième et le septième chapitre respectivement, les systèmes experts incertains et les systèmes experts flous qui sont des extensions des systèmes experts classiques. Dans le huitième chapitre, il sera présenté le langage de l'intelligence artificielle PROLOG où une bonne partie sera consacrée à la programmation logique par contraintes.

Le neuvième chapitre sera dédié à la présentation des réseaux de neurones qui simulent le modèle biologique du cerveau humain. La présentation des algorithmes génétiques, qui incarnent le principe de la sélection naturelle, sera l'objet du dixième chapitre et dans le onzième chapitre, il sera présenté l'algorithme de colonies de fourmis qui simule le comportement naturel d'une colonie de fourmis lors de la recherche collective d'une source de nourriture. Le dernier et le douzième chapitre, sera consacré à l'étude des systèmes multi agents et sera illustrée par un exemple d'application de ces systèmes dans le domaine de la planification automatique.

Enfin une conclusion générale nous permettra de conclure ce document et de récapituler le contenu présenté.

Liste des figures

Liste des tableaux

Chapitre 1. Naissance de l'Intelligence Artificielle

1.1 Définition de l'Intelligence Artificielle

Plusieurs définitions de l'Intelligence Artificielle (I.A) ont été proposées [Russel and Norvig 2010] mais les définitions suivantes sont les plus utilisées :

"The automation of activities that we associate with human thinking, activities such as decision-making, problem solving, learning . . ." [Bellman, 1978].

"The study of the computations that make it possible to perceive, reason, and act." [Winston, 1992]

"The study of mental faculties through the use of computational models." [Charniak & McDermott, 1985]

Pour résumer, nous disons que L'IA est une branche de l'informatique visant à produire des systèmes informatiques permettant de simuler les facultés cognitives chez l'être humain. On entend par « facultés cognitives » les activités mentales telles: le raisonnement, la planification et l'apprentissage.

1. 2 Bref historique de l'Intelligence Artificielle

- **De 1943-1949**

- Apparition des premiers ordinateurs.
- Pitts and MCculoch ont proposé en 1943 un modèle de neurone formel simple inspiré du neurone biologique et ont montré que toute fonction décrite dans la logique de propositions peut être représentée par un réseau de neurones formels.
- Konrad Zuse un des pères des premiers ordinateurs a pu programmer les règles du jeu d'échecs en 1945
- En 1949, Donald Hebb démontre une simple règle d'apprentissage d'un réseau de neurones, connue sous la règle d'apprentissage de Hebb.

- **Durant les années 50**

- En 1950, Alan Turing, a proposé son test d'intelligence pour les machines, connu sous le nom de « test de Turing »
- En 1951, Marvin Minsky et Dean Edmonds ont conçu le premier processeur à base de quarante neurones artificiels. Ce prototype de processeur a permis à Minsky de montrer les limites des réseaux de neurones demandant des capacités de calcul importantes alors que la technologie des ordinateurs n'était qu'à ses premiers pas.
-Apparition des langages de programmation IPL1 (1956), qui est simplement le père du langage LISP et qui a favorisé l'apparition des premiers programmes de l'intelligence artificielle notamment les

programmes de jeu d'échecs et les démonstrateurs automatiques de théorèmes en logique des propositions.

- Le mot « d'intelligence artificielle » fut introduit pour la première fois en 1956 lors d'un workshop à « Darmouth College » aux Etats Unis organisé par Jhon Mccarthy , qui a réuni des chercheurs en mathématiques, en théories des automates et en intelligence des machines.
- Frank Rosenblatt a inventé le perceptron en 1957 et a démontré sa convergence plus tard.

- **Durant les années 60**

- En 1963, Jhon Mccarthy crée un laboratoire d'I.A à l'université de Standford où il a monté un projet dit « Shacky robotics project » dont le but et d'utiliser le raisonnement logique dans la résolution des problèmes de la robotique.
- En 1965, Robinson mis au point un démonstrateur automatique de théorème pour le langage logique des prédicats du 1er ordre.
- En 1968, Tom Evans invente le programme (Analogy) permettant de résoudre les problèmes d'analogie géométrique proposés pour estimer le coefficient d'intelligence (Q.I)
- Dés 1969, on pensait déjà pouvoir faire du traitement automatique du langage naturel comme la traduction automatique par l'analyse syntaxique et les dictionnaires électroniques ainsi que le développement de programmes intelligents pouvant remplacer les experts humains.

- **De la fin des années 1960 – Début des années 1980**.

- L'IA devient une industrie et plusieurs programmes de l'IA appelés les « Systèmes Experts » ont vu le jour. On peut citer à titre indicatif, le programme MYCIN, un système expert pour le diagnostic des maladies du sang et aussi le programme ELISA un système expert psychothérapeute.

- **A partir de l'année 1986**

- Difficulté de la tâche d'automatisation de facultés humaines telles la traduction automatique, la vision artificielle ou la Compréhension du langage naturel et aussi le constat que les systèmes experts ne peuvent remplacer à tous les coups l'être humain quel que soit le domaine étudié.
- Outre ces raisons, le développement de la technologie des ordinateurs a rebasculé la communauté des chercheurs en IA à reconsidérer les réseaux de neurones et les modèles inspirés de la nature en général pour relever le défi et résoudre les problèmes de nature complexe.

- Enfin, c'est **en 1987** que l'I.A devient une science informatique à part entière. .

1.3 Types de problèmes auxquels est appliquée l'Intelligence Artificielle

L'IA est l'ensemble des procédés et algorithmes intelligents servant à résoudre des problèmes dont la solution ne peut être simple et précise. On peut formuler cette phrase autrement en disant que l'IA est applicable à tout problème dont la solution algorithmique précise n'existe pas ou si elle existe alors son temps de calcul dépasse l'échelle humaine.

1.4 Exemples de domaines d'application de l'Intelligence Artificielle

L'intelligence artificielle intervient dans une variété de domaines tels que :

- Dans le domaine **bancaire** les systèmes experts permettent l'évaluation de risque lié à l'octroi d'un crédit (credit- scoring, en Anglais)
- Dans le domaine **militaire** grâce à l'intelligence artificielle on arrive à concevoir des systèmes autonomes tels que les drones, les systèmes de commandement et d'aide à la décision.
- Dans le domaine **médical** on trouve les systèmes experts d'aide au diagnostic des maladies
- Dans le domaine **logistique** on conçoit des approches heuristiques de type résolution de problème de satisfaction de contraintes.
- Dans le domaine **éducatif** on trouve les systèmes d'apprentissage électroniques permettent de proposer des contenus spécifiques à chaque utilisateur en fonction des compétences et niveau de cet utilisateur.
- En **Industrie,** la robotique moderne inclut très souvent des briques d'intelligence artificielle. C'est le cas des robots susceptibles d'apprendre un geste à partir d'une manipulation réalisée par un opérateur. Dans la maintenance prédictive aussi, des systèmes experts peuvent détecter, avant les êtres humains, des signes d'usure sur une machine.
- Enfin**,** dans le domaine **commercial,** grâce à l'e-commerce, on connaît toutes les recommandations personnalisées d'achat et les publicités profilées, ainsi que la détection des tendances d'achat, grâce à l'analyse des échanges sur les réseaux sociaux.

1. 5 Test de Turing

Ce test fut créé par Alan Turing, professeur à l'Université de Manchester [Turing 1950] , en 1950. Il fut présenté pour la première fois dans l'article « Computing Machinery and Intelligence ». L'objectif était de définir si les '' machines sont capables de penser''. Ce test fut à la fois très critiqué et très influent. Au fil des années, il est devenu un concept très important dans le domaine de la philosophie de l'intelligence artificielle.
Le principe du Test de Turing est simple. Un évaluateur humain est chargé de juger une conversation textuelle entre un humain et une machine (Voir Figure 1).

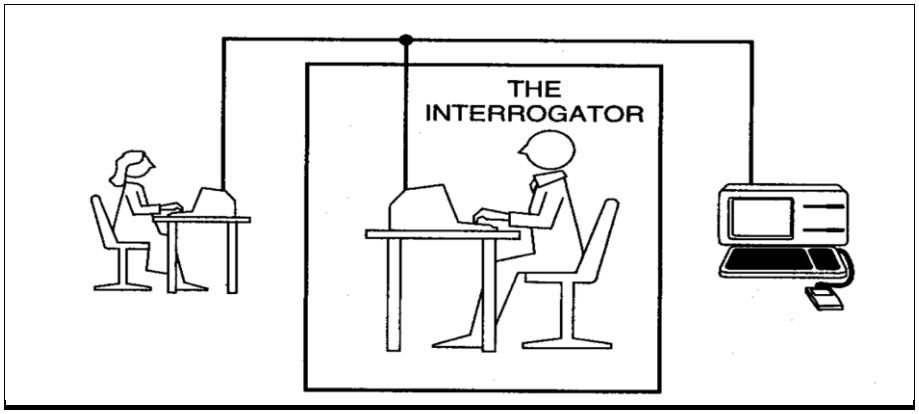


Figure 1 : Test de Turing

L'évaluateur sait que l'un des deux participants est une machine, mais ne sait pas lequel. S'il n'est pas en mesure de discerner l'homme de la machine après cinq minutes de conversation, alors on dit que la machine a passé le test de Turing avec succès.

Il existe des exemples de programmes ayant passé le test de Turing. Nous citons à titre d'exemples :

- ELISA : un programme psychothérapeute.
- CLEVERBOT : **e**xiste sur le net. Vous pouvez l'essayer sur la page : https://www.cleverbot.com/

Chapitre 2: La représentation des connaissances

2.1 Introduction

Ce chapitre permet de présenter les différents formalismes de représentation des connaissances. Cette représentation permet la manipulation de ces connaissances par un ordinateur. Il n'existe pas un formalise de représentation de connaissances applicable à tous les cas, mais ceci dépend du problème en main, et parfois dans le cas d'un problème complexe, on utilise plus qu'un modèle de représentation pour pallier à cette complexité.

2.2 Les formalismes de représentation des connaissances

Il existe plusieurs formalismes de représentation de connaissances:

2.2.1 Le formalisme logique

Ce formalisme permet de représenter les connaissances par des assertions logiques d'une manière déclarative. La logique se préoccupe de l'étude des raisonnements, de leur modélisation et de leur validation. C'est un formalisme basé sur des fondements mathématiques et donc ayant une syntaxe et une sémantique bien définies. Une logique manipule des propositions ou assertions par l'intermédiaire d'opérateurs logiques.

Il existe plusieurs logiques dont le degré d'expressivité diffère d'une logique à une autre. On en distingue deux grandes classes : la classe des logiques classiques qui renferme la logique propositionnelle et la logique des prédicats et la classe des logiques non classiques telles la logique de défaut, la logique temporelle et logique floue. Nous illustrons ci-dessous l'approche logique à travers la logique des prédicats et la logique floue.

- **Logique des prédicats**

La logique des prédicats, appelée aussi logique du premier ordre fut proposée pour la première fois par le mathématicien allemand Gottlob Frege en 1879 [Frege, 1879], est une extension de la logique des propositions par l'introduction de la notion de prédicat et de deux nouveaux opérateurs de quantification qui sont le quantificateur universel ($\forall$) et le quantificateur existentiel. ($\exists$).

La logique des prédicats a été introduite pour considérer les ensembles d'individus ou d'entités où un prédicat énonce les propriétés des individus ou décrit une relation entre deux ou plusieurs ensembles d'individus. Le langage de la logique des prédicats est formé de termes qui sont les variables et les constantes ou un foncteur qui opère sur plusieurs termes. Un atome est un prédicat reliant un certain nombre de termes. Une combinaison d'atomes par le biais d'opérateurs logiques ou de quantification est aussi un atome.

La liste des axiomes de la logique des prédicats est comme suit :

1. $(P \Rightarrow (Q \Rightarrow P))$
2. $((P \Rightarrow (Q \Rightarrow R)) \Rightarrow ((P \Rightarrow Q) \Rightarrow (P \Rightarrow R)))$
3. $((\neg P \Rightarrow \neg Q) \Rightarrow ((\neg P \Rightarrow Q) \Rightarrow P))$

4. $\forall x\ (P \Rightarrow Q) \Rightarrow (P \Rightarrow \forall x\ Q)$; où « x » ne figure pas dans P et elle est libre dans Q.
5. $(\forall x\ Q \Rightarrow Qt/x)$.

Où Q t/x désigne la formule obtenue en remplaçant uniformément « x » par « t » dans Q et que ni « x » ni aucune variable de « t » ne sont quantifiées dans Q.

- **Règles d'inférence** :

Le modus ponens : Déduit la formule Q des formules P et $P \Rightarrow Q$.
Modus tollens: Déduit la formule $\neg P$ des formules $\neg Q$ et $\neg Q \Rightarrow \neg P$.
Principe de résolution : Déduit $P \wedge Q$ des formules $X \vee P$ et $\neg X \vee Q$

- **Substitution de termes** :

Si t est un terme quelconque alors on peut déduire $\theta(t)$ de $\forall x\ \theta(x)$.

- **<u>Exemple 1</u>**

Traduire les assertions suivantes en logique de prédicats

a) Tout ingénieur informaticien est capable de concevoir des programmes informatiques.
b) Tout programmeur sait développer les différents types de programmes.
c) Tout chef de projets informatiques sait gérer les ressources informatiques humaines et matérielles.

- **<u>Solution</u>**

a) inginformaticien(x) : « x est un ingénieur informaticien »
programme (y) : « y est un programme »
concevoir (x, y) : « x conçoit y »

$\forall x\ \exists y\ ((\text{inginformaticien}(x) \wedge \text{programme}(y)) \Rightarrow \text{concevoir}(x, y))$

b) programmeur(x) : « x est un programmeur »
programme (y) : « y est un programme »
developper (x,y) « x developpe y »

$\forall x\ \forall y\ ((\text{programmeur}(x) \wedge \text{programme}(y)) \Rightarrow \text{developper}(x, y))$

c) chefinfo (x) : « x est un chef de projets informatiques »
resinfhum(x) : « x est une ressource informatique humaine »
resinfmat (x) : « x est une ressource informatique matérielle »
savoirgerer(x,y) « x sait gérer y »

$\forall x\ \forall y\ \forall z\ ((\text{chefinfo}(x) \wedge \text{resinfohum}(y) \wedge \text{resinfomat}(z)) \Rightarrow (\text{savoirgerer}(x,y) \wedge \text{savoirgerer}(x,z))$

- **Exemple 2**

a) Que peut-on déduire des assertions :
« Mohamed est au travail »,
« Mohamed n'est pas dans son bureau ou Mohamed est au travail » ?

b) Que peut –on déduire des assertions :
« Mohamed est malade ou Mohamed est au travail »,
« Mohamed n'est pas malade ou Mohamed est dans son bureau » ?

- **Solution**

P : « Mohamed est au travail »
Q : « Mohamed est dans son bureau »
R : « Mohamed est malade »

a) Nous avons : P ; $\neg Q \vee P$; $(\neg Q \vee P) \Leftrightarrow (P \Rightarrow Q)$ donc en déduit selon la règle du modus ponens la proposition Q.
b) Nous avons : $R \vee P, \neg R \vee Q$ donc en déduit selon la règle de résolution la proposition $P \wedge Q$.

- **Limites et avantages de la logique des prédicats**

Bien que la logique des prédicats soit un langage formel ayant des fondements mathématiques, elle souffre comme même de limites liées à l'expressivité. Elle ne peut pas par exemple exprimer l'exception à une règle, l'imprécision, l'obligation ou la notion de temps. Pour faire face à cette faiblesse d'expressivité d'autres logiques ont été proposées telles : la logique de défauts, la logique temporelle la logique floue. De plus, à cause de ses fondements mathématiques, la logique des prédicats est un modèle qui exige de la rigueur formelle, manque de lisibilité et de convivialité pour les personnes non du domaine.

1.2.2 Les règles de production

La notion de règle de production trouve son origine dans les travaux de Noam Chomsky [Chomsky, 1956] [Chomsky, 1957] sur les grammaires et les langages. Une règle de production est de la forme IF « condition » THEN « conclusion ». La partie conclusion est donc déduite si la partie condition est vérifiée. Un système à base de règles de production est constitué de:

- Une mémoire de travail qui représente les faits qui sont actuellement vraies.
- Un ensemble de règles de production IF- THEN: si certaines conditions sont vraies (certains faits sont dans la mémoire de travail), alors une conclusion est déclenchée afin de rajouter ou supprimer des faits.
- Un interpréteur ou mécanisme de raisonnement qui contrôle l'application des règles, étant donné certains faits.

Il faut bien noter qu'il existe trois types de mécanismes de raisonnement : chaînage en avant, chaînage en arrière et le chaînage mixte. Plus de détails seront donnés au sujet de ce formalisme de représentation dans le chapitre 4.

1.2.3 Les objets structurés

Ici les connaissances sont représentées par des objets ayant des relations entre eux. On distingue deux types de cette représentation : représentation par les réseaux sémantiques et représentation par les frames.

- **Les réseaux sémantiques**

Est un graphe orienté étiqueté proposé par [Quinlan, 1961] [Quinlan, 1968] dont les nœuds représentent les classes et les objets reliés par des relations entre eux. Ces relations peuvent être des relations de hiérarchie « is a » , relations de composition « is composed of » , relations d'attributs (« has –a »), relations topologiques (« up », « under », « righ »t, « left » …) ou toutes autres relations pouvant exister entre les classes et objets.

Exemple 3

Représenter les connaissances suivantes par un réseau sémantique (R.S)

- Les chiens sont des mammifères
- Les oiseaux ont des ailes
- Les mammifères sont des animaux
- Les chauves-souris ont des ailes
- Les oiseaux sont des animaux
- Les chauves-souris sont des mammifères
- Les poissons sont des animaux
- Les chats sont des mammifères
- Les vers sont des animaux
- Les chats mangent les poissons
- Les chiens cohabitent avec les chats
- Les oiseaux mangent les vers
- Les chats sont poilus
- Les poissons mangent les vers
- Les chiens sont poilus

a) Supposons que Tom est un chat. Quelles connaissances peut-on avoir au sujet de Tom en utilisant ce R.S ?

b) Supposons que Tom ne mange pas les poissons. Comment représenter cette connaissance dans le R.S que vous avez construit ?

Solution

La représentation des données par un réseau sémantique est donnée par la figure 2 sur la page suivante.

a) Tom est mammifère qui a du poil et qui mange les poissons et qui cohabitent avec les chiens.

b) Si Tom ne mange pas les poissons alors il présente une exception pour sa classe de chats. Cette relation doit être redéfinie pour Tom par la négation.

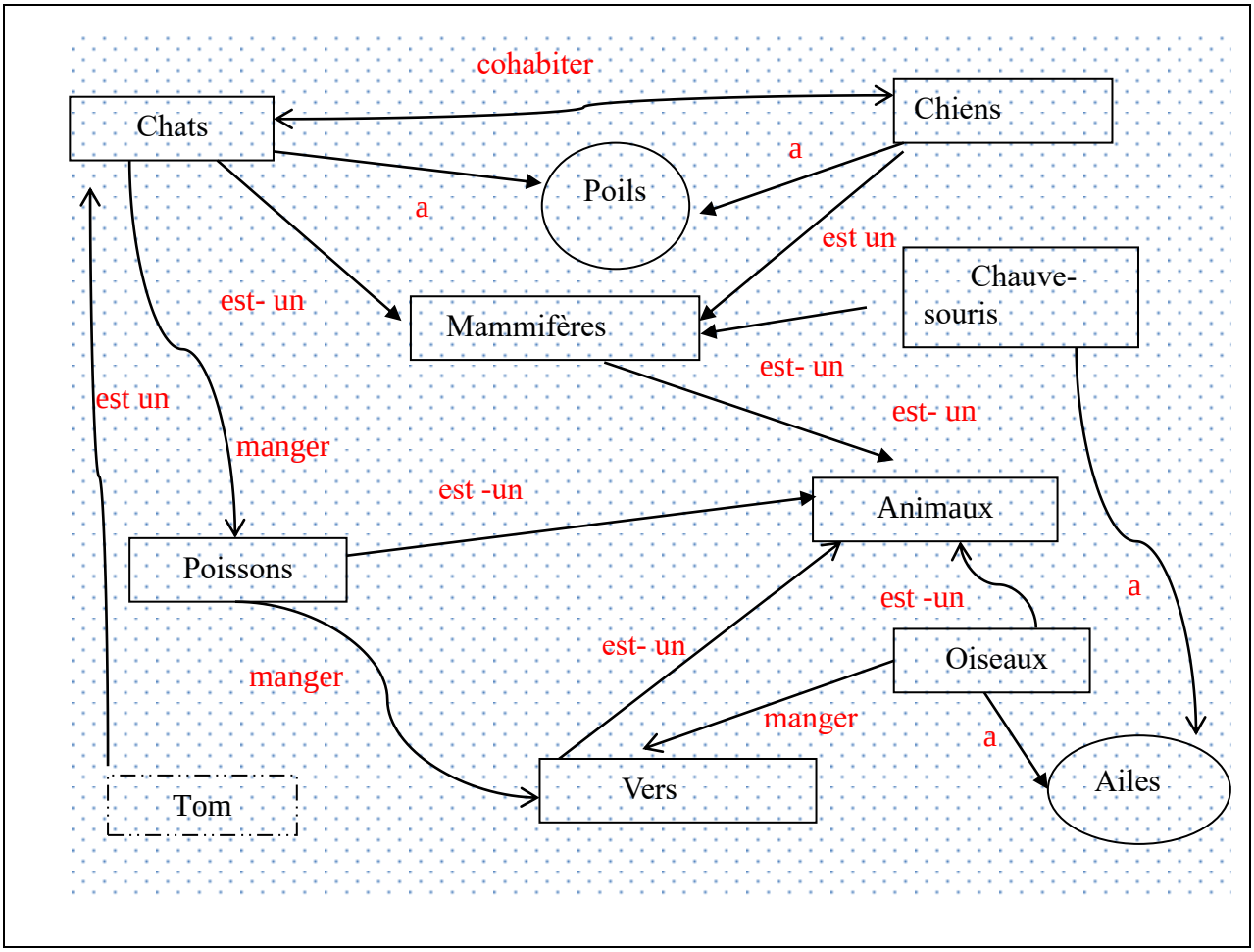


Figure 2 : Réseau sémantique

- **Les frames**

Le terme frame [Minsky, 1975], cadre en Français, est une structure servant à stocker les informations relatives à un concept particulier. La notion de frame est utilisée pour représenter les objets ou les évènements typiques tels que oiseau- type, voiture- type, repas- type etc…. Un(e) frame supporte l'héritage. On hérite des propriétés des parents sauf si les attributs des enfants sont en contradiction avec ceux des parents.

Exemple 4
Soient les frames suivants :

Mammifère
sous-classe: **Animal**
sang-chaud: oui

Eléphant
sous-classe: **Mammifère**
couleur: gris
taille: large
pays: afrique

Clyde
instance: Eléphant
couleur: rose
Sexe : male
propriétaire: Ahmed

Nellie
instance: Eléphant
taille: petit
Sexe : femelle

De cette représentation, Que peut-on inférer au sujet de Clyde et de Nellie ?

Solution

- On peut inférer que Clyde est un éléphant de couleur rose, ayant comme propriétaire Ahmed et qui vit en Afrique.
- On peut inférer aussi que Nellie est une éléphante de petite taille et qui vit en Afrique.

- **Limites et avantages des objets structurés**

Les réseaux sémantiques et les frames sont des modèles conviviaux qui offrent beaucoup de lisibilité et de souplesse mais en revanche n'ont pas une sémantique formelle sous-jacente.

1.2.4 Logique de description

C'est une méthode de représentation et de raisonnement basée sur la logique des prédicats et tire ses origines des réseaux sémantiques et des frames [Russel and Norvig 2010]. C'est donc une représentation de connaissances hybride qui allie la souplesse et la lisibilité des objets structurés avec les fondements mathématiques de la logique des prédicats. Elle définit les individus et concepts pertinents du domaine étudié (dite terminologie ou TBOX) ainsi que leurs propriétés et les relations entre eux (dites assertions ou ABOX).

La partie terminologie (TBOX) définit les concepts qui sont des prédicats unaires ainsi que les rôles qui sont des prédicats binaires en plus d'un ensemble de contraintes et propriétés de ces prédicats exprimées à l'aide d'opérateurs ou constructeurs.

La partie assertions (ABOX) définit un ensemble de constantes et d'instances de concepts et des rôles sur ces constantes.

Les composantes d'une logique de description sont donc :

- Les individus ou les objets qui sont des constantes ou chaque constante est identifiable par un nom écrit en majuscule.
- Les concepts qui sont des prédicats unaires identifiés par des noms commençant par une lettre majuscule.
- Rôles ou prédicats binaires indiquent la relation entre les objets et identifiables par des noms commençant par une lettre minuscule.
- Les constructeurs (opérateurs) sur les concepts : ¬, ≡,⊓, ⊑, ⊔,∃,∀,⊥, ⊤

Les inférences dans un système à logique de description sont : [Russel and Norvig 2010]
- La subsumption qui permet de dire qu'une catégorie est une sous-catégorie d'une catégorie ;
- La classification qui permet de dire si un objet fait partie d'une catégorie.

Exemple 5
Traduire les connaissances suivantes en logique de description

Les femmes sont des personnes de sexe féminin.
Les hommes sont des personnes de sexe non féminin.
Le masculin est l'inverse de féminin.
Les mères sont des femmes qui ont des enfants.
Les pères sont des hommes qui ont des enfants.
Les pères et les mères font les parents.
Les grand-mères ont des enfants qui sont eux des parents.
Une mère avec plusieurs enfants a au moins trois enfants.
Une mère sans fille a des enfants tous de sexe masculin.
Samia est une mère qui n'a pas de filles.
Les enfants de Samia sont Said et Sami.
Madiha est une femme.
Said est un père et a une fille qui s'appelle Madiha.

Solution

TBox :

Personne ⊑ ⊤
Femme ≡ Personne ⊓ Femelle
Homme ≡ Personne ⊓ Male
Femelle ⊓ Male ≡ ⊥
Mere ≡ Femme ⊓ ∃aEnfant.Personne
Pere ≡ Homme ⊓ ∃aEnfant.Personne
Parent ≡ Pere ⊔ Mere

GrandMere ≡ Mere ⊓ ∃aEnfant.Parent
MereAvecPlusieursEnfants ≡ Mere ⊓ ≥3 aEnfant
MereSansFille ≡ Mere ⊓ ,aEnfant.¬Femme
Epouse ≡ Femme ⊓ ∃aCommeMari. Homme

ABox :

MereSansFille(Samia)
Femme(Madiha)
Pere(Said)
aEnfant(Samia,Said)
aEnfant(Samia,Sami)
aEnfant(Said,Madiha)

CHAPITRE 3. Les Systèmes Experts

3.1 Introduction

La notion de système expert est une notion assez ancienne apparue dans les années 70 avec le célèbre software MYCIN dont le but était d'aider les médecins à effectuer le diagnostic est le soin des maladies infectieuses du sang. La version de base de MYCIN contenait 200 règles ensuite 300 règles après que les méningites ont été ajoutées. Aujourd'hui, les systèmes experts constituent une technologie bien définie faisant partie des systèmes software à base de connaissances. Les systèmes experts ont comme finalité la modélisation de la connaissance et de raisonnement d'un expert (ou d'un ensemble d'experts) dans un domaine donné. Pour cela, quatre types d'acteurs doivent contribuer à l'élaboration d'un système expert à savoir : l'utilisateur final, l'expert du domaine, l'ingénieur de connaissances et l'ingénieur en développement. L'interaction entre ces acteurs permet l'élaboration d'une première version de systèmes experts contenant une base de connaissances, une base de faits et un moteur d'inférence effectuant une forme définie de raisonnement.

3.2 Définition

- Un Système expert est un programme intelligent qui utilise des connaissances et effectue des inférences logiques sur ces connaissances afin de résoudre les problèmes et répondre aux questions relatives à un domaine d'expertise donné. [François & Laurent 2006]

- Bref, Un système expert est un programme informatique dont le but est de simuler le travail d'experts dans un domaine donné.

3. 3 Rôle d'un Système Expert

Un système expert est un outil capable de reproduire les mécanismes cognitifs d'un expert humain dans un domaine particulier. Plus précisément, un système expert est un logiciel capable de répondre à des questions, en effectuant un raisonnement à partir de faits et de règles connus.

3.4 Architecture d'un Système Expert

L'architecture d'un système expert typique est constituée de deux parties_ (Voir Figure.3 ci-dessous)

3.4. 1 Le Shell : C'est la partie purement software du système expert. Il est composé de :

a) L'interface utilisateur (User interface)
Sert à simplifier la communication entre utilisateur et le système expert, elle peut utiliser la forme question-réponse, le menu, le langage naturel etc…

b) Le moteur d'inférence (Inference engine)

C'est un programme qui met en œuvre des mécanismes généraux de combinaison des connaissances assertionnel1es (faits) et connaissances opératoires (règles) [Outellet & Tessier 1987]. Les stratégies utilisées peuvent être très différentes d'un moteur à un autre. Le moteur puise parmi les règles, les

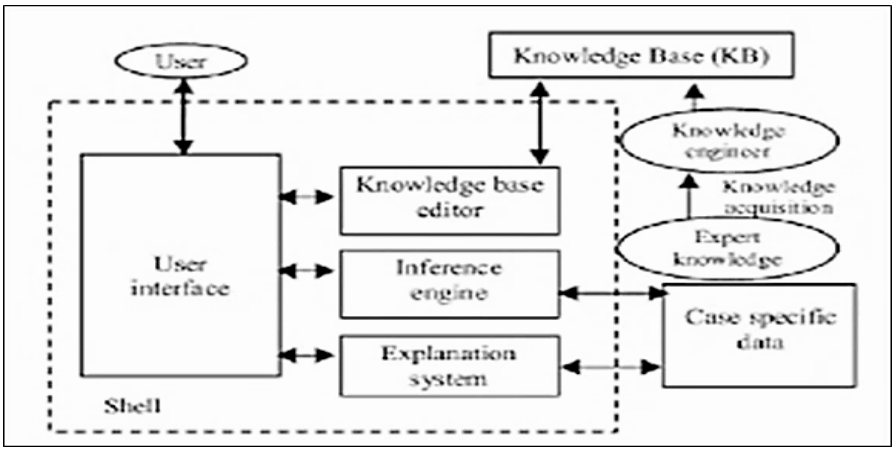


Figure 3: Architecture d'un Système Expert

enchaîne, jusqu'à satisfaire une condition d'arrêt qui dépend du moteur d'inférence et de la base des connaissances disponibles. La résolution d'un problème passe donc par l'application d'un ensemble de règles par le moteur d'inférence en appliquant une stratégie de résolution en utilisant les connaissances disponibles dans le but d'en dériver de nouvelles connaissances.

Le cycle de base d'un moteur d'inférence passe par deux phases : **Phase d'évaluation** et **phase d'exécution** (voir Figure 4 ci-dessous)

- **La phase d'évaluation**

La phase d'évaluation comprend généralement trois étapes (dépendant de la complexité du moteur d'inférence): **la sélection, le filtrage et la résolution des conflits**. La sélection (ou restriction) détermine par l'état présent de la base des faits et la base des règles (dans le cas d "un retour-arrière du moteur: par un état passé) un sous-ensemble de la base des faits et un sous-ensemble de la base des règles qui méritent d'être comparés lors de l'étape du filtrage. Une méthode courante consiste à répartir les faits et les règles en classes particulières en exploitant des connaissances particulières du domaine d'application. Comme les systèmes experts, en général, sont devenus de plus en plus importants et de plus en plus complexes, les questions d'efficacité ont nécessité l'élaboration de structures sur la base des règles et sur la base des faits. Par exemple, pour permettre une accessibilité plus rapide aux règles qui sont applicables dans une situation donnée, sans avoir à vérifier si toutes les règles sont applicables, une façon de faire est d'indexer les règles ou les partitionner selon les conditions qui les déclencheront. Une autre façon peut alors consister à privilégier les faits à établir.

- **Etape de La sélection (ou restriction) :**

Détermine par l'état présent de la base des faits et la base des règles (dans le cas d "un retour-arrière du moteur: par un état passé) un sous-ensemble de la base des faits et un sous-ensemble de la base des règles qui méritent d'être comparés lors de l'étape du filtrage. Une méthode courante consiste à répartir les faits

et les règles en classes particulières en exploitant des connaissances particulières du domaine d'application.

- Etape de filtrage

Dans cette étape on affecte à chaque variable de la partie Conditions de la règle les valeurs des faits appropriés enregistrés dans la base des faits, Un ensemble des règles pouvant être déclenché est donc pouvant être formé . Cet ensemble est appelé « ensemble de conflits »

- Etape de résolution de conflits

S'il y a plus d'une règle admise à déclencher, on va choisir la règle à activer suivant la stratégie de résolution de conflits utilisée , Cette stratégie peut être simple (la première règle de la liste, la moins complexe, la moins utilisée ...) ou complexe (la plus prometteuse, la plus fiable, la moins coûteuse...). Les stratégies de résolution de conflit communément utilisées sont :

- La première qui est filtrée, ou la première est définie en termes d'ordre sur la base des règles;
- La règle la plus prioritaire où la priorité est définie par le programmeur correspondant aux caractéristiques de la demande de la tâche à accomplir;
- La règle la plus spécifique, qui est celle dont la partie condition (la partie déclencheur) détaille le mieux la base des faits;
- Une nouvelle règle qui vient d'être déclenchée, et qui n'était pas apparue précédemment;
- Par un ordre tout simplement arbitraire;
- Par l'exploration de toutes les règles applicables en parallèle;
- La règle se référant à l'élément le plus récent ajouté à la base des faits;
- La règle qui est jugée de par sa forme, la moins complexe;
- La règle qui semble d'emploi la moins coûteuse;
- Une combinaison des approches précédentes.

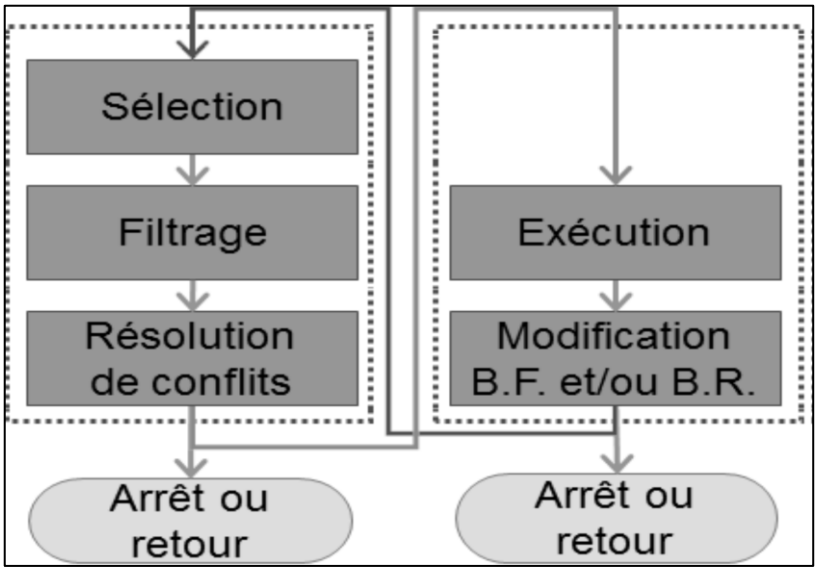


Figure 4 : Cycle de base d'un moteur d'inférence

- Phase d'exécution :

Dans cette phase du cycle de base d'un moteur d'inférence, ce dernier commande la mise en œuvre des actions définies par les règles résultant de la phase évaluation. Les stratégies de mise en œuvre peuvent être très différentes d'un moteur à l'autre. Lorsque l'ensemble des règles est vide certains moteurs très simples s'arrêtent; on dit que ces moteurs ont un "régime de contrôle irrévocable". Par contre, on parle d'un moteur à "régime de contrôle par tentative" lorsqu'il y' a remplacement de déclenchements de règles par d'autres. Pour signifier qu'un moteur revient sur une résolution de conflits antérieurs en remettant en cause des déclenchements de règles on dit qu'il opère un "retour en arrière'). En résumé un retour en arrière s'opère lorsqu'une impasse est rencontrée au cycle n, un retour-arrière est réalisé au cycle (n + l) par un retour au contexte du cycle (n-l), cependant l'ensemble des règles résultant de la résolution des conflits au cycle n-l est automatiquement écartée de l'ensemble de conflits du cycle (n + l). Ce type de retour-arrière est appelé « retour en arrière chronologique »; d'autres types de retour arrière déterminent différemment le contexte de retour.

c) Le module d'explication (Explantation system)

Permet au système expert d'expliquer son chemin de raisonnement, c'est-à-dire fournir la trace d'exécution par le système d'une requête formulée par l'utilisateur.

d) L'éditeur de la base de connaissances (Knowledge base Editor)

Permet l'édition des connaissances dans la base de connaissances.

3.4.2 La Base de Connaissances (Knowledge base)

Contient les connaissances concernant la résolution du problème. Ces connaissances sont de deux types, les règles (connaissances opératoires) et les faits (connaissances assertionelles) L'ensemble des règles est appelé la **base des règles** et l' ensemble des faits est appelé la **base de faits. .** Cette dernière contient les données spécifiques liées à l'application traitée. Elle contient les conclusions partielles trouvées lors d'une 'inférence.

A cause de l'importance de l'acquisition de connaissance dans l'architecture d'un système expert, nous allons étudier ce processus important, appelé communément ingénierie des connaissances (Knowledge Engineering) et les personnes qui ont la responsabilité de réaliser cette tâche sont appelées Ingénieurs cogniticiens ou tout simplement Ingénieurs de connaissances. L'ingénieur de connaissances a pour mission de prendre en charge la modélisation des connaissances et le raisonnement d'un expert humain sous une forme manipulable par le programme informatique. Il est préférable pour un bon Ingénieur de connaissances de satisfaire les caractéristiques désirées suivantes:

- Patience;
- Persévérance (volonté);
- Attention;
- Curiosité;
- Bonne compréhension;

- Crédibilité technique;
- Bonne motivation;
- Bonne organisation;
- Sens de communication.

Les techniques d'acquisition de connaissances sont de deux types : manuelles et automatiques

- **Techniques d'acquisition manuelle de connaissances**

1. Interviews

L'interview est la méthode la plus utilisée pour l'acquisition des connaissances de l'expert. Avec une méthode non structurée, le cogniticien discute avec l'expert de domaine au sujet de son raisonnement face à un problème. L'expert peut expliquer le processus de raisonnement soit verbalement ou bien en combinaison avec l'exécution de la tâche. L'ingénieur de connaissance sauvegarde toutes les informations nécessaires et pose des questions pour rassembler plus d'informations concernant les approches suivies par l'expert pour résoudre le problème. L'interview non structurée au début de la phase d'acquisition est important afin d'acquérir une masse importante d'informations. Ensuite une interview structurée est utilisée pour extraire des informations spécifiques concernant les techniques particulières de résonnement de l'expert.

2. Interprétation de la tâche et de protocoles

L'observation de l'expert humain dans sa tâche de résolution de problème peut être productive pour acquérir des informations. Les taches préliminaires observées doivent être habituelles et simples pour que le cogniticien prenne une vision globale sur le raisonnement de l'expert. Le processus doit être enregistré en audio ou bien en vidéo pour obtenir une vision correcte du raisonnement de l'expert. L'enregistrement peut être analysé ensuite par l'ingénieur de Connaissance. L'ingénieur peut demander à l'expert de refaire la tâche pour poser quelques remarques et ajouter des commentaires. Dans cette méthode, l'étude des actions de l'expert appelée aussi analyse des contrôles.

3. Brainstorming

Brainstorming est une méthode de génération de nouvelles idées. Dans le brainstorming, une équipe des experts se regroupe pour discuter ensemble les alternatives de résolution des problèmes de décision. Les membres de discussions peuvent être de différents services, chacun à son expertise et son raisonnement. Cette combinaison aide à créer un espace de discussion pour générer des nouvelles idées. Un chef d'équipe est nécessaire pour la direction des discussions. En plus, il guide la procédure d'établissement des idées, et demande la participation de tous les membres. Après la présentation des idées, une discussion est ouverte pour accepter, refuser ou ajouter des idées

- **Techniques d'acquisition automatique de connaissances**

L'acquisition manuelle est difficile à cause de deux raisons. Premièrement, le cogniticien doit rester en contact avec l'expert pendant une période considérable, qui peut être hors les heures de travail. Deuxièmement, l'expert lui-même dans plusieurs cas n'arrive pas à bien présenter ses informations expertes. C'est pourquoi, ces deux difficultés pour l'acquisition des connaissances peuvent être la raison justifiant le recours pour l'automatisation de l'acquisition des connaissances. La figure 5 illustre l'obtention des connaissances d'une manière automatique. La base de données est construite à l'aide des experts ou d'autres systèmes de raisonnement. L'approche d'apprentissage utilise ces données pour construire des nouvelles connaissances. Les connaissances acquises sont transférées vers la base de connaissance pour une utilisation future et la base est appelée base de connaissances dynamique.

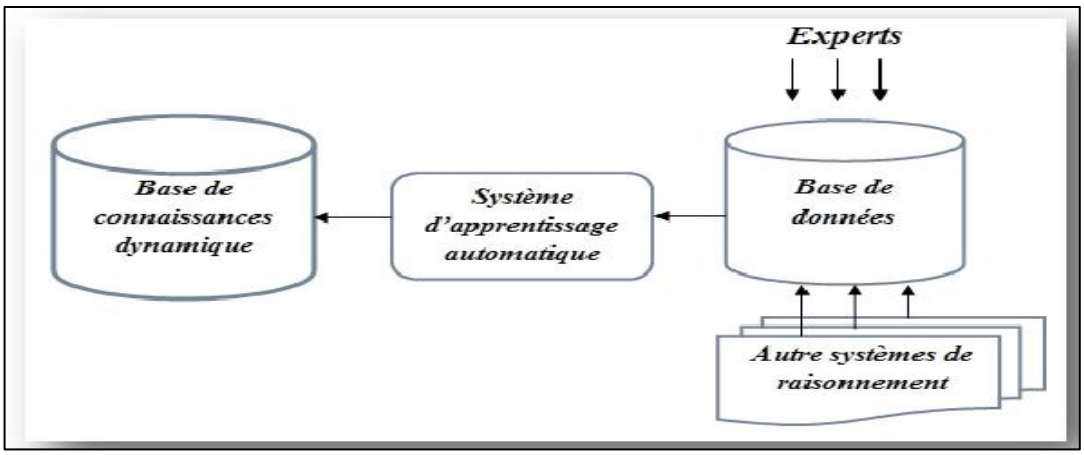


Figure 5. Acquisition automatique des connaissances

L'algorithme d'apprentissage automatique utilisé par excellence pour l'acquisition automatique des connaissances est ID3 [Quinlan, 1979]. Etant données une base de données des exemples, dont les variables sont de type catégorique, l'algorithme utilisant la théorie de l'information de Shannon permet d'induire une liste de règles incluse dans une structure appelée arbre de décision. Afin d'éclaircir le principe utilisé on va reprendre l'exemple proposé par [Alison 1987].

Dans cet exemple, on voudrait automatiser la procédure de la prise de décision d'inscrire ou non des étudiants à des cours au niveau d'un institut**.**

Pour ce faire, on dispose de formulaires de demande d'inscription des candidats relatifs aux deux années précédentes traités par l'expert. Le travail de l'ingénieur de connaissances consiste à construire le tableau d'apprentissage (data set) à partir de ces formulaires et procédure manuelle d'inscription en collaboration avec l'expert. Pour ce faire, l'ingénieur de connaissances doit affronter plus de difficultés : la difficulté de décider sur la taille de data set, c'est à dire le nombre d'exemples que va comprendre le data set, la difficulté de la définition des différents attributs et leur ordre. Quant aux classes (les différentes valeurs de la décision prise après étude d'un formulaire de demande d'inscription), elles sont déjà définies et sont utilisées au même titre que la procédure manuelle. Le data set comprend trois cents

exemples (300) dont les deux tiers utilisés pour l'apprentissage de l'arbre de décision et le un tiers pour le test.

L'application de l'algorithme d'apprentissage ID3 a permis d'induire l'arbre de décision suivant de la figure 6 ci-dessous (les nœuds représentent les attributs, les arcs représentent les valeurs des attributs et les feuilles représente les différentes classes, nous avons trois types de classes :

D : Definite offer, R : Rejected, and P : Provisional offer

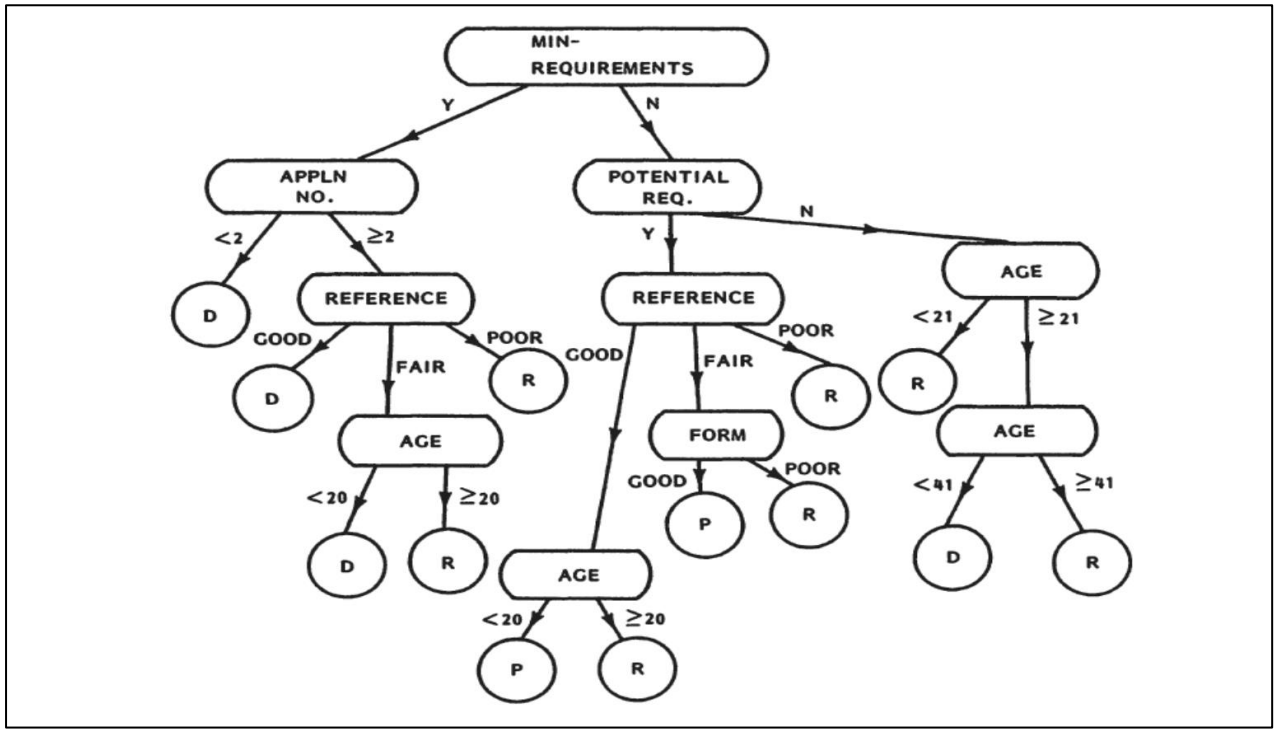


Figure 6 : Arbre de décision induit par l'algorithme ID3 **[Alison 1987].**

Les règles de production à extraire de cet arbre sont citées ci- après (nombre de règles de production égal au nombre de feuilles de l'arbre =13).

IF (Min requirements =Yes & Appln no < 2) THEN D
IF (min requirements = Yes & Appln no >=2 & Reference = good) THEN D
IF (min requirements = Yes & Appln no >=2 & Reference = fair & Age <20) THEN D
IF (min requirements = Yes & Appln no >=2 & reference = fair & age >= 20) THEN R
IF (min requirements = Yes and appln no >=2 & reference = poor) THEN R
IF (min requirements = No & Potential requested =No & Age < 21 THEN R
IF (min requirements = No & Potential requested =No & Age >= 21 & Age < 41) THEN D
IF (min requirements = No and potential requested =No and Age >= 21 and Age >= 41) THEN R
IF (min requirements = No & Potential requested =Yes & reference=good & Age <= 20 THEN P
IF (min requirements = No & Potential requested = Yes & reference = good & Age > 20 THEN R
IF (min requirements = No & Potential requested =Yes & reference=fair & Form = good THEN P
IF (min requirements = No & Potential requested =Yes & reference=fair & Form = poor THEN R
IF (min requirements = No & potential requested =Yes & reference = poor then R.

CHAPITRE 4 : Fonctionnement des Système Experts

4.1 Notion de connaissance et formalisme de représentation de connaissances

La notion de connaissances renvoie aux données expertes ou tout simplement l'expertise attachée à un domaine de discours donné , à une machine, à un équipement, ou à un procédé industriel en général etc... . On note que la connaissance ne contient pas uniquement les données **descriptives** mais aussi celles **opératoires** qui permettent de contrôler le fonctionnement du sujet si besoin est, vers un fonctionnement rationnel (veut dire optimal, efficace etc. …).

Pour pouvoir utiliser ces connaissances par un ordinateur, ces connaissances doivent être représentées en utilisant un formalisme de représentation prédéfini. Un formalisme de représentation est donc un modèle de représentation qui nous permet de coder ou de transcrire les connaissances sur la machine (Voir chapitre 2). Une fois que cette représentation est achevée, on dit que les connaissances sont acquises.

Plusieurs types de représentation de connaissances existent mais pour construire un système expert, on utilise toujours un formalisme bien défini qui s'appelle les règles de production, c'est pour cela que les systèmes experts sont dits aussi des systèmes à base de règles de production.

4. 2. Les règles de production

Une règle de production est une représentation sous la forme de : **IF f1, f2, ...fn THEN fc** . L'ensemble de règles constituant la partie règles d'un système expert est dit une base de règles. La règle précédente est déclenchée ou activée si l'ensemble de faits f1, f2, ...fn sont vraies, C'est à dire appartiennent à la mémoire de travail (base des faits), dans ce cas le fait conclusion fc est produit et ajouté à la base de faits.

4.3 Fonctionnement d'un moteur d'inférence

Comme on l'a défini dans le chapitre précédent, un moteur d'inférence permet d'exécuter les règles déclenchées par les connaissances actuellement stockées dans la mémoire de travail afin de produire de nouveaux faits.

Il existe deux types de moteurs d'inférence : chaînage en avant et chaînage en arrière.

4.3.1 Le chaînage en avant

Principe (démarche déductive)
Entrée : une base de faits F, une base de règle R
Sortie : la base des faits F transformée

Algorithme chainage-en- avant
Répéter
Construire (S, F) /* ajout à S des règles non déjà appliquées */
/ * une règle applicable est une règle non déjà appliquée et dont la prémisse est satisfaite */
Si S ≠ ∅ alors
Choisir (a, R) ; /* choix d'une règle a de S non déjà appliquée */
Appliquer (a) ;
Mise -a-jour(F) ; /* ajouter à F la conclusion de la règle a */
Mise-a-jour (S) / * S = S-{a} */
Finsi
Jusqu'à (S =∅)
Fin

Exemple 6

Soient F = {A, C, D, F} et R = { R1, R2, R3, R4}

R1 : si E, C alors B
R2 : si A, D alors E
R3 : si A, B,F alors G
R4 : si C, D alors F

Donner la nouvelle base de connaissance après application du raisonnement chainage en avant ?

Solution

Itération	Règles applicables	Règle appliquée et marquée	Ancienne base de faits	Nouvelle base de faits
1	S = {R2,R4}	R2	{A,C,D,F}	{A,C,D,F,E}
2	S = { R4,R1}	R4	{A,C,D,F,E}	{A,C,D,F,E}
3	S = { R1}	R1	A,C,D,F,E}	{A,C,D,F,E,B}
4	S={R 3}	R3	A,C,D,F,E,B}	{A,C,D,F,E,B,G}
5	S=∅			
Arrêt				

4.3.2 Le chainage en arrière

Principe: démarche hypothético-déductive
Entrées :
-une base de faits F,
-une base de règles R,
-un ensemble P de faits à prouver. /* P est le but à prouver */
Sortie :

-la base de faits transformée F.

Algorithme chaînage-en-arrière
Début
GOAL= vrai ;
Tant que P ≠ ∅ et GOAL =Vrai
f = Extraire (P) ; /* extraire un fait f de P */

Si f ∉ F alors
S = Regles (f) /* ensemble des règles dont la partie conclusion est f */
OK=false ;
Tant que S ≠ ∅ et OK= false
r = Choisir (S) /* choix d'une règle de S */
Si applicable(r) alors
Appliquer(r) ; /* prouver tous les faits en prémisse de r */
Effacer (f, P) ; /* effacer f de P*/
Ajouter (f,F) ;
OK=vrai ;
Sinon /* r est non applicable */
Ajouter (prémisses-NAF(a),P) ; /* ajouter à P les prémisses de r n'appartenant pas à F */
Supprimer (r, S) ;
Finsi
Fin Tant que
Si Ok=false alors /* aucune règle qui conclut sur f*/
GOAL= false ;
Finsi
Sinon
Effacer (f, P) ;
Finsi
Fin Tant que
Si GOAL alors Ecrire ('Le but : ', P, 'est prouvé) ;
Sinon ('Le but : ', P, 'est non prouvé') ;
Fin

Exemple 7
Soit la base de connaissances suivante
F= {A, C, D, N}
R1 : si E,C alors B
R2 : si A,D alors E
R 3 : si A,B,N alors G
R4 : si C,D alors N

Peut – on prouver le but : P= {G, N} ? Donnez l'arbre ET-OU correspondant.

Solution

On tire le premier fait G de P à prouver. Dans la règle R3, la réalisation de G exige que les faits A, B et N soient valides. Or A et N sont déjà des faits donc il reste B à réaliser. Dans R1, pour réaliser B, il faut réaliser les faits E et C, or C est un fait donc E à réaliser. Dans R2, pour réaliser E, il faut réaliser A et D, qui sont déjà des faits, donc E est un fait. De R1 en déduit que B est aussi un fait. Finalement, de R3, on déduit que G est un fait. Le deuxième fait N de P à prouver est déjà un fait. Donc le but P est prouvé.

-L'arbre ET -OU

La figure 7 sur la page suivante montre l'arbre And- Or où le nœud racine ''P'' est le but principal a démonter, les nœuds intermédiaires représentent les buts secondaires et les feuilles représentent les faits de la base des faits.

En commençant par la racine ''P'', la branche gauche ''G'' est composée de ramifications '' AND'' et les feuilles de cette branche sont toutes des faits donc le but G est vérifié. Quant à la branche droite ''N'' de l'arbre est composée de deux feuilles ''C'' et ''F'' liées par la conjonction ''OR'' ; Or la feuille ''C'' est un fait et donc le but ''N'' est aussi vérifié. G et N sont vérifiés donc P est vérifié.

4.3.3 Le chaînage mixte

Le principe de cette stratégie de raisonnement est de fonctionner en premier lieu en marche arrière. Dès qu'un nouveau fait est connu, alors fonctionner en marche avant puis poursuivre à fonctionner en marche arrière.

Exemple 8

Effectuez un raisonnement mixte sur la base de connaissances suivante et donner la nouvelle base de faits produite.

Base des faits F = {A, C, D, N}
Base des règles R = {R1, R2, R3, R4}

R1 : si E, C alors B
R2 : si A, D alors E
R3 : si A, B, N alors G
R4 : si C, D alors N

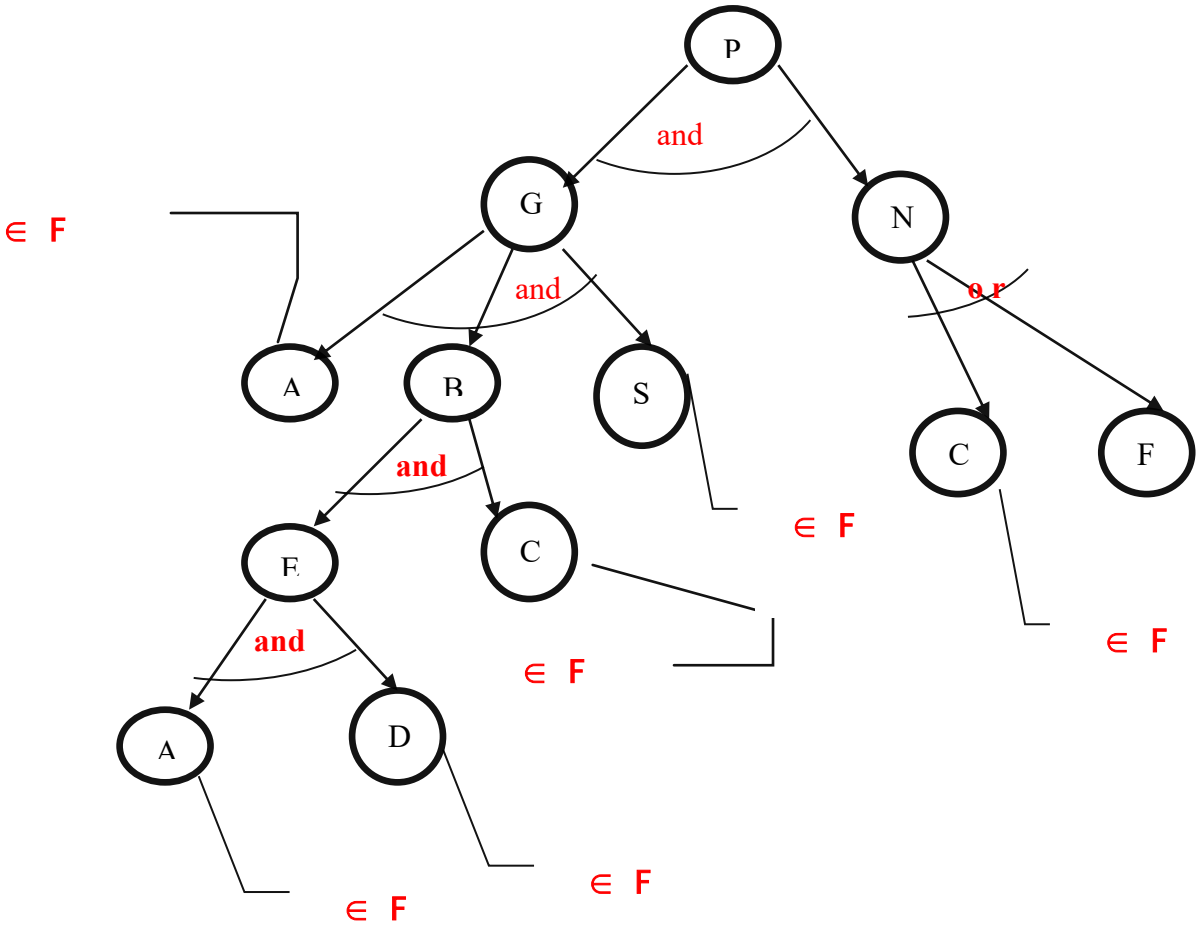


Figure 7: L'arbre ET- OU

Solution

Marche en arrière :
R1 : pour prouver B, il faut prouver E et C, or C est un fait.
R2 : pour prouver E, il faut prouver A et D qui sont des faits alors E est un fait
Nouvelle base de faits F= {A, C, D, N, E}.

Marche en avant :
R1 : on en déduit B donc nouvelle base de faits F = {A, C, D, N, E, B}.

Marche en arrière :
R3 : pour prouver G, il faut prouver A, B et N qui sont tous des faits , donc G est prouvé.
Nouvelle base de faits F={A,C, D, N, E, B, G} …… Arrêt.

Chapitre 5 : Approche de Développement des Systèmes Experts

5.1 Processus de développement d'un système expert

Le développement d'un système expert passe par un certain nombre d''étapes et vise la réalisation d'une architecture standard d'un système expert comme vue au chapitre 2. L'ensemble de ces étapes est communément appelé le cycle de vie d'un système expert.

Étape 1 : Acquisition des connaissances expertes

Cette étape est réalisée par un ou plusieurs cogniticiens en coordination avec un ou plusieurs experts dans le domaine de discours étudié. Elle consiste en premier lieu, à la collecte de toutes les données expertes nécessaires à la réalisation du cahier de charges établi précédemment avec le client. Le groupe des cogniticiens utilise ensuite un formalisme, en l'occurrence, les règles de production, pour traduire ces connaissances dans un format manipulable par le moteur d'inférence. Le format de connaissances obtenu s'appelle la base de connaissances.(BC) ou (KB en Anglais)

Étape 2 : Développement des modules du système expert

Ici on engage une équipe de développeurs permettant le développement et le test des différents modules du système expert et de tisser les liens entre eux, le cas échéant. Ces modules sont :
module de raisonnement (MR) ou module d'inférence, le module d'explication du raisonnement(MER) le module d'édition de la base de connaissance (MEBC) et finalement l'interface utilisateur.(IU)

Étape 3 : Validation du système expert

Une fois développé, le système expert est soumis à une étape de validation pour vérifier si les conditions et souhaits du client exprimés dans le cahier de charges sont respectés ou non. Si oui, tant mieux, sinon un retour en arrière vers les étapes précédentes est obligatoire pour apporter les corrections qui s'imposent.

Étape 4 : Maintenance du système expert

Tout système en fonctionnement pérenne et de nature évolutif doit subir dans son cycle de vie des opérations de maintenances périodiques ou occasionnelles. Ces opérations de maintenance peuvent être de nature curative qui visent à corriger une insuffisance ou préventive visant à améliorer la performance du système.

5.2 Exemples de systèmes experts

5.2.1 DENDRALL

Parmi les plus importants des systèmes experts est le DENDRALL, développé à l'université de STANDFORD en fin des années soixante. DENDRALL permet d'inférer la structure moléculaire des

éléments chimiques en se donnant leurs formules chimiques et leurs masses spectrographiques. Ce produit a connu un grand succès et a été utilisé dans la plus part des laboratoires de chimie à travers le monde.

5.2.3 PROSPECTOR

Le centre d'intelligence artificielle de SRI a développé PROSPECTOR pour l'US Geological Survey afin d'aider les géologues dans l'exploration minérale. Le système, l'un des premiers systèmes experts informatisés au monde, a tenté de représenter les connaissances et le processus de raisonnement des experts en géologie. Le système a été testé dans le processus de connaissance et de raisonnement des géologues. Son utilisation principale est prévue par l'exploration géologique dans les premières étapes de l'investigation d'éventuels sites de forage. Prospector prédit l'existence d'un gisement de molybdène dans l'état de Washington.

5.2.3 MYCIN

Développé par une équipe de chercheurs en IA a l'université de Standford pour aider les spécialistes pour diagnostiquer les maladies de sang. Le module de raisonnement de ce système expert permet de raisonner sur l'information incertaine qui intrinsèque a l'information médicale. Plus de détails sur ce système expert seront données dans le chapitre suivant.

5.2.4 Un Système Expert « Pédagogique »

Pour mieux éclaircir la notion de système expert dont le but bien entendu est d'imiter la capacité décisionnelle de l'expert humain, on va traiter un exemple pédagogique simulant une enquête policière visant à déterminer le(s) voleur(s) de gâteaux.

- **Enoncé de cas**

Yasmina a préparé des gâteaux. Elle les a rangés dans un placard qu'elle a fermé à clé. A l'heure du café, les gâteaux sont disparus. Grâce à une enquête, les indices et les faits suivants sont rapportés :

1. Mounir et le demi-frère de Jihan
2. Yasmina est la maman de jihan
3. Karim est l'oncle de Mounir
4. Soufiane, Raouf , Houssem et Meriem et sont des voisins directs
5. Jihan ainsi que les membres de sa famille possèdent la clé du placard
6. Raouf a des difficultés financières et possèdent des canaris
7. Mounir doit suivre un régime et est blessé à la main
8. On a trouvé des miettes de gâteaux sur le bureau de Karim
9. Houssem est malin, riche et mince
10. Meriem passe son temps libre à tester des recettes
11. La porte du placard a été retrouvée forcée

Après mûre réflexion, on a émis les hypothèses suivantes :

1- Pour voler des gâteaux, il faut en avoir besoin et avoir la possibilité d'y accéder.
2- On peut avoir besoin des gâteaux soit parce que on est gourmand, soit parce que on veut les donner à quelqu'un.
3- Pour accéder aux gâteaux, il faut soit avoir la clé du placard, soit le forcer.
4- La personne ayant forcé le placard a de fortes chances d'être blessée.
5- On peut avoir besoin de gâteaux si on doit nourrir quelqu'un et qu'on n'a pas l'argent pour en acheter.
6- Une personne suivant un régime ou aimant cuisiner peut être gourmande
7- On fait des miettes en mangeant les gâteaux.
8- Si on possède des animaux, il faudrait les nourrir.

- Travail demandé :

a- Représenter ces connaissances sous forme de faits et de règles de production.
b- Utiliser cette base de connaissances pour répondre à la question : qui a ou qui ont volé les gâteaux ?

- **Solution**

a) Représentation des données du problème sous forme de faits et règles de production

Les règles :

SI besoin(X,gateaux) ET avoir-acces(X,gateaux) ALORS voler(X,gateaux)
SI besoin(X,gateaux) ET forcer(X,porte-placard) ALORS voler (X,gateaux)
SI gourmand(X) OU doit-nourrir((X,Y) ALORS besoin(X,gateaux)
SI possede (X,cle) ALORS avoir-acces(X,gateaux)
SI forcee(porte-placard) ET blesse(X) ALORS forcer(X, porte-placard)
SI possede (X,Y) ET animal(Y) alors doit-nourrir (X,Y)
SI doit-nourrir(X,Y) ET besoin(X,argent)) ALORS besoin(X,gateaux)
SI suivre-regime(X) OU aime-cuisiner(X) ALORS gourmand(X)
SI manger(X,gateaux) ALORS faire-miettes(X,gateaux)

Les faits :

voisins_directs(Houssem, Raouaf,Meriem, Soufiene)
mere(yasmina, jihan)
demi-frere (Mounir,Djihan)
oncle(karim, Mounir)
possede(jihan, cle)
possede (Mounir, cle)
possede(Yasmina,cle)
besoin(Raouf, argent)
possede(Raouf, canaris)

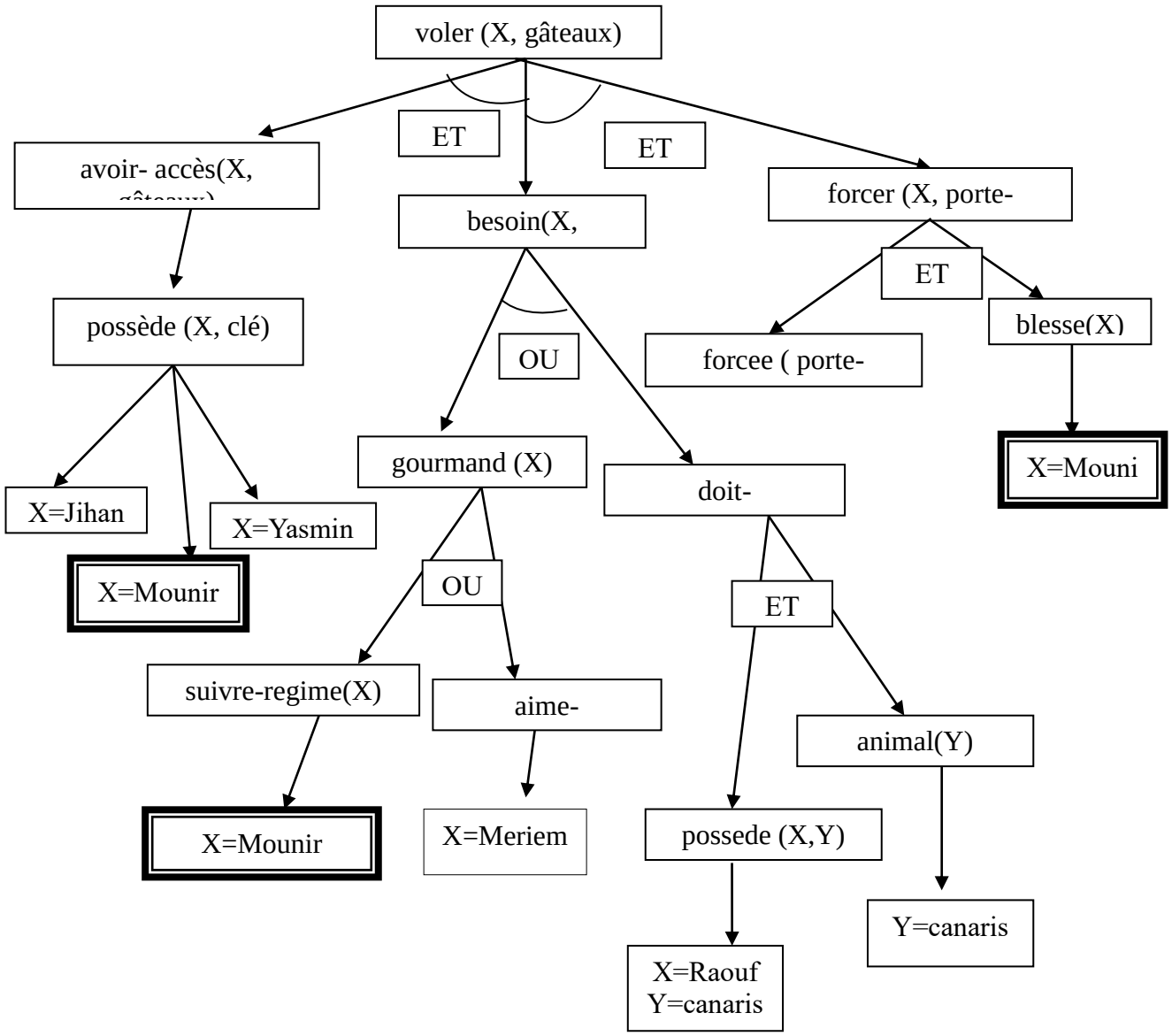


Figure 8 : L'arbre ET-OU de l'enquête policière

animal(canaris)
suivre-regime(Mounir)
blesse(Mounir)
faire-miettes(Karim,gateaux)
malin (Houssem)
riche(Houssem)
mince(Houssem)
aime-cuisiner(Meriem)
forcee(porte-placard)
besoin(Raouf,argent)

b) Réponse à la question : qui a (ont) volé les gâteaux ?

Le raisonnement sur la base des connaissances (règles et faits) par chaînage en arrière revient à construire l'arbre ET-OU de la figure 8 ci-dessus. Elle montre que celui qui a volé les gâteaux c'est bien Mounir.

5.3 Classification des systèmes experts : On en distingue trois classes :

Classe 1 : appelé d'ordre 0

Fondée sur le calcul propositionnel. Les faits sont à valeurs booléennes.
Exemple de règle : SI Agé_de_plus_de_18_ans ALORS Majeur

Classe 2 : appelée d'ordre 0+

Ici les faits peuvent être à valeurs réelles ou symboliques
Exemple de règle : SI Age $\geqslant$ 18 ALORS statut=Majeur

Classe 3 : appelée d'ordre 1

Basée sur le calcul des prédicats et l'utilisation des variables et de l'unification
Exemple de règle : SI Age (X, Age) ET Age $\geqslant$ 18 ALORS X= Majeur

5.4 Avantages des systèmes experts

1. Disponibilité et fiabilité accrues. L'expertise est accessible sur n'importe quel matériel informatique et le système répond toujours à temps.
2. Expertise multiple : Plusieurs systèmes experts peuvent être exécutés simultanément pour résoudre un problème et acquérir un niveau d'expertise supérieur à celui d'un expert humain.
3. Explication : Les systèmes experts décrivent toujours comment le problème a été résolu.
4. Réponse rapide : Les systèmes experts sont rapides et capables de résoudre un problème en temps réel.
5. Coût réduit : Le coût de l'expertise pour chaque utilisateur est considérablement réduit.

5.5 Limites des systèmes experts

1. Les systèmes experts nécessitent des ingénieurs du savoir pour saisir les données, l'acquisition de données est très difficile.
2. Le système expert peut choisir la méthode la plus inappropriée pour résoudre un problème particulier.
3. Les problèmes d'éthique dans l'utilisation de toute forme d'IA sont très pertinents à l'heure actuelle.
4. C'est un monde fermé avec des connaissances spécifiques, dans lequel il n'y a pas de perception profonde des concepts et de leurs interrelations jusqu'à ce qu'un expert les fournisse.
5. Cette variante de systèmes experts est classique et ne peut modéliser les connaissances entachées d'incertitude et de floues, ce qui a donné naissance à des systèmes hybrides [Bouzenita 2012].

Chapitre 6 : Les Systèmes Experts Incertains

6.1 Introduction

L'information manipulée par les systèmes experts standards est supposée certaine mais dans le monde réel elle est généralement entachée d'incertitude. Un exemple typique de l'information incertaine est l'information échangée dans le domaine médical. Les systèmes experts qui prennent en charge cette incertitude et la manipule s'appellent les systèmes experts incertains. Dans ce qui suit, nous allons présenter un des systèmes experts incertains le plus connu, qui s'appelle MYCIN. Mais avant çà, nous allons faire un survol des différentes théories permettant de prendre en charge l'information incertaine.

6.2 Théories de traitement de l'incertitude

6.2.1 Théorie de la probabilité

Une probabilité P(E) exprime la chance de production d'un évènement E. par exemple on dit ma chance de réussite au test de conduite est de 0.80 ou 80 % veut dire que d'après mon historique d'apprentissage de la conduite, j'ai réussi 8 cas parmi 10 cas effectués au total. Cette probabilité peut être exprimée autrement en disant que mon degré de confiance de réussir au test de conduite est de 0.8. La relation utilisée dans le raisonnement basé sur la théorie de probabilité est la relation de Bayes exprimée par :

$P(A/B) = (P(B|A) * P(A)) / P(B)$ avec : $P(B) = P(B/A) * P(A) + P(B |\neg A) * P(\neg A)$.

Pour illustrer l'utilisation de cette relation, nous allons traiter l'exemple ci-dessous.

Exemple 9

soient les évènements E1 , E2 et E3 supposés indépendants et les hypothèse possibles H1, H2 et H3 supposées mutuellement exclusives et exhaustives , exprimées comme suit :

E1 : ' Le micro-ordinateur s'arrête soudainement juste après dix minutes de fonctionnement''
E2 : ' Le miro - ordinateur s'arrête et redémarre tout seul sans arrêt après sa première mise en marche'
E3 : ' Le miro - ordinateur ne démarre pas et émet deux (2) beeps sonores'

H1 : 'Le ventilateur de refroidissement du miro ordinateur est défectueux ou poussiéreux'
H2 : ' Le micro-ordinateur est infecté par un virus'
H3 : ' Les barrettes mémoires sont défectueuses ou mal placées '

Un expert en réparation du micro - ordinateurs nous a fournis les données suivantes figurant dans le tableau 1 ci-dessous.

	Hypothèses (Hi)		
Probabilités	i=1	i=2	i=3
P (Hi)	0.5	0.3	0.2
P (E1/Hi)	0.8	0.7	0.1
P (E2/Hi)	0.2	0.7	0.0
P (E3/Hi)	0.0	0.0	0.8

Tableau 1. Données expertes.

a) Représenter ces connaissances par un réseau bayésien
b) Si on suppose que la panne observée de notre micro-ordinateur est E1, quelle serait l'origine de la panne la plus probable parmi les hypothèses citées. (Indication : Calculer P (H1/E1), P (H2/E1) et P (H3/E1).

Solution

a) Un réseau bayésien est un graphe orienté acyclique dont les nœuds représentent les variables aléatoires définissant le problème et dont les arcs représentent les relations causales entre ces variables. Ces relations causales sont étiquetées par des probabilités.

b) Selon la relation de Bayes donnant la probabilité conditionnelle, nous avons :

P(H1/E1) = P(E1/H1).*P(H1) / (P(E1/H1)* P(H1) +P(E1/H2)* P(H2) + P(E1/H3* P(H3))
= 0.8*.0. 5/ (0.8*0.5+0.7*0.3+0.1*0.2) = 0.4/(0.40+ 0.21+0.02)= 0.40/0.63= 0.634

P(H2/E1) = P(E1/H2).*P(H2) / (P(E1/H1)* P(H1) +P(E1/H2)* P(H2) + P(E1/H3* P(H3))
= 0.7*0.3/ (0.7*0.3+0.7*(0.4+0.3))= 0.21/ (0.40+ 0.21+0.02)= 0.21/0.63=0.333

P(H3/E1)= P(E1/H3).*P(H3) / (P(E1/H1)* P(H1) +P(E1/H2)* P(H2) + P(E1/H3* P(H3))
= 0.1*0.2 / (0.40+ 0.21+0.02) = 0.02/ *0.63 = 0.02*/0.63= 0.033

En termes de probabilités, P(H1/E1) est la plus grande donc l'origine de la panne E1, en termes de probabilités, est bien H1.

- **Avantages et inconvénients**

Le modèle de représentation et de raisonnement bayésien est un modèle basé sur le calcul des probabilités, donc formel. Il permet de raisonner en l'absence d'informations certaines. En plus, le réseau bayésien est un graphe donc intuitif et facile à lire.
Par contre, ses inconvénients peuvent être résumés par le fait que les variables aléatoires sont supposées indépendantes et que les hypothèses sont aussi supposées exhaustives et mutuellement exclusives, ce qui est contrariant et restreint l'application de ce modèle aux problèmes vérifiant uniquement ces contraintes. Autre limite de ce modèle c'est que les données de départ dépendent des personnes expertes

dans le domaine traité et donc non facilement disponibles. De plus, le graphe engendré qui est le réseau bayésien devient de taille importante lorsque les variables définissant le problème sont importantes ce qui influe sur la performance du système utilisant ce modèle.

6.2.2 La logique de défaut

Est une logique non monotone qui permet de raisonner en l'absence de connaissances complètes et incertaines sur le domaine de discours. Dans cette logique proposée par Reiter [Reiter 1980], on définit un ensemble de règles d'inférences appelées défauts (D) qui permet de raisonner sur une base de connaissances (A), exprimée dans un langage de prédicats de 1er ordre. Une théorie de défaut est donc définie par le couple (A, D) où A est un ensemble de connaissances non complétes et réfutables et D est un ensemble de règles de défaut où chaque défaut est de la forme a : b / c ; qui se lit « si a est vraie et si b est cohérente avec ce qui est connu déjà alors inférer c ». a, b et c sont des formules bien formées et nommées respectivement, la condition, la justification et la conséquence du défaut. Bref, la logique de défaut est un cadre nous permettant de raisonner à défaut d'informations.

Exemple 10

Soit une théorie de défaut définie par : (A,D) ou A ={ R(b),S(b)}, D = { R(x) : F(x) /F(x)}. Calculer l'extension E de A ?

Solution

A contient deux faits et D contient une seule règle de défaut. L'application de cette règle de défaut génère une extension E de la théorie de défaut comme suit : E = A U F(b) = {R(b),S(b),F(b)}.

Avantages et inconvénients

La logique de défaut permet le raisonnement en présence d'informations complètes inconsistantes et incertaines. Aussi, la logique de défaut permet le raisonnement non monotone c'est-à-dire que des connaissances qui ont été ajoutées à la base de connaissances peuvent être rétractées avec l'arrivée de nouvelles informations. En revanche, la logique de défaut suppose que les règles de défaut soient indépendantes les uns des autres, chose qui n'est pas toujours vraie dans les cas pratiques. Un autre inconvénient est les calculs à effectuer pour combiner l'application de l'ensemble des règles de défaut, qui pourraient être intensifs surtout lorsque le nombre de règles de défaut est important. [Ri 07]

5.2.3 Théorie de l'incertitude de Buchanan & Shortliffe

Selon les deux chercheurs [Buchanan & Shortliffe & 1975], le facteur de certitude attaché à une information incertaine est gradué comme montré dans le tableau 2 ci – dessous.

Certitude Facteur (CF)	Terme utilisé
-1	Certainement NON
-0,8	Presque certain que NON
-0.6	Probablement NON

- 0.4	Peut-être que Non
Entre -0.2 et +0.2	Inconnu
+0.4	Peut-être que Oui
+0..6	Probablement Oui
+0,8	Presque certain que OUI
+1	Certainement Oui

Tableau 2 : Graduation de l'information incertaine selon Buchanan & Shortliffe

- Calcul des coefficients de certitude.

Règle avec une seule prémisse

R : Si E ALORS H CF(R)

CF(H,E)=CF(E)*CF(R)

Exemple 11 : *Si (présence de nuages sombres) (0.5) ALORS (il va pleuvoir) CF(R) = 0.8*

*CF(H,E) = 0.5*0,8 = 0.4*

Si plusieurs prémisses en conjonction

R : Si E1 ^ E2 ^ … ^ En ALORS H CF(R)
CF(H,E1 ^ E2^… ^ En) = min{CF(Ei)}*CF(R)

Exemple 12 :
Si (présence de nuages sombres) (0.5) ET (présence du vent) (0.3) ALORS (il va pleuvoir) CF(R) = 0.8

*CF(H,E) = min(0.5, 0.3) *0,8 = 0.3*0.8=0.24*

Si plusieurs prémisses en disjonction

R : Si E1 V E2 V … V En ALORS H CF(R)
CF(H,E1 V … V En)=max{CF(Ei)}*CF(R)

Exemple 13:
Si (présence de nuages sombres) (0.5) OU (présence du vent) (0.5) ALORS (il va pleuvoir) CF(R) = 0.8

*CF(H,E) = max(0.5, 0.5) *0,8 = 0.5*0.8=0.40*

Propagation avec des règles à conclusions similaires :

Si E1 alors H CF(R1)
Si E2 alors H CF(R2)

Cas 1: Cfres(H) = CF(H,E1)+CF(H,E2)-CF(H,E1)*CF(H,E2), si CF(E1) et CF(E2) positifs.
Cas 2: CF(H,E1) + CF(H,E2)-CF(H,E1)*CF(H,E2), si CF(E1) et CF(E2) négatifs.
Cas 3 : (CF(H,E1)+CF(H,E2)) / (1-min{|CF(H,E1)|,|CF(H,E2)|}) si CF(E1) * CF(E2) < 0 .

Exemple 14

Si E1 alors H CF(R1)
Si E2 alors H CF(R2)

Avec : CF(R1) = CF(R2) = 0.8

Si CF(E1)=CF(E2)=1, on obtient Cfres(H)=0.8+0.8-0.8*0.8 = 0.96
Si CF(E1)=1, CF(E2)=-1, on obtient Cfres(H)= (1*0.8+0.8*(-1)) / (1- min(abs(0.8),abs(-0.8)) = 0
Si CF(E1)= - 0.8, CF(E2)= - 0.6, on obtient Cfres(H)= - 0.8*0.8+ 0.8*(-0.6) –(-0.64*(-0.48))
= (-0.64-0.48) - (0.64*0.48) = - 0.59

Un des systèmes experts les plus célèbres utilisant la therorie de Shortliffe et Buchanan s'appelle MYCIN, développé à l'université de Standford par une équipe de développeurs dirigée par Buchanan & Shortliffe. MYCIN assiste les médecins dans le diagnostic des maladies du sang.

Exemple 15

Cet exemple permet d'illustrer le fonctionnement des systèmes experts incertains fonctionnant avec la théorie d'incertitude de Shortliffe et Buchanan tel MYCIN.

Soient les règles incertaines suivantes ;

R1 :	SI Température ≥ 39 ALORS Infection	CF(0,8)
R2 :	SI Maux- Tête & Vomissement ALORS Coup de soleil	CF(0,6)
R3 :	SI Douleur abdominale & Diarrhée ALORS infection	CF (0.7)
R4 :	SI GLYCÉMIE A JEUN .> 2 mg/l ALORS Personne -Diabétique	CF(0.9)
R5 :	SI Personne -Obèse Alors Personne -Diabétique	CF(0.4)
R6	SI Personne -Obèse Alors Personne-Hypertendue	CF(0.6)

Et soient les faits incertains suivants :

F1 : Température >=39	CF(0.9)
F2 : Personne -Obèse	CF(0.8)
F3 GLYCÉMIE A JEUN.> 2 mg/l	CF(0.7)
F4 : Douleur abdominale	CF(0.9)
F5 : Diarrhée	CF(0..9)

Nous allons Inférer cette base de connaissances et produire la nouvelle base de faits.

No	Faits Courants	Règles invoquées	Nouveaux faits
01	{F1,F2,F3,F4,F5}	{R4, R1, R3,R6,R5}	{F6,F7,F8,F9}
02	{ F1,F2,F3,F4,F5, F6,F7,F8,F9}	Arrêt	

On ordonne les règles d'une maniéré croissante selon la valeur du CF.

- Application de la règle R4 : donne

F6 : Personne- Diabétique CF (0.9 x 0.7) = CF (0.56)

- Application de la règle R1 et R3 qui concluent sur la même conclusion avec R3 en conjonction de faits :

min (CF(F4),CF(F5)) = 0.9 >0 et CF(F1) =0.9 >0 . On applique donc le cas1 :

L'application de R1 et R3 donne :

F7: Infection CF (0.9 +0.9 - 0,9x0.9) = CF (1,18 -0.81) = CF(0.99)

Application de la règle R6 donne :

F8 : Personne Hypertendue CF (0.8 x 0.6) = CF (0.48)

Application de la règle R5 donne :

F9 : Personne Diabétique CF (0.8 x 0.4) = CF(0.32).

Remarque : Puisque nous avons généré un fait nouveau avec deux coefficients de certitude différents :

F6 : Personne Diabétique CF(0.56) et F9 : Personne Diabétique CF(0.32)

Donc on aura : F6 : Personne Diabétique CF(min (0.56,0.32) = CF(0.32)

La base de faits finale est donc :

F1 : Température >=39 CF(0.9)
F2 : Personne -Obèse CF(0.8)
F3 GLYCÉMIE A JEUN .> 2 mg/l CF(0.7)
F4 : Douleur abdominale CF(0.9)
F5 : Diarrhée CF(0..9)
F6 : Personne Diabétique CF(0.32)
F7 : Infection CF(0.99)
F8 : Personne Hypertendue CF (0.48)

CHAPITRE 7 : Les Systèmes Experts Flous

7.1 Introduction

Les informations échangées par les êtres humains sont souvent entachées de modalités floues telles :

- Les caisses de l'état sont presque vides, il est temps d'instaurer des mesures d'austérité sévères.
- Mohamed est fort en matières scientifiques, il va certainement réussir son Bac.
- Samira a pris beaucoup de poids, son médecin lui a prescris un régime alimentaire rigoureux.

Dans les assertions précédentes, les mots soulignés sont tous flous et mesurent un degré d'appartenance à un ensemble.
Si le domaine d'expertise comprend des informations floues, cas souvent rencontré, il est impératif que lors de la conception d'un système expert de tel domaine de prendre en considération ce type d'informations : autrement dit, savoir les représenter et savoir les manipuler. C'est d'ailleurs l'objet de la logique floue mise en œuvre par le mathématicien Lotfi Zadeh [Zadeh 1965][Zadeh 1978].
Le raisonnement flou est une manière de raisonner en l'absence d'informations précises. Lotfi Zadeh le définit par « raisonnement par les mots ».

7.2 Notions générales sur la logique floue

Tandis que la valeur de vérité en logique classique admet seulement deux valeurs vrai ou faux, la valeur de vérité en logique floue connait des valeurs de vérité graduelles qui se résument dans le tableau suivant (Voir Tableau 3.)

Valeur de vérité	Signification
0.0	Absolument faux
0.2	En majorité faux
0.4	Quelque peu faux
0.6	Quelque peu vrai
0.8	En majorité vrai
1.00	Absolument vrai

Tableau 3 : Valeurs de vérité en logique floue

La valeur de vérité en logique floue représente un degré d'appartenance à une étiquette d'une variable floue (une variable floue possède plusieurs étiquettes ou valeurs floues). Par exemple la variable floue température possède les valeurs floues : FROID, TIEDE, CHAUD. L'exemple suivant illustre cette notion.

Soient les mesures de températures suivantes et leurs degrés d'appartenance aux étiquettes (valeurs floues) CHAUD, TIEDE et FROID, (Voir Figure 9)

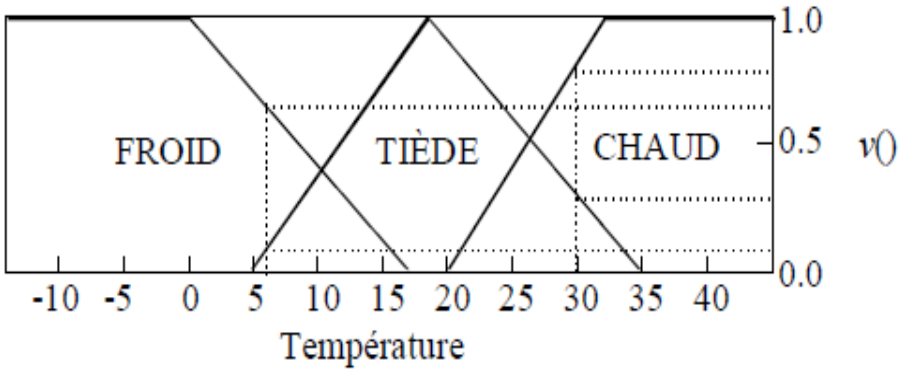


Figure 9 : Fonction caractéristique de la variable floue Température

La fonction qui donne le degré d'appartenance d'un sous ensemble de valeurs est appelée fonction caractéristique. Par exemple, pour l'étiquette CHAUD (Voir figure, les valeurs <=20 C^o n'y appartiennent pas (degré d'appartenance nul), tandis que Les températures >= 33 C^o y appartiennent surement (degré d'appartenance égal à 1) et le reste des températures entre 20 C^o et 33 C^o y appartiennent avec des degrés entre 0 et 1.

Une valeur de température mesurée peut appartenir à plusieurs étiquettes mais à des degrés de vérités différents, par exemple : pour les températures 6 C^0 et et 30 $C^{0,}$, nous avons :

$$V_{FROID}(6)=0.6 ; V_{TIEDE}(6)=0.05 ; V_{CHAUD}(6)=0.0 ;$$

$$V_{FROID}(30)=0.0 ; V_{TIEDE}(30)=0.2 ; V_{CHAUD}(30)=0.8 ;$$

Des opérations sur les sous-ensembles flous (étiquettes de variables floues) permettant de définir la valeur de la fonction caractéristique de l'Union de deux ensembles flous, de l'intersection de deux ensembles flous et du complément d'un sous ensemble flou, comme illustré par le tableau 4 suivant :

Le sous ensemble flou	**Degré de vérité = Degré d'appartenance à l'ensemble X**
X	V_X
Y	V_Y
X ∩ Y	$V_{(X\ Et\ Y)}$= Min (Vx , Vy)
X U Y	$V_{(X\ OU\ Y)}$= Max (V_X , V_Y)
X^{-1} (Comp. De X)	V_{X}^{-1} = 1- V_X

Tableau 4: Opérations sur les sous-ensembles floues

7.3 Les règles floues

Une règle floue est de la forme générale suivante :

SI var_1 est $etiq_1$ **ET/OU** var_2 est $etiq_2$…. **ET/OU** var_n est $etiq_n$ **ALORS** var_m est $etiq_m$ **ET/OU** var_k est $etiq_k$ ….**ET/OU** var_r est $etiq_r$

Avec etiq $_i$: des étiquettes de variables floues Var_i

Exemple 16

Si température est élevée ET pluviométrie est grande alors climat tropical.
Si température est glaciale ET vitesse-vent est grande alors climat polaire.

7.4 L'inférence floue

Pour construire un système fonctionnant en mode de raisonnement flou, on suit les étapes suivantes [Ri 06] :

- Définir les variables d'entrée et de sortie
  (Exemple d'entrées : température ; Exemple de sorties : vitesse ventilateur
- Définir les étiquettes d'entrée et de sortie
  (Exemple d'étiquettes : froid, tiède, chaud pour température Nulle, grande, petite pour la vitesse du ventilateur)
- Définition des fonctions d'appartenance
- Définir les règles d'inférence sur les étiquettes
  (Exemple de règle : Si température est chaud Alors vitesse ventilateur grande)
- Choisir une méthode de conversion flou-précis pour les sorties,

(Exemple de méthode : centre de gravité du trapèze, ou autre méthode)

- Choisir une technique de composition de règles,
  (Exemple de technique : moyenne pondérée)
- Coder le système en utilisant par exemple un outil spécialisé comme Fuzzy Logic Toolbox sur MATLAB .

Un système de décision fonctionnant en mécanisme d'inférence floue suit les étapes suivantes :

Etape 1 : Fuzzification des entrées
Etape2 : Evaluation des règles floues
Etape3 : Agrégation des conclusions des règles
Etape4 : Deffuzzification.

Nous allons expliquer ces étapes à travers l'exemple ci-après, traité sur l'environnement MATLAB. L'exemple modélise un système d'inférence floue qui consiste à la prédiction de La forme du corps en fonction de la taille et du poids du corps. Les variables d'entrée sont représentées par les variables floues **poids** et **taille** et la sortie est représentée par la variable floue **forme.**

Ce système d'inférence floue « monfis » est représenté dans la figure 10 ci-dessous, qui définit les différents paramètres du système à savoir : la méthode de composition and, or implication, l'agrégation des règles et la manière d'appliquer la defuzzification.

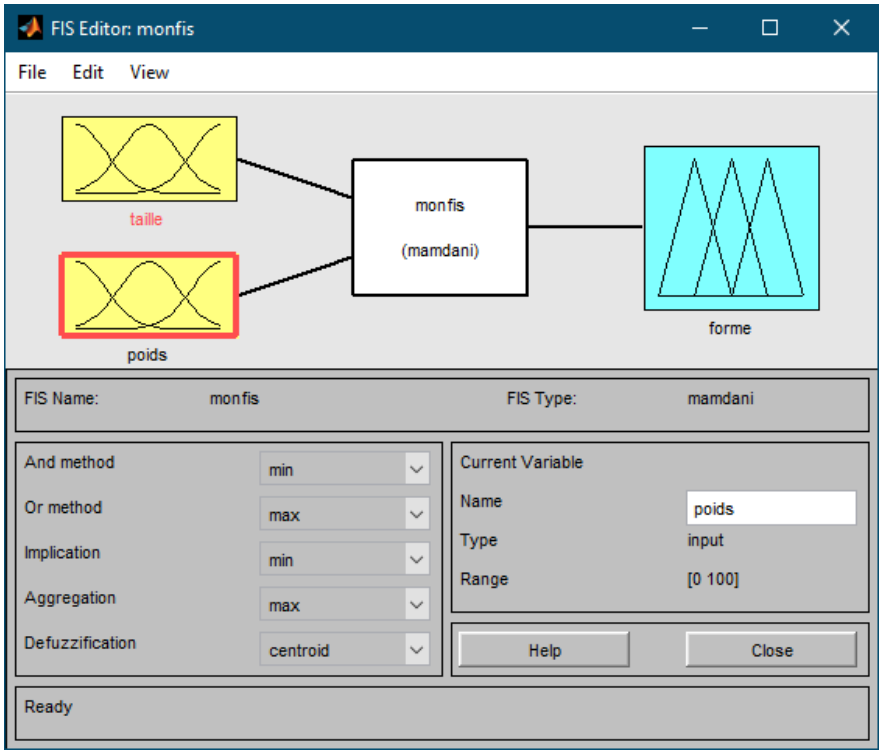


Figure 10. Le système d'inférence floue « monfis »

Les étiquettes de la variable d'entrée ***taille*** *(petite, moyenne et grande)* sont illustrées par les figures 11, 12 et 13 tandis que les étiquettes de la variable ***poids*** *(léger, normal, lourd)* sont illustrées par les figures 14, 15 et 16 ci-dessous

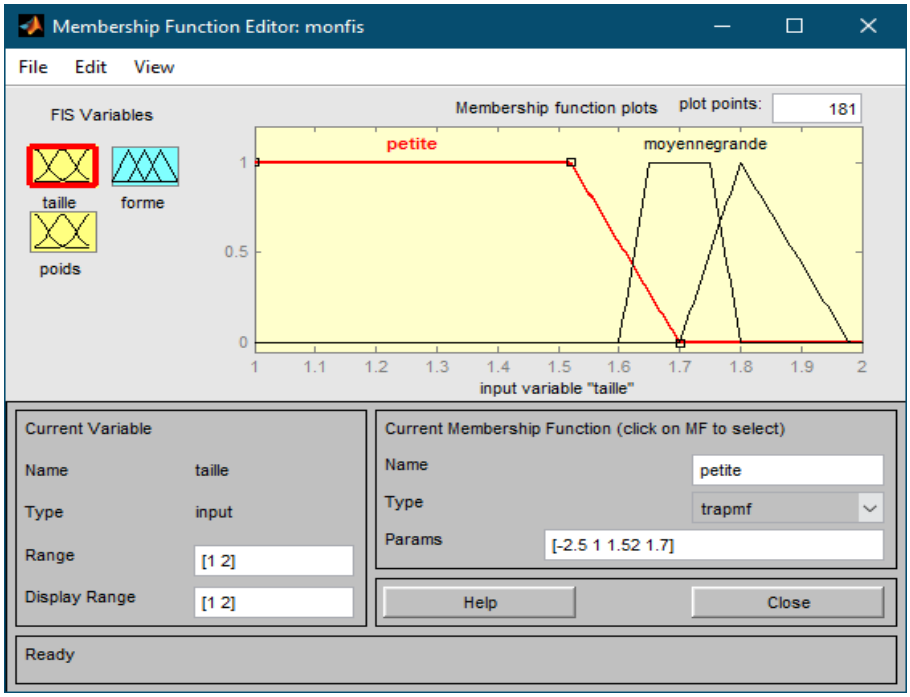


Figure 11. Représentation de l'étiquette « petite » de la variable floue d'entrée « taille »

Sur la figure 10 ci-dessus, les paramètres de notre système flou sont résumés. On voit affichées la méthode de calcul du degré d'appartenance avec des prémisses en conjonction (min) ou en disjonction (max) , la méthode d'agrégation des règles (max) et la méthode de défuzzification des sorties (centroid).

Les figures 17 et 18 montrent les étiquettes de la variable de sortie ***forme*** *(bonne, mauvaise).*
Les règles de production de ce système d'inférence floue sont données par la figure 19 ci-dessous et on note que toutes les règles ont la même valeur de validité égale à 1., c'est-à-dire que toutes les règles ont le même degré d'importance. Deux exemples d'exécution de ce système flou sont donnés respectivement par la figure 20 et la figure 21. Pour chaque exemple d'exécution, on donnera une interprétation des résultats obtenus.

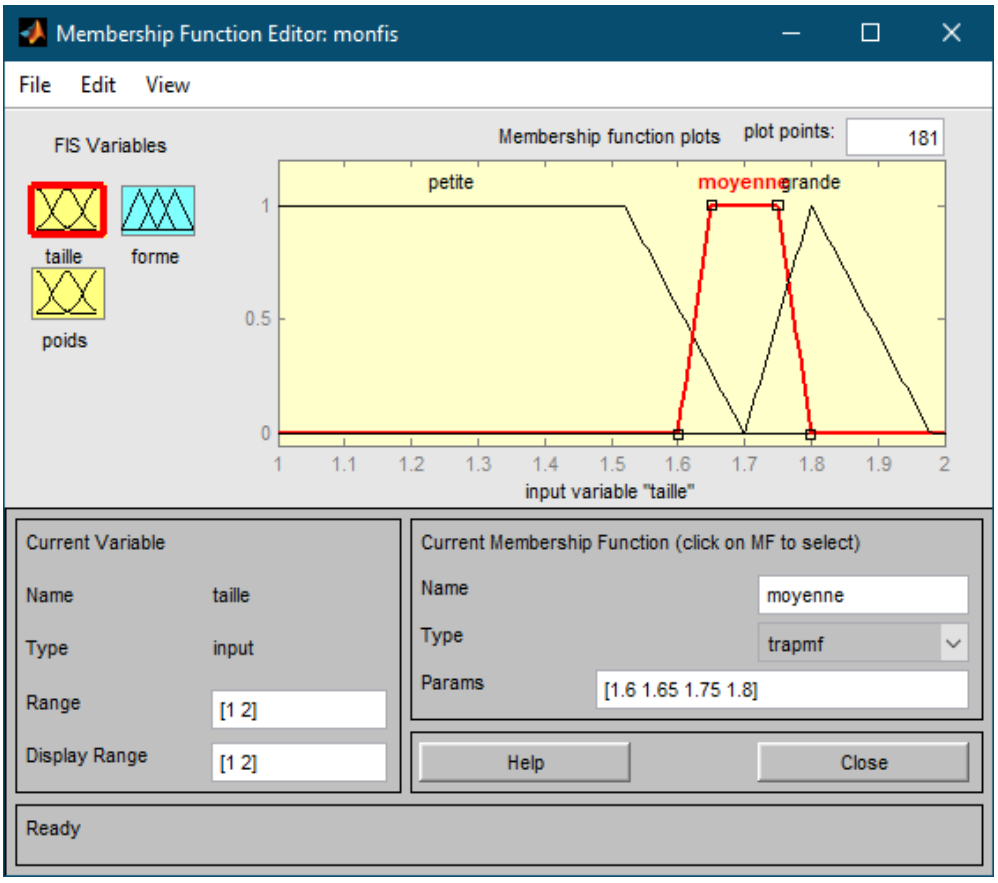


Figure 12. Représentation de l'étiquette « moyenne » de la variable floue d'entrée « taille »

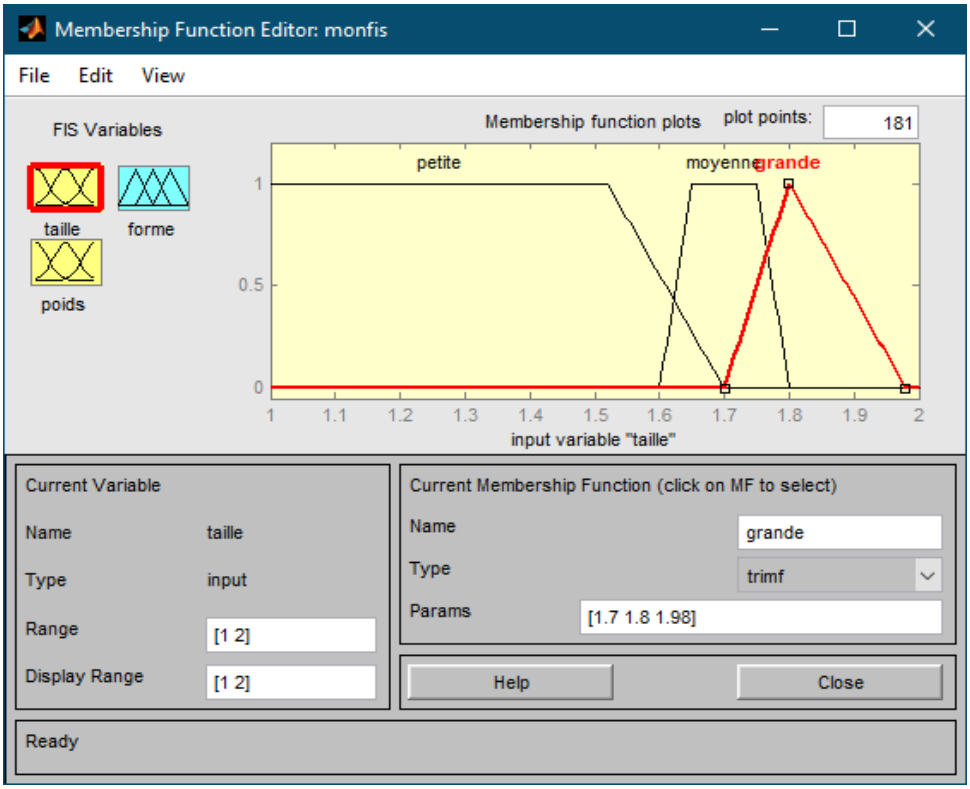


Figure 13. Représentation de l'étiquette « grande » de la variable floue d'entrée « taille »

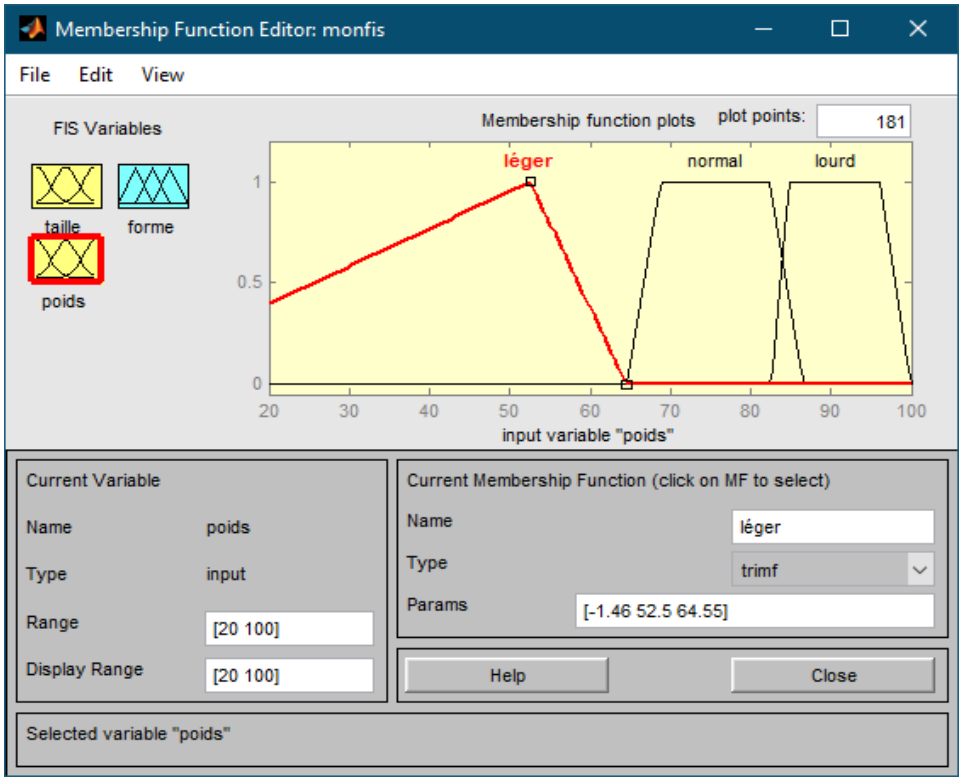


Figure 14. Représentation de l'étiquette « léger » de la variable floue d'entrée « poids »

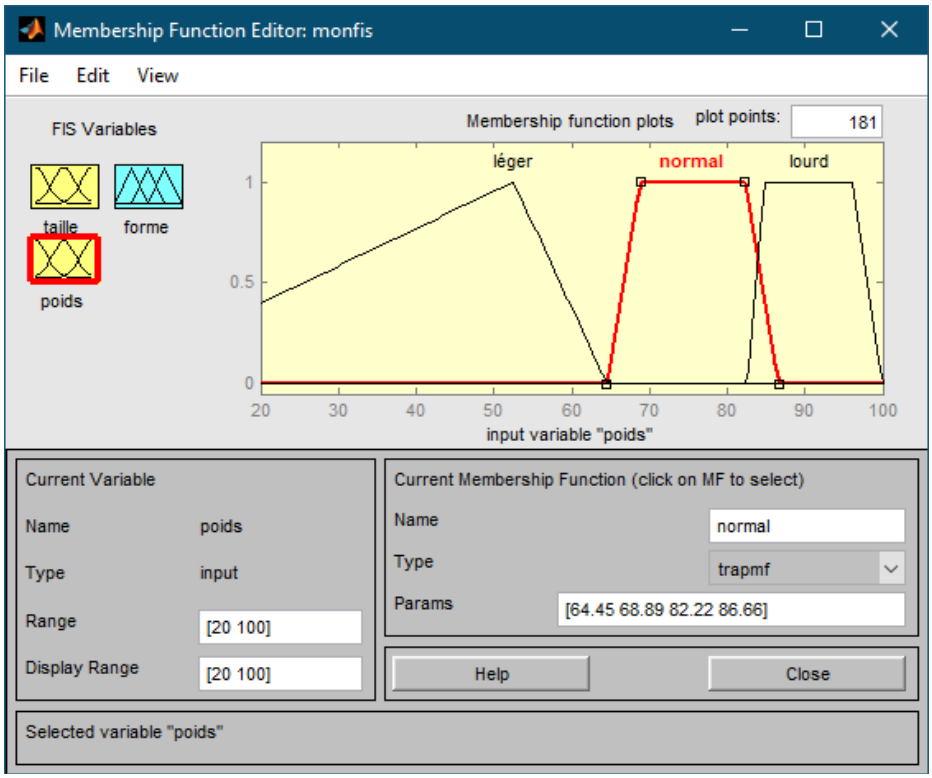


Figure 15. Représentation de l'étiquette « normal » de la variable floue d'entrée « poids »

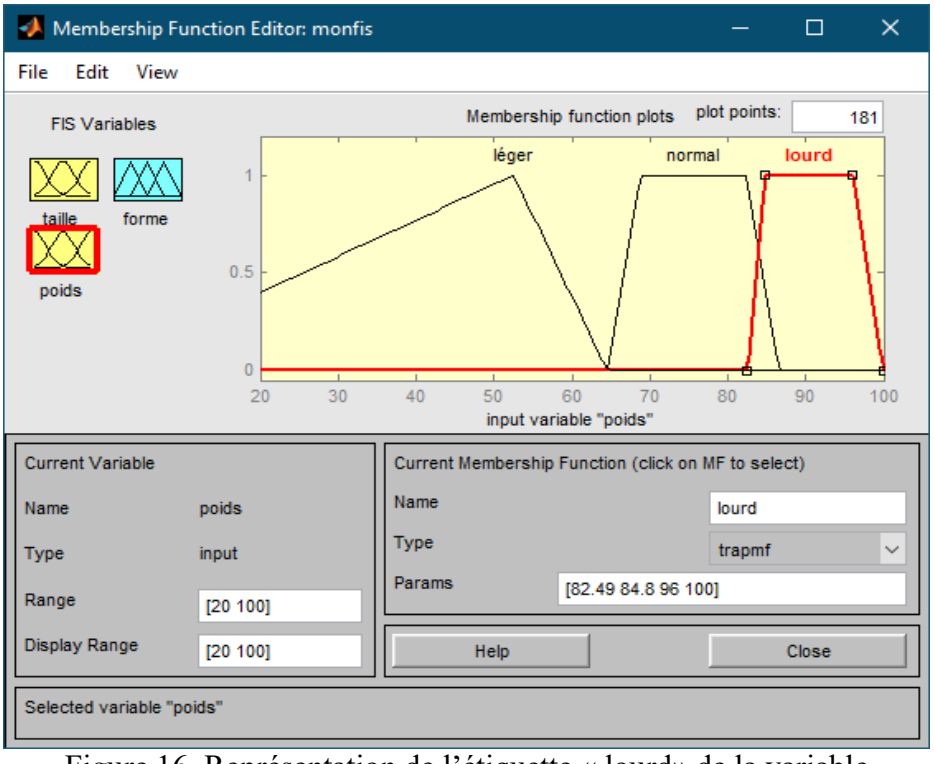


Figure 16. Représentation de l'étiquette « lourd» de la variable floue d'entrée « poids »

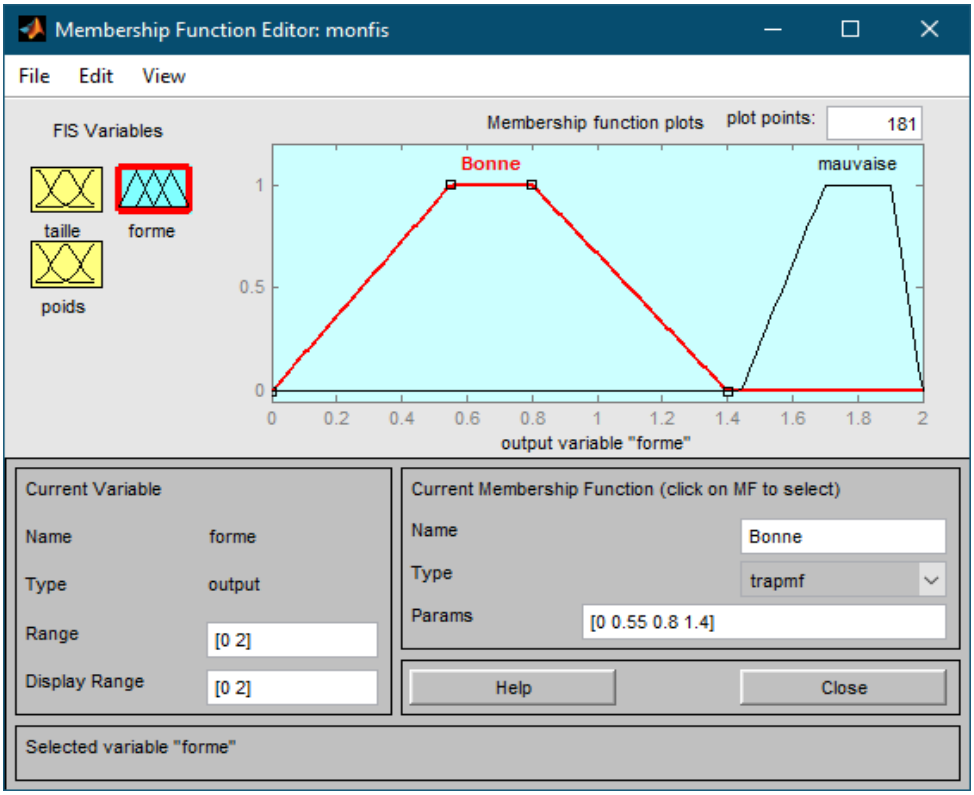


Figure 17. Représentation de l'étiquette « bonne» de la variable floue d'entrée « forme »

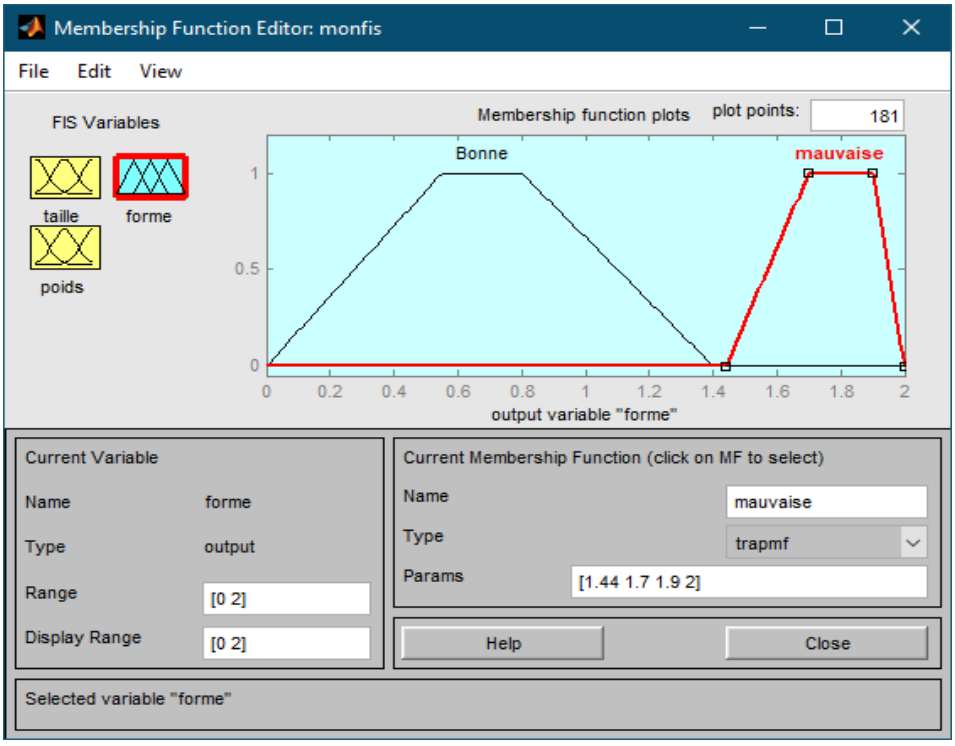


Figure 18. Représentation de l'étiquette « mauvaise» de la variable floue d'entrée « forme »

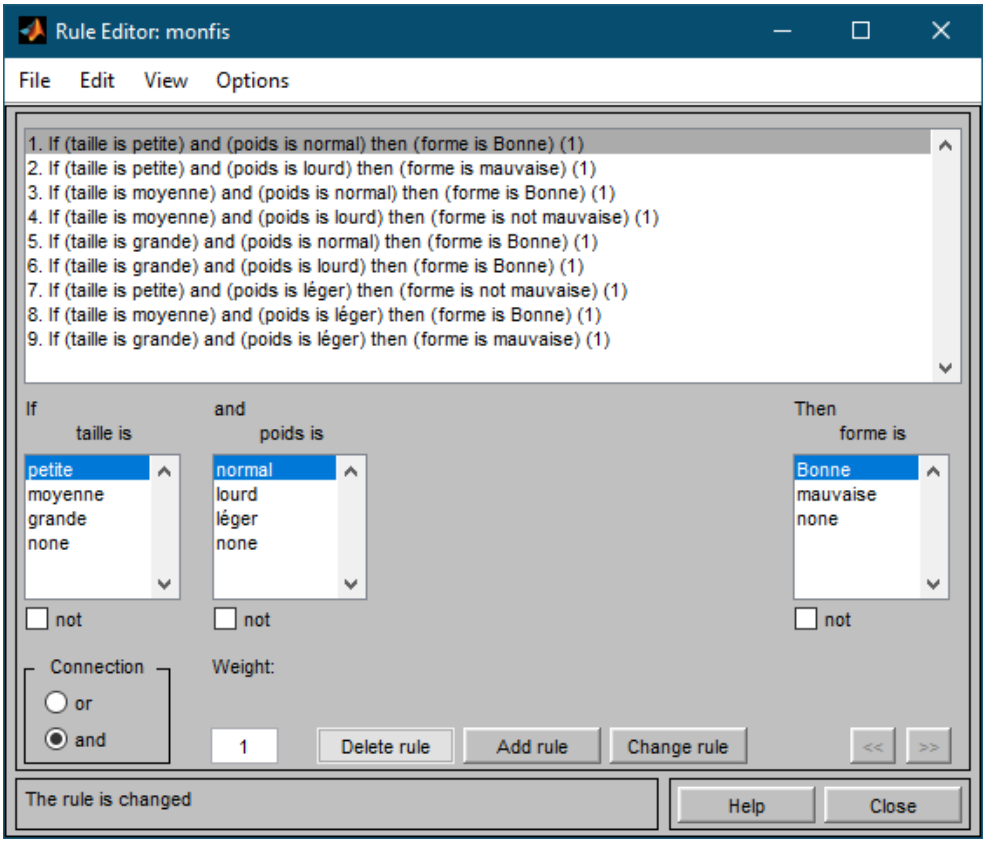


Figure 19 : Les règles du système d'inférence floue.

- **Interprétation des résultats obtenus**

Dans le premier exemple les données numériques en entrée sont : taille =1.77 et le poids = 60. Pour la valeur de la taille = 1.77, les fonctions caractéristiques de la variable floue taille nous indiquent que cette valeur de taille est soit une moyenne ou grande taille, avec des degrés d'appartenance différents, mais certainement n'est pas une petite taille. Tandis que pour la valeur de poids=60, les fonctions caractéristiques de la variable floue poids nous informent que cette valeur de poids est léger avec un certain degré d'appartenance mais certainement n'est pas une valeur de poids qu'on peut qualifier de moyen ou lourd.

Par conséquent les règles à évaluer sont les règles 8 et 9.

règle 8 : if (taille is moyenne) and (poids is leger) then (forme is bonne)
règle 9: if(taille is grande) and (poids is leger) then (forme is mauvaise)

- Evaluation de la règle 8:

On calcule le degré d'appartenance de la taille =1.77 aux étiquettes *moyenne* et *grande* en utilisant leurs fonctions caractéristiques puis on prend le minimum entre les deux (puisque il y'a une la conjonction *and* entre les deux). Cette valeur en sortie sera le degré d'appartenance à l'étiquette *bonne* de la variable de sortie floue *forme*. On forme alors la

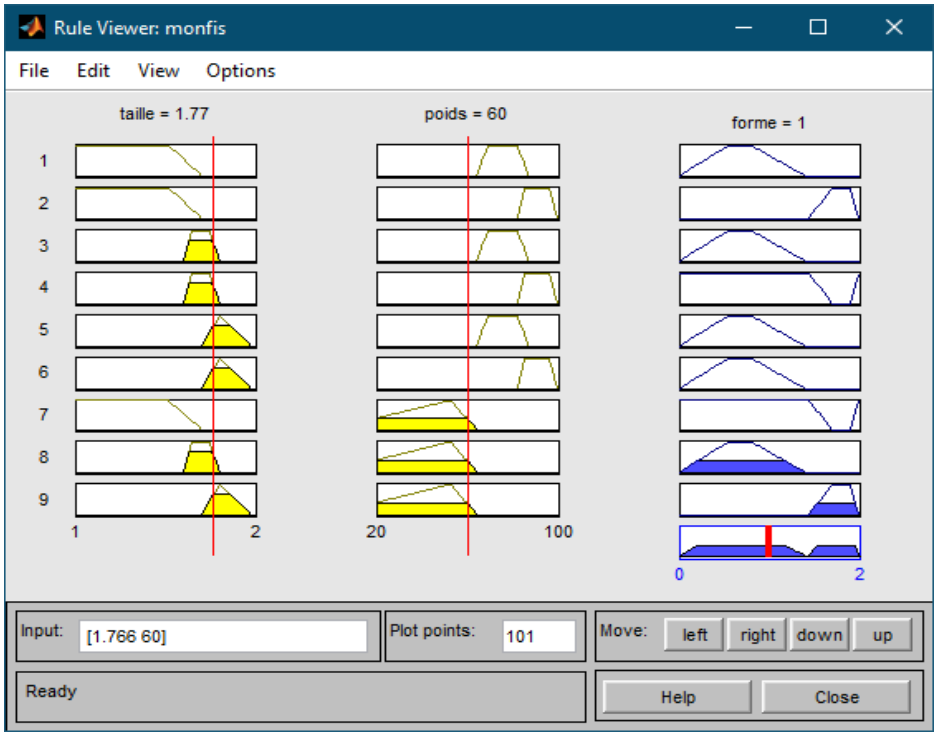


Figure 20 : Première exécution avec résultats de monfis

surface du trapèze formée par l'axe des abscisses, la courbe de l'étiquette *bonne* et la droite horizontale qui passe par le dit minimum, comme montré en bleu (règle 8) dans la figure 20 ci-dessus. De la même façon, on évalue la règle 9 et on obtient la surface du trapèze en couleur bleu règle 9) sur la même figure 20 ci-dessus.

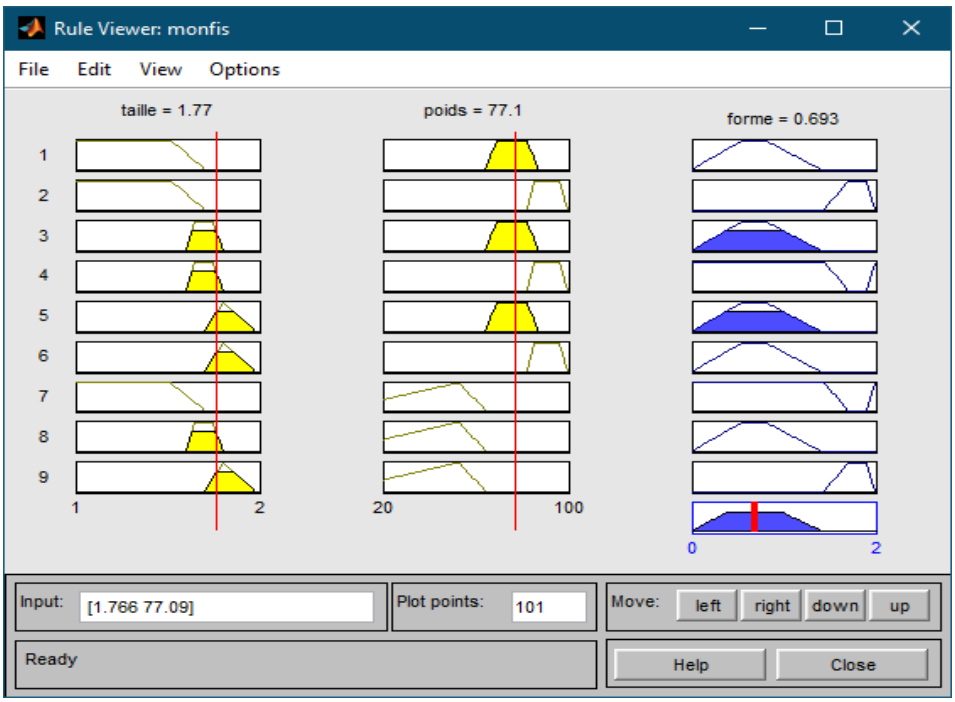


Figure 21: Deuxième exécution avec résultats de monfis.

Enfin pour la defuzzification, On calcule le centre de gravité des deux surfaces agrégées dont l'abscisse représente la sortie numérique de la variable forme, égale à 1 sur la même figure 20 ci-dessus.

Dans le deuxième exemple, figure 21, les valeurs des entrées sont taille= 1,77 et poids= 77.1. Dans ce cas et en appliquant le même raisonnement, on pourra déduire que les règles à évaluer sont dans ce cas les règles 3 et 5. L'évaluation de ces deux règles et l'application du même principe vu dans l'explication de l'exemple 1, à savoir fuzzification des entrées, composition des règles, agrégation des résultats des règles et defuzzification de la sortie, nous permet de calculer une valeur numérique de la sortie, égale à 0.693 sur la même figure 21 ci-dessus.

CHAPITRE 8: PROLOG, Un Langage de Programmation des Systèmes Experts

8.1 Présentation Générale de Prolog

Prolog est l'acronyme du terme composite « programmation logique ». Il est basé sur le langage des prédicats du 1er ordre, qui est un langage logique. Il est utilisé en intelligence artificielle pour concevoir des systèmes à base de connaissances et aussi pour le traitement automatique des langages naturels. A l'encontre des autres langages de programmation , de type procédural ou impératif, ou fonctionnel, PROLOG est de type déclaratif, c'est à dire que le programmeur n'a que faire la déclaration des connaissances liées au domaine étudié et l'interpréteur Prolog recherche les solutions possibles répondant aux toutes requêtes de l'utilisateur en utilisant le mécanisme de raisonnement intégré au langage Prolog. Le langage Prolog utilise un raisonnement de type chainage en arrière et permet de trouver les réponses aux requêtes de l'utilisateur par unification des termes. Historiquement parlant, le langage a connu les évolutions suivantes :

1965:John Robinson décrit les principes théoriques de la programmation logique via la règle de résolution.
1972 : Création de PROLOG par A. Colmerauer et Ph. Roussel à Marseille (Luminy), pour le traitement des langues naturelles.
1977: Premier compilateur PROLOG a vu le jour , par D.H. Warren à Edimbourg
1980 : reconnaissance de PROLOG comme langage de développement en Intelligence Artificielle
Depuis, plusieurs versions ont été développées avec un standard ISO, plusieurs extensions possibles.

8.2. Les éléments du langage PROLOG

8.2.1 Les constantes et les variables

Une constante en PROLOG commence par une lettre minuscule ou mise entre apostrophe ou un entier ou un flottant. Les termes suivants sont tous des constantes :

Exemple 17: aHmeD, 'Ahmed', 25, 0.25

Une variable en PROLOG, Commence toujours par une lettre majuscule ou par un tiret bas _ qui dénote une variable anonyme. Les variables utilisées dans un fait ou une règle sont quantifiées par PROLOG par le quantificateur universel, tandis que celles utilisées dans une requête sont quantifiées par le quantificateur existentiel.

Exemple 18: X, Xy, _ , _X.

8.2.2 Les opérateurs

L'interpréteur Prolog considère les opérateurs comme des foncteurs et transforme les expressions exprimées sous forme infixées en une forme préfixées comme montre l'exemple suivant :

Exemple 19 : 3*5+6*7 est transformée en l'expression +(*(3,5),*(6,7))

Les opérateurs en PROLOG, sont de type arithmétique, logique et de relation. Ceux les plus utilisés sont résumés dans le tableau 5 ci-dessous :

Opérateur arithmétique	Signification	Opérateur logique	Signification	Opérateur de relation	Signification
+(X)	Plus	X /\ Y	ET	X=Y	unification
-X	Moins	X \/ Y	OU	X==Y	Egalité
X+Y	Addition	X # Y	OUX	X > Y	Supérieur
X-Y	Soustraction	\(X)	Inversion des bits de X	X < Y	Inférieur
X*Y	Produit	X >>Y	Décalage de Y bits de X vers la droite	X =< Y	Inf. ou égal.
X/Y	Division flottante	X <<Y	Décalage de Y bits de X vers la gauche	X >= Y	Sup. ou égal.
X//Y	Division entière			X \=Y	Différent

Tableau 5 : Les opérateurs en PROLOG.

8.2.3 Les Foncteurs

C'est une fonction qui effectue un traitement. La syntaxe générale d'un foncteur est $f(t_1,t_2,\ldots,t_n)$
où f est le symbole du foncteur et t_i des termes où chaque terme peut être une constante, une variable ou un autre foncteur. Il existe en Prolog des foncteurs mathématiques, graphiques et autres.

Exemple 20 :

cputime/0 : permet d'avoir le temps CPU.
is_list/1 : permet de savoir si l'argument est une liste ou non.
denominator/1 : permet d'avoir le dénominateur d'un nombre rationnel (N/D).
div/2 : effectue la division entière entre deux nombres.

8.2.4 Les prédicats

Un prédicat est une relation mettant en relation un ou plusieurs objets. L'identificateur d'un prédicat doit commencer par une lettre minuscule.

Exemple 21

pere (X,Adam).
pere(ali,mohamed).

element(a, [a,b,c])
associe(mohamed,samir).

8.2.5 Les clauses

Une clause est une connaissance qui peut être un fait ou une règle. Un fait comme son nom l'indique est une connaissance dont la véracité n'est pas liée à une condition. Par contre la règle elle est formée de deux parties, une partie gauche appelée *conclusion* et une partie droite appelée *condition*. Les deux parties de la règle sont séparée par le symbole **:-** qui joue le rôle de l'implication logique. A noter qu'une clause doit impérativement être terminée par un point.

Exemple 22

```
footballer (ahmed).              Un fait
sportif(X) :-  footballeur(X).   Une règle
```

Le fait se lit « Ahmed est un footballeur » et la règle se lit : « Tout footballeur est un sportif »

Une clause peut tenir sur plus d'une ligne mais doit être terminée par un point comme nous l'avons mentionné précédemment.

Dans le cas où la partie conclusion d'une règle est composée, les prédicats la composant seront séparés par une virgule(,) pour indiquer la conjonction (et) et par un point-virgule pour indiquer la disjonction(ou).

Exemple 23 :

```
sportif(X) :-  footballeur (X) ; handballeur (X) ; voleyballer(X).
frere(X,Y) :- pere(X,Z), pere(Y,Z).
frere(X,Y) :- mere(X,Z),mere(Y,Z).
ascendant(X,Y):- mere(X,Y) ; pere(X,Y)
```

Un programme Prolog est donc un ensemble de clauses dont l'ordre n'est pas obligatoire. Il peut être organise en un ensemble de paquets ou chaque paquet est un ensemble de clauses contenant le même prédicat.

8.2.6 L'affectation

Elle est réalisée grâce à l'operateur ***is*** , qui permet d'unifier une variable avec une expression arithmétique.

Exemple 24 : X is 8*5+2 .

8.2.7 Les structures itératives

En Prolog, il n'y a pas de structures itératives telles que for, while, do … while, qui sont propres aux langages impératifs. En revanche toutes ces fonctionnalités itératives sont programmées de manière récursive [Ri, 05]. Ainsi, pour parcourir une liste, on examine le premier élément de la liste, puis on effectue le traitement approprié sur cet élément avant d'appeler récursivement le prédicat sur le reste de la

liste. C'est de cette façon que l'on construit la liste résultat. La répétition par le moyen de la récursivité sera abordée par la suite avec la structure de données LISTES.

8.2.8 Les structures conditionnelles

En Prolog, la structure conditionnelle alternative si-alors-sinon n'existe pas au même titre qu'il y'a dans un langage de programmation impératif [Ri 04] mais il existe une structure alternative permettant de la simuler, qui est : A -> B ; C

Qui se lit si A est prouvée alors prouvez B et ignorez C et vice -versa

Exemple 25

max(X,Y, Z) :- (X =< Y -> Z = Y ; Z = X).

8.2.9 Les listes

Une liste en Prolog est une structure de données contenant des valeurs de même type entre deux accolades. Le nom d'une liste doit commencer par une lettre majuscule.

Exemple 26

L1=[1,2,11,5].
L2=['a','bc','xy'] .
L3=[]. Liste vide.

Une liste en Prolog, peut être définie d'une manière récursive par son premier élément et la liste du reste de la liste, comme illustré dans l'exemple suivant :

Exemple 27

L1=[1,[2,11,5]].
L1=[1,[2,[11,5]]].
L2=['a',['bc',xy'].
L2=['a',['bc',['xy']]].

Comme nous l'avons indiqué précédemment, le traitement de liste en Prolog, grâce à sa définition récursive, permet d'incarner le principe de répétition grâce à la définition de prédicats récursifs, comme illustré dans les exemples suivants.

Exemple 28 : calcul de la somme des éléments d'une liste de nombres par un prédicat récursif.

somme([],0).
somme([X| L],S) :- somme(L,R), S is R+X.

Exemple 29 : test d'appartenance d'un élément X à une liste L par un prédicat récursif.

```
appart(X,[X|_]).
appart(X,[_|L]) :- appart(X,L).
```

Exemple 30 : calcul de la longueur d'une liste L =[_|L1], par un prédicat récursif.

```
long([],0) .
long([_|L1],N) :- long(L1,M) , N is M+1.
```

8.2.10 Les Sets

Un set est une liste dont les éléments ne sont pas cités plus d'une fois. Donc un set est une liste mais la réciproque n'est pas toujours vraie.

Exemple 31:

C =[r, v, b, j]. est une liste et un set aussi.
N=[1,5,4,2,1]. est une liste mais non un set car l'élément 1 est répété deux fois.

La bibliothèque Prolog contient des prédicats permettant de vérifier si c'est une Liste ou un Set. De même, de faire la conversion d'une liste vers un set.

Exemple 32

```
?is_set([1,2,3]).
true.

?is_set(1,1,2,3]).
false
?is_list([1,1,2,3]).
true.
?Is_list([1,2,3,4]).
true
?list_to_set([1,1,2,3],X) .
X=[1,2,3]
?list_to_set([1,2,2,4,5,5],[1,2,4,5]).
true.
```

8.3 La programmation logique par contraintes (PLC) sous Prolog

8.3.1 Introduction

Le terme "Programmation Logique avec Contraintes" (PLC) (En Angalis CLP : Constraint Logic Programming) , apparue au milieu des années 80 dans le domaine de l'Intelligence Artificielle (IA) par les auteurs de langages de programmation logique comme Prolog. La PLC consiste à poser un problème sous forme de relations logiques (appelées contraintes) entre plusieurs variables. Un problème formulé de la sorte comporte donc un certain nombre de ces variables, chacune ayant un domaine (i.e. un ensemble de valeurs possibles), et un certain nombre de contraintes. Le domaine d'une variable peut être fini

(intervalle de nombres entiers, intervalle d'ensembles par exemple) ou infini. L'objectif de la PLC est donc tout simplement la résolution de problèmes exprimés sous forme d'un ensemble de contraintes de type logique. Des exemples de problèmes auxquels est appliquée la PLC sont les problèmes combinatoires tels les problèmes de planification et les problèmes de placement ou d'aménagement.
Un Langage de programmation logique, tel Prolog, offre des fonctionnalités et une bibliothèque pour modéliser et résoudre les problèmes à contraintes.

8.3.2 Notion de problème de satisfaction de contraintes

Un PSC (Problème de Satisfaction de Contraintes, en Anglais CSP : Constraint Satisfaction Problem), est un problème modélisé sous la forme d'un ensemble de contraintes posées sur des variables, chacune de ces variables prend ses valeurs dans un domaine.
De façon plus formelle, on définira un CSP par un triplet *(X, D, C)* tel que

- $X = \{ X_1, X_2, ..., X_n\}$ est l'ensemble des variables (les inconnues) du problème ;
- *D* est la fonction qui associe à chaque variable *Xi* son domaine *D(Xi)*, c'est-à-dire l'ensemble des valeurs que peut prendre *Xi ;*
- $C = \{C_1, C_2, ..., C_k\}$ est l'ensemble des contraintes.

Exemple 33

Soit le CSP (X, D, C) suivant :

- X = {a, b, c, d}
- D(a) = D(b) = D(c) = D(d) = {0,1}
- C = {a ≠ b, c ≠ d, a+c < b}

Ce CSP comporte 4 variables *a*, *b*, *c* et *d*, chacune pouvant prendre 2 valeurs (*0* ou *1*). Ces variables doivent respecter les contraintes suivantes : *a* doit être différente de *b* ; *c* doit être différente de *d* et la somme de *a* et *c* doit être inférieure à *b*.

8.3.3 Résolution d'un problème de satisfaction de contraintes

La résolution d'un PSC consiste à trouver une affectation de toutes les variables du CSP dans leurs domaines respectifs qui ne viole aucune des contraintes du CSP.

Exemple 34

Dans le CSP précèdent, une solution possible est l'affectation { a=0,b=1,c=0,d=1}
Lorsqu'un CSP n'a pas de solutions, il est dit sur-contraint. Dans ce cas, on cherche dans la pratique à trouver les solutions qui violent le minimum de contraintes qualifiées de moins importantes. Dans le cas contraire où il existe un nombre important de solutions, il est dit sous-contraint. Le cas échéant, de point de vue pratique on définit une échelle de préférences sur ces solutions en définissant une fonction objective de façon à ne retenir que la ou les solutions optimale(s).

8.3.4 Notion de solveur de contraintes

Un langage de programmation logique tel Prolog incorpore un interprète chargé de la résolution de problèmes modélisés par des CSP appelé solveur de contraintes. Gnu Prolog et SW-prolog incorporent

chacun un solveur CLPFD (Constraint Logic Programming over Finite Domains, ou en Français Programmation Logique par Contraintes sur les Domaines Finis). On va illuster cette notion de solveur de contraintes par l'exemple ci-dessous.

Exemple 35

Soit le CSP (V , D, C) défini comme suit :
V={x, y, z} ; D= {D(x) = [2,5] ; D(y)=D(z) = [-1,5]) ; C= {x>0 ;y >2 ; x + y- z <1}.
Traduire et résoudre ce CSP sous le langage SWI-Prolog ?

Le programme SWI-prolog permettant de traduire le CSP donné est le suivant en texte encadré.
La première ligne indique à PROLOG d'inclure la bibliothèque CLPFD qui est l'interface pour l'utilisation de la programmation logique par contraintes sur les domaines finis. La deuxième ligne défini le problème en termes de variables, de leurs domaines respectifs et l'ensemble des contraintes sur ces variable et ordonne au solveur prolog de chercher les valeurs de ces variables vérifiant lesdites contraintes grâce au prédicat *label (liste_de_variables).*

```
:- use_module(library(clpfd)).
problem(X,Y,Z):- X in 2..5, Y in -1..5, Z in -1..5,X#>0,Y#>2,(X+Y-Z)#<1,label([X,Y,Z]).

?problem (X,Y,Z).
X = 2,
Y = 3,
Z = 5.
```

Donc on a une seule solution pour ce problème CSP (X=2,Y=3,Z=5)

8.3.5 Les algorithmes des solveurs de contraintes

Le principe des algorithmes sur lesquels sont basés les solveurs de contraintes est de chercher dans l'ensemble des affectations possibles jusqu'à, soit trouver une solution, soit énoncer qu'il n'existe pas de solution. Il existe deux types d'algorithmes de résolution : algorithmes complets, algorithmes incomplets :

- Algorithmes complets : Ici on effectue une recherche exhaustive de tout l'espace de recherche et l'existence d'une solution est certaine si le CSP est consistant mais en revanche cette technique n'est applicable à des types de problèmes à explosion combinatoire ou l'algorithme du solveur ne termine pas dans un temps raisonnable, voire dépasse l'échelle humaine.

- Algorithmes incomplets : A l'inverse des algorithmes complets, dans le cas des algorithmes incomplets la recherche dans l'espace des combinaisons ne fait pas d'une exhaustive mais plutôt on cherche une solution (affectation) acceptable mais dans un temps raisonnable. Il existe différents algorithmes incomplets pour résoudre de façon générique des CSPs sur les domaines finis, généralement basés sur des techniques de recherche locale et méta heuristiques tels les algorithmes de colonie de fourmis.

Dans ce qui suit, on va décrire quelques algorithmes de solveurs de contraintes.

- **Algorithme générer et tester (GT)**

Le solveur génère l'ensemble des affectations possibles et s'arrête dès que l'on en trouve une qui satisfasse les contraintes . Autrement dit, il énumère une par une les valeurs des variables. Quand chaque variable a une valeur, on teste si la contrainte est satisfaite ou pas. L'algorithme fait appel à la fonction récursive "GET(A,(X,D,C))". Cette fonction, commence par une affectation partielle A et un CSP (X,D,C). Au premier appel de cette fonction, l'affectation partielle A sera vide. La fonction retourne vrai si on peut étendre l'affectation partielle A en une affectation totale consistante (une solution), et faux sinon. L'algorithme de ce solveur est donné par la figure 22 ci-dessous

```
fonction GET(A,(X,D,C))  retourne un booléen
Précondition :
(X,D,C) = un CSP sur les domaines finis
A = une affectation partielle pour (X,D,C)
Postrelation : retourne vrai si l'affectation partielle A peut être étendue en une solution pour
(X,D,C), faux sinon.
Début
si (toutes les variables de X sont affectées à une valeur dans A )alors /* A une affectation totale
*/
 si   (A est consistante)  alors                    /* A est une solution */
      retourner vrai
  sinon  retourner faux
   finsi
     sinon                                          /* A est une affectation partielle */
    choisir une variable Xi de X qui n'est pas encore affecté à une valeur dans A
pour (toute valeur Vi appartenant à D(Xi))    faire
    si GET(A U {(Xi,Vi)}, (X,D,C)) = vrai alors retourner vrai
  finpour
        retourner faux
         finsi
fin
```

Figure 22 : Algorithme générer et tester

- **Algorithme « retour en arrière »/ « Backtracking »**

Pour pallier aux limites de l'algorithme générer et tester, l'algorithme backtacking (Voir figure 23) procède de la manière que si une affectation est inconsistante alors on arrête l'exploration de cette voie et on fait un retour arrière jusqu'à la première affectation partielle consistante. Ce principe est représenté dans la fonction récursive « simpleRetourArrière(A,(X,D,C)) » . *A* contient une affectation partielle et *(X,D,C)* décrit le CSP à résoudre.

```
Fonction simpleRetourArrière(A,(X,D,C)) retourne un booléen
Précondition :
        A = affectation partielle
        (X,D,C) = un CSP sur les domaines finis
Postrelation :
        retourne vrai si A peut être étendue en une solution pour (X,D,C), faux sinon
début
        si A n'est pas consistante alors retourner faux finsi
        si toutes les variables de X sont affectées à une valeur dans A alors
                /* A est une affectation totale et consistante = une solution */
                retourner vrai
        sinon /* A est une affectation partielle consistante */
                choisir une variable Xi de X qui n'est pas encore affectée à une valeur dans A
                pour toute valeur Vi appartenant à D(Xi) faire
                        si simpleRetourArrière(A U {(Xi,Vi)}, (X,D,C)) = vrai alors retourner
                        vrai
                finpour
         retourner faux
        finsi
fin
```

Figure 23 : Algorithme backtracking

- **Solveur basé sur l'algorithme par anticipation « Look Ahead »**

Pour améliorer l'algorithme "simple retour-arrière", on peut tenter d'anticiper ("Look Ahead" en anglais) les conséquences de l'affectation partielle en cours de construction sur les domaines des variables non encore affectées : Si on voit qu'une variable non affectée *Xi* n'a plus de valeur dans son domaine *D(Xi)* qui soit "localement consistante" avec l'affectation partielle en cours, alors il n'est pas nécessaire de continuer à développer cette branche, et on peut retourner en arrière pour explorer d'autres possibilités. Ce filtrage anticipé de variables peut être effectué selon trois niveaux : filtrage de nœuds, filtrage d'arcs et filtrage de chemins.

```
Algorithme  Filtrage _nœuds  (CSP(X,D,C), A)
/* CSP (X, D, C) ,  A : l'affectation  en cours A={X1, X2,..,Xi-1} */
Debut
Pour chaque variable Xi  de  X non affectée dans A
  Faire
    Pour chaque valeur  v de  D(Xi)
    Faire
      Si   (A U {(Xi, v)} )  est  inconsistante    /* ne respecte pas toutes les contraintes  C  */
        Alors
          D(Xi) =  D(Xi)  -{v}                    /* enlever la valeur( v,w ) de (D(Xi),D(Xj))   */
      Finsi
   Finfaire
Finfaire
Fin
```

Figure 24 : Algorithme de filtrage de noeuds

Le filtrage de nœuds (Voir algorithme figure 24 ci-dessus) permet d'éliminer toutes les valeurs des variables non encore affectées qui violent les contraintes unaires liées à ces variables. Un tel filtrage permet d'établir ce qu'on appelle la *consistance de nœuds*, aussi appelée *1-consistance*.

un filtrage plus fort, mais aussi plus long à effectuer, consiste à anticiper de deux étapes l'énumération : pour chaque variable *Xi* non affectée dans *A*, on enlève de *D(Xi)* toute valeur *v* telle qu'il existe une variable *Xj* non affectée pour laquelle, pour toute valeur *w* de *D(Xj)*, l'affectation *A U {(Xi,v),(Xj,w)}* soit inconsistante. Notons que ce filtrage doit être répété jusqu'à ce que plus aucun domaine ne puisse être réduit. Ce filtrage permet d'établir ce qu'on appelle la ***consistance d'arc***, aussi appelée ***2-consistance***. La figure 25 donne l'algorithme de filtrage des arcs.

```
Algorithme  Filtrage _arcs  (CSP(X, D, C), A)
/* CSP (X, D, C) : Problème  de Satisfaction  de Contraintes */
/* A : l'affectation  en cours A={X1, X2,.., Xi-1}
Debut
Pour chaque  couple de variable s (Xi  ,  Xj ) de  X non affectées
  Faire
    Pour chaque valeur  v de  D(Xi) et w de D(Xj)
    Faire
      Si   (A U {(Xi, v)} U {(Xj, w) )  est  inconsistante    /* ne respecte pas toutes les
contraintes  C  */
         Alors
             D(Xi) x D(Xj)   = D(Xi) x D(Xj)   - { ( v, w)  }    */ enlever le couple de valeurs
            (v,w) de l'ensemble des couples de valeurs possibles  D(Xi) x D(Xj) * /
       Finsi
   Finfaire
Finfaire
Fin
```

Figure 25 : Algorithme de filtrage d'arcs.

Un autre filtrage encore plus fort, mais aussi encore plus long à effectuer, consiste à anticiper de trois étapes l'énumération. Ce filtrage permet d'établir ce qu'on appelle la consistance de chemin, aussi appelée 3-consistance. Par généralisation, on peut anticiper de k étapes l'énumération pour établir la k-consistance, l'opération de filtrage revient à résoudre le CSP, c'est-à-dire que toutes les valeurs restant dans les domaines des variables après un tel filtrage appartiennent à une solution.

8.4 Un Système Expert « Pédagogique » en Prolog

Comme exemple de système expert, on donne ci-après la traduction de l'exemple « enquête policière » de l'exemple traité dans la section 4.2.4 en langage SWI-PROLOG avec une requête d'exécution.

Exemple 36

La liste des clauses (faits et règles) du programme en question est donnée par la figure 26 ci-dessous. Le programme est composé de 9 règles et 18 faits. Pour résoudre le problème et avoir automatiquement le résultat de l'enquête policière, on interroge le système expert par l'introduction de la requête : ? voler (X, gateaux). La signification de cette requête est « quelles sont toutes les personnes ayant volé le

gateaux ». La réponse à cette question par l'interpréteur du système expert est X= mounir. (voir figure 27 , ci-dessous) qui est d'ailleurs la seule réponse du système à la requête soumise.

enquete.pl [modified]

File Edit Browse Compile Prolog Pce Help

enquete.pl [modified]

```
/*  les regles */

voler(X,gateaux) :- besoin(X,gateaux), avoiracces(X,gateaux).
voler(X,gateaux) :-besoin(X,gateaux), forcer(X,porteplacad).
besoin(X,gateaux) :- gourmand(X) ; doitnourrir(X,Y).
avoiracces(X,gateaux) :- possede(X,cleplacad).
forcer(X,porteplacad) :- blesse(X), forcee(porteplacard).
doitnourrir(X,Y) :- possede(X,Y), animal(Y).
besoin(X,gateaux) :- doitnourrir(X,Y), besoin(X,argent).
gourmand(X) :- suivreregime(X) ; aimecuisiner(X).
manger(X,gateaux) :- fairemiettes(X,gateaux)

/* Les faits  */
voisinsdirects(houssem, raouaf,meriem, soufiene).
mere(yasmina, jihan).
demifrere(mounir,jihan).
oncle(karim, mounir).
possede(jihan, cleplacard).
possede(mounir, cleplacard).
possede(yasmina,cleplacard).
besoin(raouf, argent).
possede(raouf,canaris).
animal(canaris).
suivreregime(mounir).
blesse(mounir).
aimecuisiner(meriem).
fairemiettes(karim,gateaux).
riche(houssem).
mince(houssem).
malin(houssem).
forcee(porteplacard).
```

Line: 13

Figure 26 : Clauses du système expert

```
% c:/users/walid  g/onedrive/documents/prolog/enquete compiled 0.02 sec, -1 clause
?- voler(X,gateaux).
X = mounir ;
false.

?-
```

Figure 27 : Exécution d'une requête

Chapitre 9. Les Réseaux de neurones

9.1 Première définition

Un réseau de neurones artificiels est un modèle informatique du cerveau humain dont l'unité de base est appelée le neurone formel, inspiré du neurone biologique.

9.2 Le cerveau humain, quelques chiffres

Le cerveau humain contient 10^{11} (100 milliards) de cellules nerveuses interconnectées. La vitesse de transmission du signal du cerveau humain est de l'ordre de 100 m/s et le nombre de connexions qu'il contient est de l'ordre de 10^{15}

9.3 Le neurone biologique

Est la cellule nerveuse constituée d'un corps cellulaire contenant un noyau et des ramifications appelées les dendrites (l'organe d'entrée des signaux émanant des autres cellules) et se prolonge par un axone qui est le moyen de transmission du signal vers les autres cellules. (Voir figure 28)
Le sens du signal : dendrites, corps cellulaire, axone, synapses, autres neurones. (Voir figure 29)

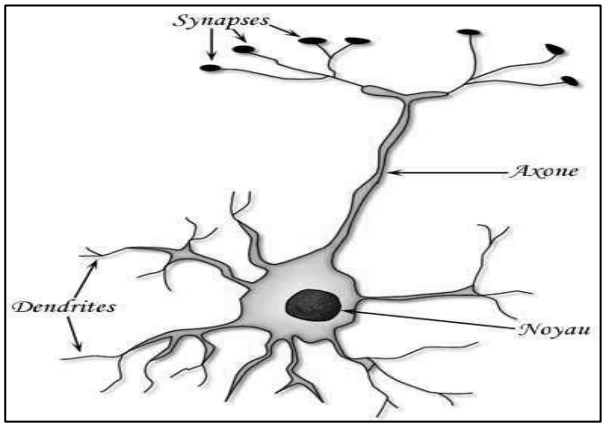


Figure 28 : Le neurone biologique

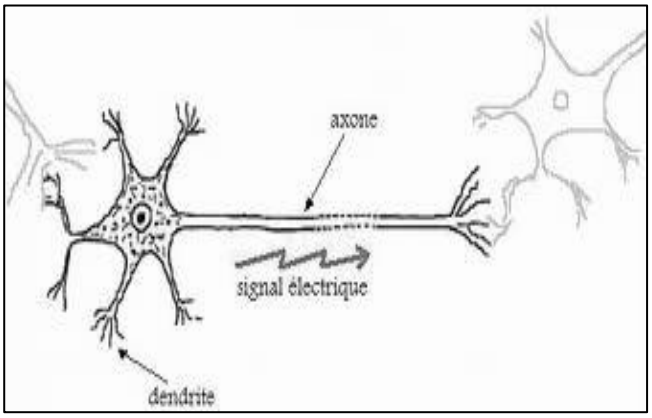


Figure 29. La transmission du signal dans le neurone biologique

9.4 Le neurone formel

Est le modèle mathématique du neurone biologique (Voir figure 30 et figure 31). Il est composé des éléments suivants :

- Entrées : signaux émanant des autres neurones ou de l'extérieur.
- Poids: pondérations des valeurs des entrées. Représentent les synapses.
- Corps cellulaire : effectue la sommation (∑) des entrées pondérées et applique un seuillage par une fonction d'activation (f) au résultat.
- Sortie : représente l'axone et prend comme valeur le résultat de calcul effectuée par le corps cellulaire.

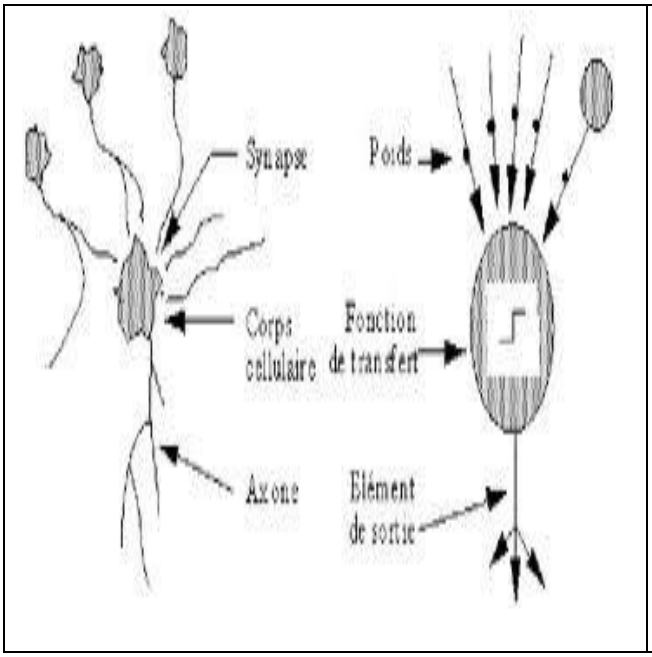


Figure.30 : Correspondance entre neurone biologique et neurone formel

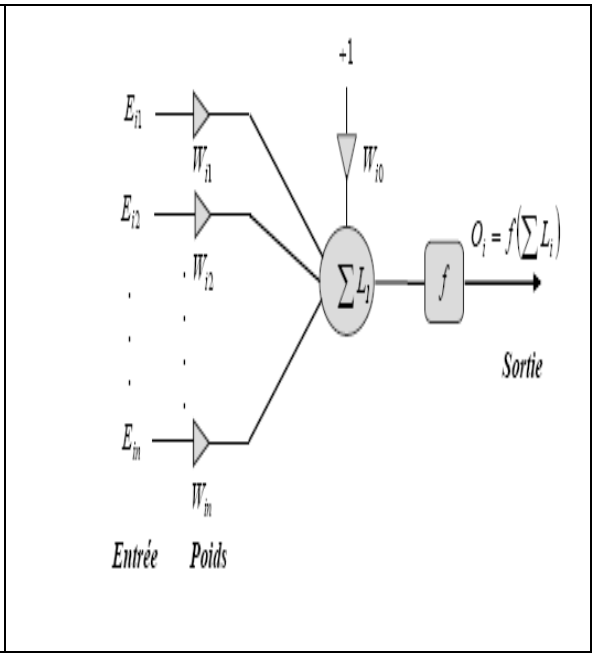


Figure. 31: Neurone formel

Remarque : Dans la figure 31 du neurone formel, l'entrée (+1) ayant le poids W_{i0}) représente le seuil interne du neurone, dit aussi le biais.

9.5 Fonctionnement du neurone formel

Le fonctionnement du neurone formel passe par les trois étapes suivantes :

1) Réception des données d'entrée E_i
2) Sommation : $L_k = \sum W_i * E_i \quad / \ i : 0 \ldots k$ (on suppose que nous avons k neurones)
3) Seuillage par une fonction d'activation f, généralement non linaire.

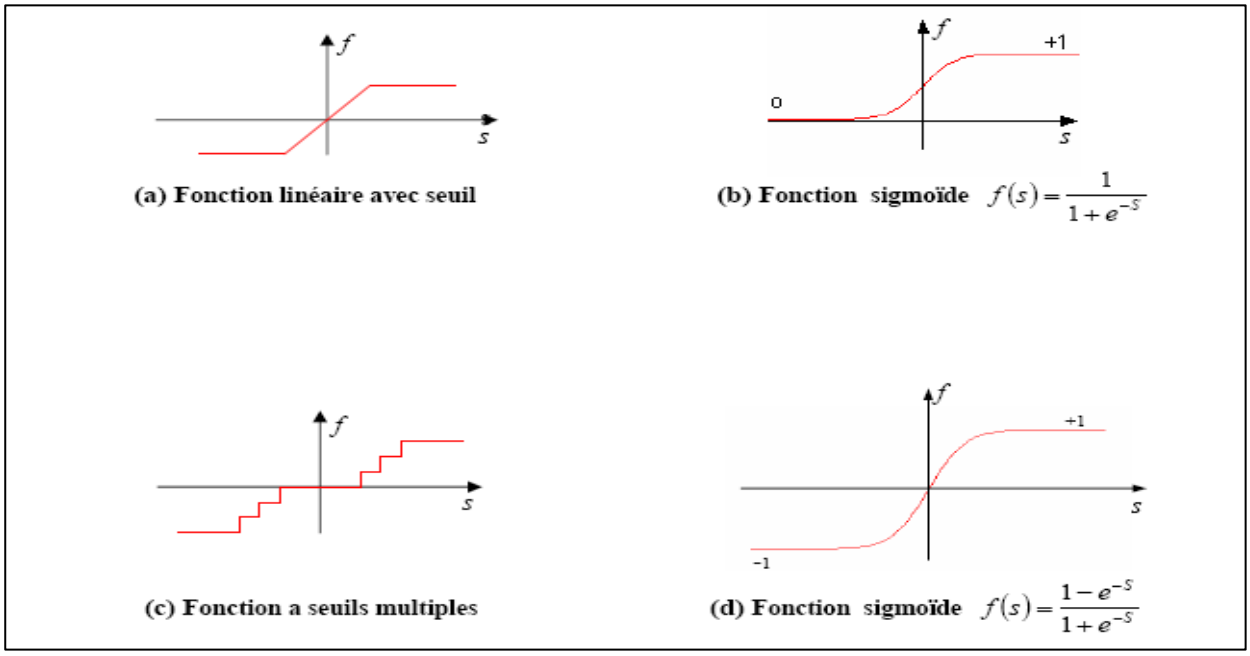
$O_k = f(L_k)$, permet de déterminer l'état interne du neurone en fonction de l'entrée totale, qui sera transmis via la sortie vers d'autres neurones.

9.6 Deuxième définition

Un réseau de neurones est un ensemble de petites unités de calcul appelés « neurones artificiels » interconnectés par des liaisons pondérées appelés « poids synaptiques » et dont l'état est représenté par la valeur de de sa sortie.

9.7 Quelques exemples de la fonction d'activation

La fonction d'activation et un traitement effectué par le neurone de sortie sur la quantité de sommation calculée par le même neurone. Nous en illustrons ci-après quelques exemples.



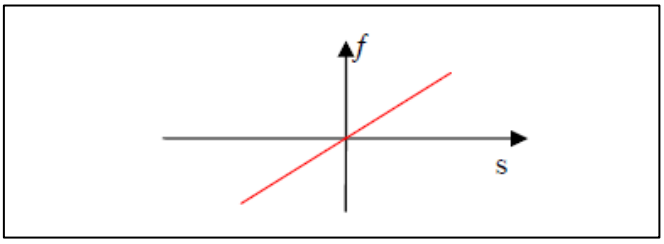
(a) Fonction linéaire avec seuil

(b) Fonction sigmoïde $f(s)=\frac{1}{1+e^{-s}}$

(c) Fonction a seuils multiples

(d) Fonction sigmoïde $f(s)=\frac{1-e^{-s}}{1+e^{-s}}$

(e) Fonction de Heaviside



(f) Fonction linéaire sans saturation

9.8 Propriétés des réseaux de neurones

Les réseaux de neurones artificiels (RNA) possèdent des propriétés très intéressantes qui sont :

1) Parallélisme : Ensemble d'unités de calcul travaillant d'une manière synchrone (simultanément).

2) Capacité d'adaptation **:** Grâce à l'apprentissage permettant de prendre en compte des nouvelles données du monde extérieur.

3) Mémoire distribuée : Dans les réseaux de neurones, la mémoire correspond à une carte d'activation de ces neurones. Cette carte est en quelques sorte un codage du fait mémorisé ce qui attribue à ces réseaux l'avantage de résister au bruit

4) Capacité de généralisation : Les réseaux de neurones peuvent apprendre à retrouver des règles à partir d'une base d'exemples.

9.9 Types des réseaux de neurones

Réseaux bouclés (récurrents) et réseaux non bouclés.

- **Les réseaux de neurones non bouclés**

Sont des réseaux où l'état actuel de chaque neurone ne dépend pas de l'état antérieur de cet neurone. Ce type de réseau est dit sans mémoire ou statique (Voir figure 32)

- **Les réseaux de neurones bouclés**

Ce réseau contient des cycles et l'état actuel de ses neurones dépend de leurs états précédents. Ils sont dits dynamiques. (Voir figure 33)

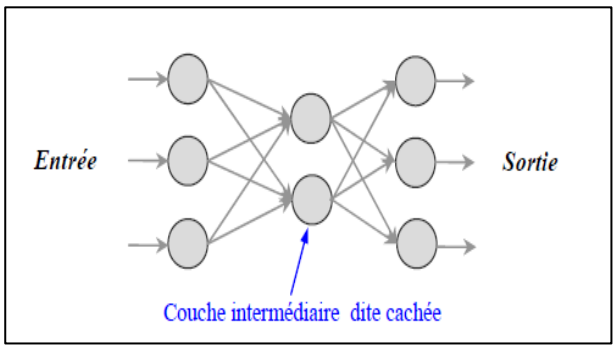


Figure 32 : Réseau de neurones non bouclé

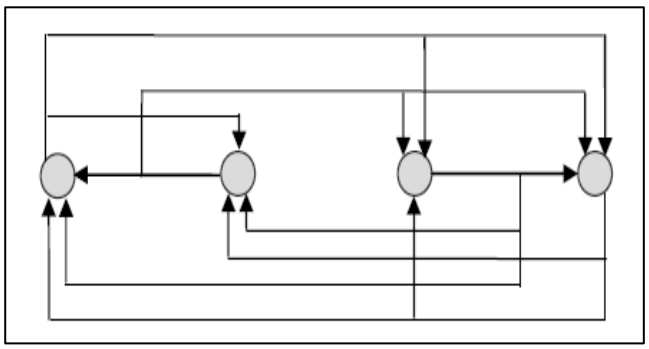


Figure 33 : Réseau de neurones bouclé

Une autre classification des réseaux de neurones est basée sur la notion de couches :

- **réseau de neurones mono couche**.

Une couche d'entrée et une couche de sortie (une couche de perception et une couche de décision ou de sortie).

- **réseau de neurones multi couches** :

Une couche d'entrée, une couche de sortie plus une ou plusieurs couches intermédiaires, dites couches cachées.

9.10 L'apprentissage dans les réseaux de neurones

Dans un réseau de neurones, on peut faire de l'apprentissage supervisé et l'apprentissage non supervisé.

- **L'apprentissage supervisé :**

Consiste à ajuster les poids de connexions W_{ij} progressivement de telle sorte d'avoir les réponses du réseau soient égales (ou du moins les plus proches possibles) aux sorties attendues dites aussi sorties désirées.

- **L'apprentissage non supervisé**

Permet de faire la catégorisation (clustering ou groupage) d'un ensemble de points, objets ou individus. Le résultat de l'apprentissage non supervisé est un ensemble de catégories ou classes.

9.11 Le perceptron mono-couche

Le Perceptron a été développé par Rosenblatt [Rosenblatt 1957] pour résoudre, à l'aide des neurones de Mc Culloch et Pitts (neurones formels) , les problèmes liés à la vision humaine. Ce modèle de réseau de neurones est composé d'une couche d'entrée ou de perception de données et une couche de sortie ou de décision composée d' un ou plusieurs neurones. Voir figure 34 ci-dessous.

La fonction d'activation de ce modèle de réseau de neurones est généralement la fonction Heaviside. L'entrée $x_0 = 1$ est appelée le biais.

Exemple 37

Soit le perceptron mono couche de la figure 34 avec les données suivantes :

Entrées : $x_0 = 1$, x_2 et x_3 ; Poids : $w_0 = -1$, $w_1 = 1$, $w_2 = 1$. Vérifier que ce perceptron implémente la fonction AND (x_1, x_2, x_3) ?

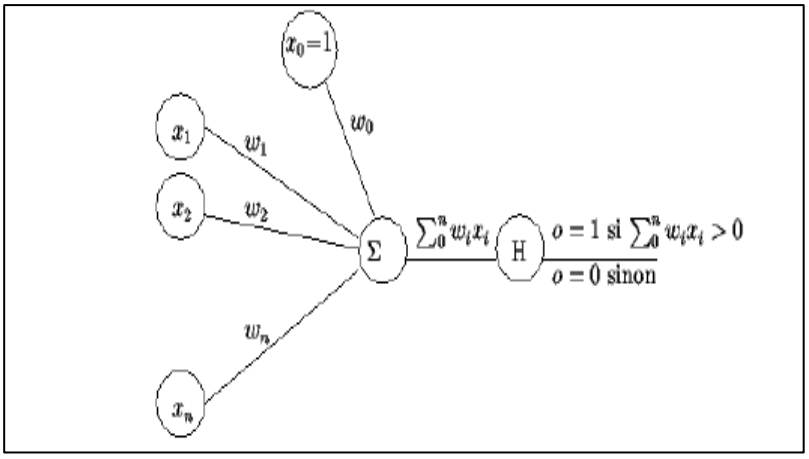


Figure 34 : Le perceptron

Solution

AND (x_0, x_1, x_2) = H (x_0*w_0+x_1*w_1+x_2*w_2) = H(-1+x_1+x_2)

Compte tenu de la fonction Heaviside, On voit bien que la fonction AND est égale 1 si et seulement si x1=x2=1, donc ce perceptron implémente bien la fonction logique AND.

Exemple 38

Soit un perceptron de trois entrées et une sortie (3x1) utilisant une fonction de transfert définie par : f(x) = -1 si x <0 et f(x) = 1 si x >= 0
En supposant que $X_0=1$ et [X1, X2, X3, y] $\in \{0,1\}^4$, Trouver la fonction logique réalisée par ce réseau pour: $\omega_0 = 0.2$, $\omega_1 = -0.2$, $\omega_2 = 0.7$, $\omega_3 = 0.5$. Justifier la réponse ?

Solution

La somme des entrée pondérée : 1*0.2 + x1*0.2+x2*0.2+x3*0.7+x3*0.5

Y= -1 si (1*0.2 + x1*0.2+x2*0.2+x3*0.7+x3*0.5) <0.
Et , Y= 1 si (1*0.2 + x1*0.2+x2*0.2+x3*0.7+x3*0.5) >= 0 , Or Y = -1 n'est pas possible car Y ∈ {0,1} et (1*0.2 + x1*0.2+x2*0.2+x3*0.7+x3*0.5) >0 car [X1, X2, X3] ∈ $\{0,1\}^3$

On en déduit que la fonction logique implémenté par ce perceptron est la fonction tautologie (la sortie du perceptron toujours égale à 1).

9.12 L'apprentissage dans le perceptron mono couche

L'apprentissage dans le perceptron s'effectue avec deux variantes : sans contrôle de l'erreur sur la sortie (Voir algorithme donné par la figure 35) et avec contrôle de l'erreur sur la sortie.

Algorithme apprentissage_ P1C_1
Données
- Vecteur d'entrée Xi = (e_0, e_1…, e_n) ;Vecteur des poids W ={ w_0, w_1,..,w_n} initiaux
- La fonction de transfert T « Signe » ; La variable désirée d_i ; Le pas d'apprentissage μ

Sorties :
- Vecteur des poids W ={ w_0, w_1,..,w_n} mis a jour.

Début
Etape 1 : Initialisation des poids et du biais à des petites valeurs choisies au hasard.
Etape 2 : Présentation d'une entrée X_l = (e_0,e_1, ... e_n) de la base d'apprentissage associé au vecteur de poids W_i = (w_0, w_1…w_n) , l'entrée e_0 = 1 est associé le poids w_0
Etape 3 : Calcul de la sortie du réseau Oi : $Oi = \sum_o^n ei * wi$
Etape 4 : Si la sortie calculée Oi est différente de la sortie désirée di alors modification des poids
$W_i = W_i + \mu.(d_i - O_i).X_i$
Etape 5 : Tant que tous les exemples de la base d'apprentissage ne sont pas traités correctement (i.e.
modification des poids), retour à l'étape 2.
Fin

Figure 35 : algorithme d'apprentissage du perceptron sans contrôle de l'erreur.

Algorithme apprentissage_ P1C_2

Variables :

- Vecteur d'entrée Xi = (e_0, e_1…, e_n)
- Vecteur des poids W ={ w_0, w_1,..,w_n}
- La fonction de transfert T « Signe »
- La variable désirée d_i
- La variable calculée O_i

Données :

- Le tableau d'apprentissage.
- Les valeurs des poids initiaux
- Le pas d'apprentissage µ
- L'erreur acceptée Err

Debut

Etape 1 : Initialisation des poids et du biais à des petites valeurs choisies au hasard.

Etape 2 : Présentation d'une entrée Xl = (e_0,e_1, ... e_n) de la base d'apprentissage associé au vecteur de poids W_i = (w_0, w_1…w_n)

Etape 3 : Calcul de la sortie du réseau O_i : $Oi = \sum_o^n ei * wi$

Etape 4 : Si (Abs (d_i- O_i)) > Err alors modification des poids : Wi = Wi + µ.(di - Oi).X_i

Etape 5 : Tant que tous les exemples de la base d'apprentissage ne sont pas traités correctement (i.e. modification des poids), retour a l'étape 2.

Fin

Figure 36 : Algorithme d'apprentissage du perceptron avec contrôle de de l'erreur.

Exemple 39

Soit le perceptron de la figure. 37 à deux entrées et sans biais. Les conditions initiales **µ** = +1 et les poids initiaux sont nuls. Prendre la fonction de transfert T (x) = 1 si x>0 et T(x)=-1 si x<=0.

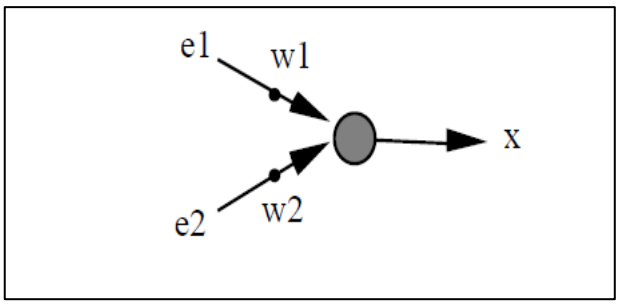


Figure 37: perceptron à deux entrées sans biais

Réaliser l'apprentissage de ce perceptron sans contrôle de l'erreur, en se donnant le tableau d'apprentissage décrit par le tableau 6 ci-dessous, avec nombre d'itérations =2

No. exemple	e1	e2	d
1	1	1	1
2	1	-1	1
3	-1	1	-1
4	-1	-1	-1

Tableau 6 : Données d'apprentissage

Solution

Itération 1 :

Calculons la valeur de O_1 pour l'exemple (1) :

$O_1 = w_1.e_1 + w_2.e_2 = 0* 0 . 1 + 0* 0 . 1 = 0$; $T(0) = -1$

La sortie est fausse, il faut donc modifier les poids en appliquant :

w1 = w1 + e1.(1- (-1)) = 0+1 *2= 2

w2 = w2 + e2. = 0+ 1(1-(1)) = 2

On passe à l'exemple suivant (2) :

$O_2 = 1*2 + 2*-1 = 0$ donc $T(0)= -1$

La sortie est fausse, il faut donc modifier les poids en appliquant :

w1 = 2 + 1(1-(-1)) = 4

w2 = 2 + -1(1-(-1)) = 0

- On passe à l'exemple suivant (3)

$O_3= -1 *4+1*4 = 0$; $T(0)=-1$

Sortie correcte alors pas de modification des poids :

On passe à l'exemple suivant (4)

$O_4= 4*-1+ 0*-1 = -4$; $T(-4) = -1$

Pas de changement de poids car la sortie calculé est égale à la sortie désirée.

Iteration 2

Calculons la valeur de O_1 pour l'exemple (1) :

$O_1 = w_1.e_1 + w_2.e_2 = 4x 1 + 0x 1 = 4$ et $T(4) = 1$

La sortie est correcte, pas de modification de poids.

On passe à l'exemple suivant (2) :

$O_2 = 1*4 + 0*-1 = 4$ et $T(4)= 1$
La sortie est correcte, alors on ne modifie pas les poids.

On passe à l'exemple suivant (3)
$O_3= -1*4+1*0 = -4$; $T(-4)=-1$
Sortie correcte donc pas de modification de poids :

On passe à l'exemple suivant (4)
$O_4= 4*-1+ 0*-1 = -4$; $T(-4) = -1$
Pas de changement de poids car la sortie calculée est égale à la sortie désirée.

On s'arrête à l'itération 2 et l'opération de l'apprentissage s'arrête aussi car aucun changement de poids pour tous les exemples du tableau d'apprentissage.

9.13 Le perceptron multi-couches (PMC)

Le perceptron multi couches est une extension du perceptron mono – couche. Il Possède une couche d'entrée, une couche de sortie et une ou plusieurs couches cachées. Chaque neurone d'une couche donnée est connecté à tous les neurones de la couche précédente et à tous les neurones de la couche suivante. Il utilise principalement les fonctions sigmoïdes. Les fonctions d'activation par couche peuvent être différentes mais elle est la même pour tous les neurones de la même couche. Le P.M.C est un réseau non bouclé (Voir figure 38 ci-dessous).

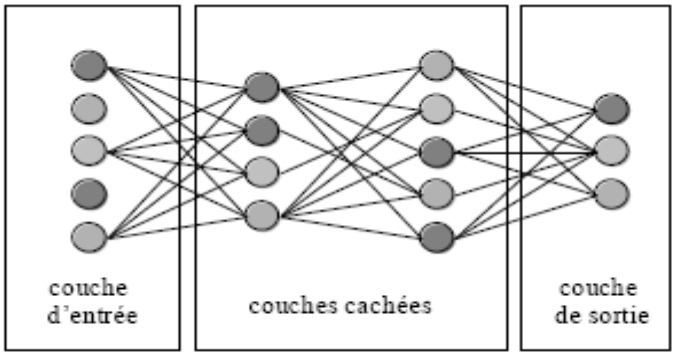


Figure 38 : Le perceptron multi-couches

Cette architecture a permis la classification de données dans les problèmes non linéaires et donc a permis au de dépasser la limite du perceptron mono couche qui ne s'applique qu'aux problèmes de classification linéaire (données linéairement séparables). [Minsky & Papert, 1969]

9.14 L'apprentissage d'un réseau de neurones multi couches

L'année 1986 a vu l'apparition de l'algorithme de rétro -propagation de l'erreur en Anglais « backpropagation » permettant l'apprentissage supervisé du P.M.C , publié par Rumelhart , Hinton et

William [Rumelhart et al. 1986] qui permet d'optimiser les paramètres d'un P.M.C en réponse à un ensemble de données d'apprentissage. La figure 39 donne le principe de cet algorithme d'apprentissage et la figure 40 en donne une version détaillée.

Principe de l'algorithme «rétropropagation » (Backpropagation, en Anglais)
****
/*** x : (x1,x2…,xn) le vecteur d'entrée ****/
/ *** o : (o1,o2,…,om) le vecteur de sortie --- calculé ------ *** /
/ *** t : le vecteur cible (t1,t2,…,tm) ---- attendu ----- *** /
****
Début
Boucle 1 : Répéter jusqu'à ce que le réseau puisse mapper (o à t) d'une manière consécutive pour tous les exemples
Boucle 2 : Pour chaque exemple de la base des exemples, faire :
1. Introduire un exemple x au réseau.
2. Calculer le vecteur de sortie o.
3. Comparer o avec t. Si o est égal à t (ou très proche) alors Aller à Boucle2.
Sinon :
1. Calculer l'erreur sur la couche de sortie et la rétro-propager sur la couche cachée.
2. Ajuster les poids
3. Aller à Boucle2.
Fin

Figure 39 : Principe de l'algorithme d'apprentissage dans le PMC

9.13.2 L'algorithme d'apprentissage Détaillé

/* un réseau avec une seule couche cachée*/
Etape 1: Initialiser les poids W_{ji} et les seuils des neurones à des petites valeurs aléatoires.
Etape 2: Présenter le vecteur d'entrée et de sortie désirés correspondants.
Etape 3: Calculer :
1- La somme des entrées des neurones d'une couche cachée : …
2- Les sorties des neurones de la couche cachée :
3- La somme des entrées de la couche de sortie :
- Les sorties du réseau :
Etape 4: calculer :
1- Les termes de l'erreur pour les neurones de la couche de sortie

$$\delta_k = O_k(1 - O_k)\left(O_k^d - O_k\right)$$

2- Les termes de l'erreur pour les neurones de la couche cachée

$$\delta_h = O_h(1 - O_h).\sum_{k \in sorties} W_{kh} * \delta_k$$

Etape 5 : Ajuster Les poids de la couche de sortie et la couche cachée (prendre $\mu \in [0.01, 0.09]$)

$$\Delta Wji = \mu * \delta j * Xji \; ; \quad Wji = Wji + \Delta Wji$$

Etape 6 : Si la condition sur l'erreur ou sur le nombre d'itération est atteinte, aller à l'étape 7,
Sinon revenir à l'étape 2 jusqu'à la stabilisation du réseau.
Etape 7 : Fin.

Figure 40 : Algorithme d'apprentissage dans le PMC

Exemple 40

Soit le réseau de neurones de la figure 41. Les neurones d'entrée sont X1 et X2, X4 et X5. La couche cachée est constituée d'un seul neurone : X 3 et les neurones de sorties sont : X6 et X7 .
La valeur du facteur d'apprentissage est $\mu = 0{,}05$. La fonction d'activation à appliquer est de la forme suivante :

$$f = \begin{cases} 1 & si \sum_{i=0}^{n} WjiXi > 0 \\ -1 & sinon \end{cases}$$

Avec :
X_{ji} : l'entrée qui provient de l'unité i vers l'unité j.
W_{ji} : le poids du lien entre l'unité i et j.
T_k : la sortie attendue du réseau.
O_k : la sortie obtenue (calculée) du réseau.

Dérouler l'algorithme RETRO_PROPAG pour l'exemple suivant pour une seule itération.

ENTREES : X1=4 ; X2= 2 ; X4=0 ; X5=2
SORTIES : X6= 2 ; X7=1

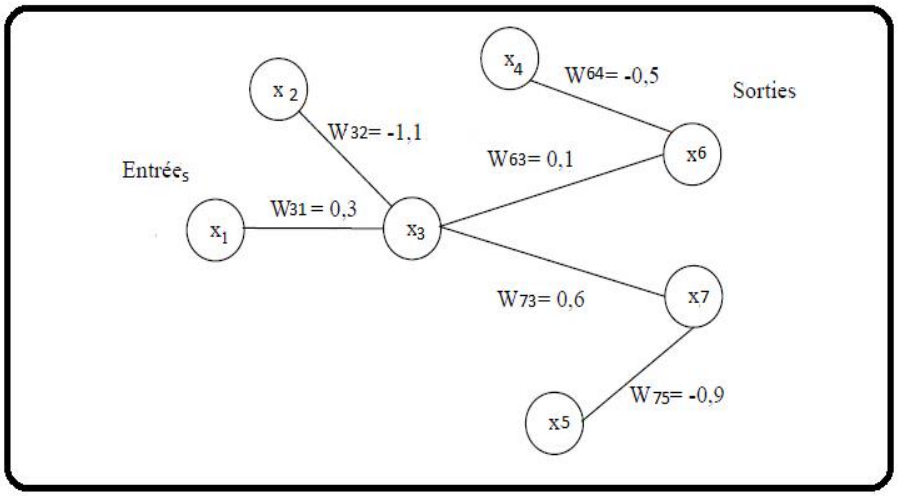


Figure 41 : Un PMC avec une couche cachée

Solution

Calcul de la sortie O_3 du neurone caché :

$O_3 = f(x_2*w_{32}+x_1*w_{31}) = f(1*-1.1 +3*0.3) = f(-0.2) = -1$ car $-0.2 < 0$

Calcul des sorties O_6 et O_7 des neurones de sortie :

O6 = f (x3*w63+x4*w64) = f(-1*0.1+1*-0.5)=f(-0.6) = - 1 car - 0.6 < 0

O7 = f(x 3*w73+x5*w75) = f(-1*0.6+1*-0.9) = f(-0.6-0.9) = f(-1.5) = -1 car -1.5 < 0.

Calcul de l'erreur sur les neurones de sortie

$\delta_6 = O_6(1-O_6)(T_6 - O_6) = -1(1+1)(1+1) = -4$
$\delta_7 = O_7(1-O_7)(T_7 - O_7)$ -1(1 + 1)(1 + 1) = -4

Calcul de l'erreur sur le neurone caché

$\delta_3 = O_3(1-O_3) * (w_{63} * \delta_6 + w_{73} * \delta_7) = -1(1+1)*(0.1*-4 +0.6*-4) = -2(-0.4-2.4) = 5.6$

Ajustement des poids

$\Delta W_{ji} = \mu * \delta_j * x_{ji}$	$W_{ji} = W_{ji} + \Delta W_{ji}$
$\Delta w_{31} = 0.05 * 5.6 * 3 = 0.84$;	$w_{31} = 0.3 + 0.84 = 1.14$
$\Delta w_{32} = 0.05 * 5.6 * 1 = 0.28$;	$w_{32} = -1.1 + 0.28 = -0.82$
$\Delta w_{63} = 0.0.5 * -4 * -1 = 0.2$;	$w_{63} = 0.1 + 0.2 = 0.3$
$\Delta w_{64} = 0.05 * -4 * 1 = -0.2$;	$w_{64} = -0.5 + (-0.2) = -0.7$
$\Delta w_{73} = 0.05 * -4 * -1 = 0.2$;	$w_{73} = 0.6 + 0.2 = 0.8$
$\Delta w_{75} = 0.05 * -4 * 1 = -0.2$;	$w_{75} = -0.9 + (-0.2) = -1.1$

9.15 Le réseau de neurones de Hopfield

Proposé par Hopfield en 1982 [Hopfield, 1982] dans le cadre de ses recherches en physique moléculaire. Est un réseau de neurones complètement connecté (Voir figure 42) . Il y' a une liaison double entre deux neurones telle que $W_{ij} = W_{ji}$. Chaque neurone joue le rôle de neurone d'entrée et de sortie. Ce modèle de réseau ne contient aucune boucle, c'est-à-dire (W_{ii}=0). La fonction d'activation appliquée est de la forme :

$$f(X) = \begin{cases} 1 si X > 0; \\ -1 si X < 0; \\ 0 si X = 0 \end{cases}$$

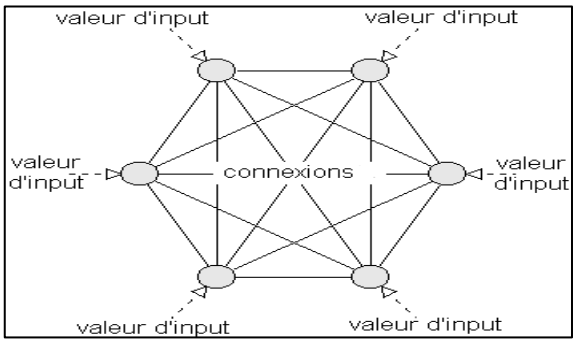


Figure 42 : Réseau de Hopfield à six (6) neurones.

La règle d'apprentissage de Hebb à appliquer permettant de modifier les poids s'écrit de la forme :

$W_{ij}=1/n(\sum_{m=1..n} X_i^m * X_j^m)$; avec $W_{ii} = 0$; n étant le nombre des exemples.

et X^m_i et X^m_j sont les valeurs de la variable X_i et X_j de l'exemple m.

L'état d'un neurone k, c'est-à-dire sa valeur à l'instant t, dépend de l'état des neurones qui lui sont connexes est donné par la relation :

$$X_k(t) = \sum_i X_i(t-1) * W_{ik} = S_k(t)$$

Ainsi l'état du réseau à l'instant t est constitué par tous les états de ses neurones.

$$S(t) = (S_1(t), S_2(t),\ldots, S_n(t))$$

La dynamique du réseau est asynchrone : à chaque instant, un seul neurone qui change d'état.
Le réseau est stable si :

$$S(t) = S(t\text{-}1).$$

Un autre paramètre de ce réseau de neurones qui permet de mesurer le degré de stabilité du réseau est la fonction d'énergie définie par:

$$E(t) = \frac{-1}{2} * \sum_{i,j} W_{ij} * S_j(t) * S_i(t)$$

9.16 Fonctionnement du réseau de Hopfield

Pour un exemple de donnée d'entrée qui correspond à une quantité d'énergie de départ, le réseau évolue vers un état stable qui correspond à une quantité d'énergie minimale. Deux cas de configuration sont possibles:

1) La donnée en entrée fait partie de l'ensemble d'apprentissage, alors cette même donnée est récupérée sur la sortie à la fin (auto- association).

2) Si la donnée en entrée n'appartient pas à l'ensemble d'apprentissage, alors lorsque le réseau se stabilise, le résultat obtenu correspond à un exemple d'apprentissage le plus proche de celle en entrée (généralisation)

C'est pour cette raison que les réseaux de Hopfield sont connus par les mémoires associatives (mémoires avec accès par le contenu)

Exemple 41

Faire l'apprentissage d'un réseau de Hopfield pour la matrice des poids du tableau 7 sachant que l'état initial = (1,1,-1,1) . La fonction d'activation : f(x) = 1 si x>0 et f(x) =-1 sinon.

Solution

On applique la relation :

$$X_k(t) = \sum_i X_i(t-1) * W_{ik} = S_k(t)$$

W_{ij} =

0.0	–0.1	0.1	–0.1
–0.1	0.0	–0.1	0.1
0.1	–0.1	0.0	–0.1
–0.1	0.1	–0.1	0.0

Tableau 7 : Matrice de poids

Nous avons l'état initial du réseau, qui est l'état des quatre neurones au temps t=0.
(X1(0), X2(0), X3(0) , X4(0)) = (1,1,-1,1)
Calcul de l'état des quatre neurones au temps t = 1.
$X_1(1) = X_1(0)*W_{11}+X_2(0)*W_{21}+X_3(0)*W_{31}+X_4(0)*W_{41}$ =
= 1*0+1*-0.1+ (-1)*0.1+1*-0.1 = - 0.3
$X_2(1) = X_1(0)*W_{12}+X_2(0)*W_{22}+X_3(0)*W_{23}+X_4(0)*W_{24}$
= 1*-0.1+1*0+-1*0.1+1*0.1 = -0.1
$X_3(1) = X_1(0)*W_{13}+X_2(0)*W_{23}+X_3(0)*W_{33}+X_4(0)*W_{43}$
= 1*0.1+1*-0.1+-1*0+1*-0.1 = -0.1
$X_4(1) = X_1(0)*W_{14}+X_2(0)*W_{24}$ +X3(0)*W34+X4(0)*W44
= 1*-0.1+1*0.1-1*-0.1+1*0
= 0.1
Donc l'état du réseau au temps t=1 est :
$(X_1(1), X_2(1), X_3(1), X_4(1))$ = (-0.3, -0.1,-0.1,0.1)

L'apprentissage de ce réseau de Hopfield consiste à réitérer cette opération, c'est à dire calculer l'état du réseau au temps t=2, au temps t = 3 … , au temps t= T jusqu'à ce que l'état du réseau se stabilise (les valeurs des quatre neurones ne changent plus). Dans ce cas on dira que le réseau a appris la donnée en entrée représentée par l'état initial.

Chapitre 10 : Les algorithmes génétiques

10.1 Introduction

Les algorithmes génétiques appartiennent à la famille des méthodes évolutionnaires qui sont basées sur le principe de la sélection naturelle et dont le but est de fournir une solution approchée dans un temps raisonnable dans le cas où le problème en main ne possède pas une solution algorithmique précise ou présente une explosion combinatoire.
En 1960, C'est John Holland et son équipe de recherche qui ont commencé à étudier les méthodes inspirées de la nature notamment le principe de la sélection naturelle. Ce dernier stipule que dans une population, les individus qui s'adaptent le mieux à leur environnement ont de fortes chances de survivre et de se reproduire pour donner naissance à de nouvelles générations. Il fallait attendre l'année 1975 [Holland et al. 1975] pour voir apparaitre leur première publication sur le sujet et que l'appellation « Algorithmes Génétiques » ait été introduite pour la première fois dans le glossaire informatique en tant qu'une branche de l'intelligence artificielle.

10.2 Définition

Les algorithmes génétiques permettent la recherche de solutions approchées à des problèmes d'optimisation en un temps raisonnable, lorsqu'il n'existe pas une solution algorithmique précise pour les résoudre ou que la durée de traitement ou de calcul de la solution dépasse l'échelle de la vie humaine

10.3 Principe

Le principe d'un algorithme génétique (voir figure xx) est comme suit :

1- Générer une population d'individus de taille n : x_1, x_2, x_3,…, x_n.
2- Calculer les chances de survie (qualité ou encore fitness) de chaque individu de la population en cours : $f(x_1)$, $f(x_2)$, $f(x_3)$, …, $f(x_n)$.
3- Vérifier si le critère de terminaison est atteint. Si oui, terminer.
4- Sélection d'individus de la population courante pour leur appliquer les opérateurs génétiques.
5- Appliquer ces opérateurs génétiques (croisement +mutation) aux individus sélectionnés.
6- Remplacer les individus produits par les individus de la population précédente et générer la nouvelle population
7- Retourner à l'étape 2.

10.4 Mesures à prendre pour appliquer les algorithmes génétiques

Afin d'appliquer la technique des algorithmes génétiques, il est primordial de répondre aux questions suivantes :

a) Quels sont les individus du problème posé ?

b) Comment coder ces individus ?
c) Comment faire la sélection des individus, autrement dit, comment définir la fonction d'adaptation et la stratégie de sélection ?
d) Comment faire le croisement entre les gènes des individus ?
e) Muter ou ne pas muter les individus et comment ?

10.5 Terminologie relative aux algorithmes génétiques

- **Gène** : caractère génétique
- **Chromosome** : Une séquence de gènes
- **Individu** : Chromosome, représente un point dans l'espace de recherche de solutions.
- **Code** : Définit la méthode de représentation des individus pour pouvoir les manipuler
- **Population** : C'est l'ensemble des individus, ou encore l'ensemble des chromosomes d'une même Génération.
- **Sélection** : opérateur génétique permettant de choisir un ensemble d'individus d'une population
- **Croisement** : opérateur génétique permettant de donner deux enfants à partir d'un couple d'individus.

- **Mutation** : opérateur génétique permettant de modifier un ou plusieurs gènes d'un chromosome

10.6 Validité et cohérence des individus

Selon la méthode de codage et sa signification, on doit être toujours certain que les individus de notre population soient valides. En effet, que ferait-on d'une solution donnant une bonne note de qualité, mais n'ayant aucun sens pratique une fois interprété. Il faut donc s'assurer que la fonction de production des individus crée toujours des individus valides

10.7 Critère de terminaison

Généralement, un algorithme génétique se termine après un certain nombre de générations, mais on peut également terminer l'exécution de l'algorithme lorsqu'une certaine condition soit atteinte, par exemple lorsque la qualité d'un individu dépasse un certain seuil.

10.8 Le codage des individus

L'opération de codage des individus est une opération très importante qui influe sur le processus de recherche de solutions. Il existe trois types de codage d'individus :

• Le codage binaire

C'est le plus utilisé. Chaque gène dispose du même alphabet binaire {0, 1} Un gène est alors représenté par un entier long (32 bits), les chromosomes qui sont des suites de gènes sont représentés par des tableaux de gènes et les individus de notre espace de recherche sont représentés par des tableaux de chromosomes.

- **Le codage réel**

Cela peut-être utile notamment dans le cas où l'on recherche le maximum d'une fonction réelle.

- **Le codage de Gray :**

Dans le cas d'un codage binaire on utilise souvent la "distance de Hamming" comme mesure de la dissimilarité entre deux éléments de la population, cette mesure compte les différences de bits de même rang de ces deux séquences. Et c'est là que le codage binaire commence à montrer ses limites. En effet, deux éléments voisins en termes de distance de Hamming ne codent pas nécessairement deux éléments proches dans l'espace de recherche. Cet inconvénient peut être évité en utilisant un "codage de Gray" : le codage de Gray est un codage qui a comme propriété qu'entre un élément n et un élément n + 1, donc voisin dans l'espace de recherche, un seul bit diffère.

10.9 Les opérateurs génétiques

- La Sélection

Fondé sur la théorie de la sélection naturelle. Elle définit quels seront les individus de P qui vont être dupliqués dans la nouvelle population P'. les individus les plus aptes à répondre à certains critères seront sélectionnés. Si n est le nombre d'individus de P, on doit en sélectionner n/2. On distingue différentes méthodes de sélection

- **Méthode de la "loterie biaisée**" (roulette Wheel)

Où chaque individu a une chance d'être sélectionné proportionnelle à sa performance. La performance de chaque individu se mesure par la fitness de même individu divisé par la somme des fitness de tous les individus de la population.

- **La Méthode élitiste**

On trie de manière décroissante la population P selon la fitness de ses individus puis on prend les n meilleurs individus.

- **La Sélection par Tournois**

On effectue un tirage avec remise de deux individus de P, et on prend celui qui a la fitness la plus élevée avec une probabilité p comprise entre 0.5 et 1, et on répète n fois.

- Le Croisement

Consiste à combiner deux individus quelconques (dits parents) pour obtenir deux autres individus (dits enfants). On coupe en un ou plusieurs points deux individus (aux mêmes endroits dans les deux individus) et on échange les parties situées entre ces points. L'opérateur de croisement est appliqué avec un taux de croisement « Pc » généralement entre 0.25 et 0.75.

On distingue trois types de croisement :

- **Croisement en un point**

On choisit au hasard un point de croisement, pour chaque couple (le croissement s'effectue au niveau binaire).

- **Croisement en deux points**

On choisit au hasard deux points de croisement et on permute les gènes des deux individus qui sont entre les deux points.

- **Croisement multi - points**

On choisit au hasard des points de croisement (3 points ou plus) et on permute les gènes des deux individus qui sont entre les points.

- **La Mutation**

Est la modification aléatoire d'un paramètre du patrimoine génétique (l'inversion d'un bit dans un chromosome). Les mutations empêchent l'évolution de se figer. La probabilité de mutation « Pm » est très faible, comprise entre 0.01 et 0.001.

Exemple 42

En appliquant la technique d'algorithme génétique, trouver l'entier x de l'intervalle

[0 16[qui maximise la fonction :

$$f(x) = \frac{1}{4}\left|15x^2 - x^3\right| + 4$$

Commencer par une population de 4 individus arbitraires, codés chacun sur 4 bits. La sélection des individus se fait par élitisme, le croisement en 1 point et la mutation est aléatoire. Les fils remplacent les individus faibles de la population en cours. Arrêter les calculs à la 2eme génération.

Solution

Dans ce problème d'optimisation les individus sont les solutions recherchées x qui maximisent la fonction f. De ce fait, la fonction fitness est la même fonction f et donc la fitness de l'individu x est f(x).

- Population initiale de la 1ere génération P0 = {4, 0, 14, 1}.

- Les individus sont définis aléatoirement dans l'intervalle [0 16[.

- Codification de P_0 sur 4 bits P_0= {0100, 0000, 1110, 0001}.

- Evaluation de P_0

Pour évaluer la population P0, on procède au calcul de la fitness de chaque individu de cette population en mettant en application la fonction fitness f(x). Le tableau 8 donne cette évaluation.

Étiquette du Chromosome	Chaîne du chromosome	Entier décodé	Qualité (*fitness*)
X1	0100	4	48
X2	0000	0	4
X3	1110	14	53
X4	0001	1	7,5

Tableau 8 : Évaluation de la 1ere génération (P_0)

- Sélection par élitisme : les deux meilleurs individus sont : X1 = 1000 et X3 =1110
- Croisement en un point (point 2)

  Parents 1000 ⊗ 1110 → 1010, 1100 2 enfants

- Mutation : par exemple, Inversion du dernier bit de chaque individu enfant :

  Enfant1 : X5 1010 → 1011 enfant1 muté
  Enfant2 : X6 1100 → 1101 enfant2 muté

- Génération de la population P_1 = {1010,1100, 1011, 1101}
- Évaluation de P1 : donnée par le tableau 9 ci-dessous.

Étiquette du Chromosome	Chaîne du chromosome	Entier décodé	*Fitness* F(Xi)
X1	1010	4	48
X3	1100	14	53
X5	1011	11	125
X6	1101	13	88.5

Tableau 9 : Évaluation de la 2eme génération (P_1)

- Critère de terminaison atteint en 2emegénértaion (P_1) donc arrêt. La meilleure solution dans cette génération est X5.

Exemple 43

Appliquer le principe des algorithmes génétiques pour résoudre le problème du voyageur de commerce pour n= 5 villes, notées A, B, C, D, E. Le tableau 10 donne les distances entre ces cinq villes. Prendre P_0 constitué de 4 individus, faire la sélection par élitisme, le croisement en deux

points , la mutation d'une manière aléatoire et les fils remplacent les individus faibles .S'arrêter à la 3eme génération.

	A	**B**	**C**	**D**	**E**
A	0	100	50	30	25
B		0	60	75	20
C			0	150	45
D				0	35
E					0

Tableau 10 : Matrice des distances entre villes

Solution

Au point de vue théorique, il y'a ((n-1)!)/2 possibilités, c'est-à-dire 0.5 * (5-1) ! = 12 possibilités. Le nombre de solutions possibles augmentent d'une manière exponentielle avec l'augmentation du nombre de villes et on est donc devant un problème d'explosion combinatoire (A titre d'exemples, pour n= 10 villes nous avons 181 440 possibilités, et pour n=15, nous avons 87 178 291 200 possibilités). Sa complexité est de l'ordre de : O(n !), « n » étant le nombre de villes. Il n'existe pas un algorithme de complexité polynomiale pouvant résoudre ce problème dans un temps raisonnable. . Pour éviter ce problème d'explosion combinatoire, on cherche une solution optimale (acceptable dans un temps raisonnable) en utilisant le principe des algorithmes génétiques.

- Première génération P_0 = {ABCDEA, CBDAEC, BDACEB, ADBCEA}

P_0 est donnée aléatoirement, constituée d'individus qui sont des trajets possibles. Un trajet doit passer une seule fois par toutes les villes et retourner à la ville de départ.

- La fonction d'adaptation est f = 1/ d avec d : la distance du trajet. Plus la distance est petite, plus la fonction fitness est grande, plus le trajet est meilleur.

- Evaluation de P_0 :donnée par le tableau 11 ci-dessous :

Individu (I)	Distance parcourue d (I)	Fitness individu. f (I)= 1/d(I)
ABCDEA	100+60+150+35+25= 370	0.0027
CBDAEC	60+75+30+25+45= 235	0.0042
BDACEB	75+30+50+45+20= 220	0.0045
ADBCEA	30+75+60+45+25 = 235	0.0042

Tableau 11 : Evaluation de P_0

- Sélection par élitisme : {CBDAEC, BDACEB}

- Croisement en deux points (point 1 et point 3)

Parents CBDAEC ⊗ ADBCEA → CDBAEC, ABDCEA **2** enfants

- Mutations ABCDEA → ACBDEA enfant1 muté
  BDACEB → BDAECB enfant2 muté

- Génération de la deuxième génération (P_1) : {CBDAEC, BDACEB, ACBDEA, BDAECB}

- Evaluation de P_1 : donnée par le tableau 12 ci-dessous.

Individu (I)	Distance parcourue d (I)	Fitness individu. f (I)= 1/d(I)
CBDAEC	60+75+30+25+45= 235	0.0042
BDACEB	75+30+50+45+20= 220	0.0045
ACBDEA	50+60+75+35+25= 245	0.0040
BDAECB	75+30+25+45+60= 235	0.0042

Tableau 12 : Evaluation de P_1

- Sélection par élitisme : {BDACEB, BDAECB}

- Croisement en deux points (point 1 et point 3)
  Parents BDACEB ⊗ BDAECB → BDACEB, BDAECB 2 enfants identiques aux parents.

- Mutation BDACEB → BDCAEB enfant1 muté
  BDAECB → BDEACB enfant2 muté

Génération de la troisième génération (P_2) : {BDACEB, BDAECB, BDCAEB, BDEACB}

- Evaluation de P_2 donnée par le tableau 13 ci –dessous.

Individu (I)	Distance parcourue d (I)	Fitness individu. f (I)= 1/d(I)
BDACEB	75+30+50+45+20= 220	0.0045
BDAECB	75+30+25+45+60= 235	0.0042
BDCAEB	75+150+50+25+20= 320	0.0031
BDEACB	75+35+25+50+60= 245	0.0040

Tableau 13 : Evaluation de P_2

Le critère d'arrêt qui est la troisième génération est atteint donc on s'arrête. La meilleure solution (trajet) pour la troisième génération est : **BDACEB.**

Chapitre 11 : Algorithme de colonies de fourmis

11.1 Introduction

L'algorithme de colonie de fourmis est inspiré du comportement d'une colonie de fourmis pour la recherche collective de la nourriture, proposé à l'origine par Marco Dorigo et ses co-auteurs [Marco et al. 1990][Marco et al. 2006] pour la recherche de chemins optimaux dans un graphe.
Des biologistes ont observé, dans une série d'expériences menées à partir de 1989, qu'une colonie de fourmis ayant le choix entre deux chemins d'inégales longueurs menant à une source de nourriture avait tendance à utiliser le chemin le plus court. En effet, cette tendance à emprunter le chemin le plus court est motivée comme suit (Voir figure 43 ci-dessous).

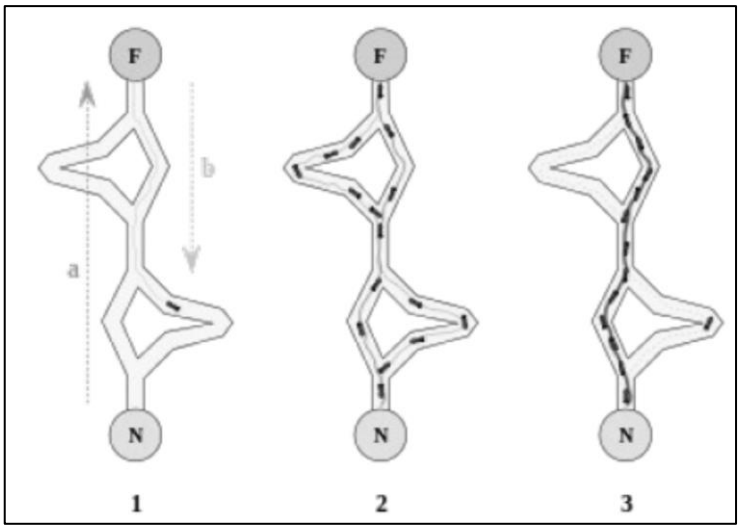


Figure 43 : Colonie de fourmis et choix entre chemins

1) La première fourmi trouve la source de nourriture (F), via un chemin quelconque (a), puis revient au nid (N) en laissant derrière elle une piste de phéromone (b). **2)** Les fourmis empruntent indifféremment les quatre chemins possibles, mais le renforcement de la piste rend plus attractif le chemin le plus court. **3)** Les fourmis empruntent le chemin le plus court, les portions longues des autres chemins perdent leur piste de phéromones**.**

Ce comportement peut être expliqué comme suit :
1. Une fourmi (appelée « éclaireuse ») parcourt plus ou moins au hasard l'environnement autour de la colonie ;
2. Si celle-ci découvre une source de nourriture, elle rentre plus ou moins directement au nid, en laissant sur son chemin une piste de phéromones ;
3. Ces phéromones étant attractives, les fourmis passant à proximité vont avoir tendance à suivre, de façon plus ou moins directe, cette piste ;
4. En revenant au nid, ces mêmes fourmis vont *renforcer* la piste ;

5. Si deux pistes sont possibles pour atteindre la même source de nourriture, celle étant la plus courte sera, dans le même temps, parcourue par plus de fourmis que la longue piste ;
6. La piste courte sera donc de plus en plus renforcée, et donc de plus en plus attractive ; la longue piste, elle, finira par disparaître, les phéromones étant volatiles ;
7. A terme, l'ensemble des fourmis a donc déterminé et « choisi » la piste la plus courte.

11.2 Définition :

L'algorithme de colonie de fourmis est un algorithme d'optimisation permettant de trouver le chemin le plus court entre deux points d'un graphe. Il est inspiré de la méthode utilisée par une colonie de fourmis pour rechercher le chemin le plus court reliant le nid de la colonie à une source de nourriture.

11.3 Principe de recherche de nourriture par une colonie de fourmis (Le modèle naturel)

a) En sortant de nid, chaque fourmi cherche aléatoirement une source de nourriture dans l'environnement non lointain du nid.
b) Si une fourmi trouve une source de nourriture, elle revient à son nid en laissant sur son chemin de retour une substance chimique appelée phéromone.
c) Cette substance chimique attire vers elle les fourmis qui sont à proximité et qui vont à leur tour le renforcer en y déposant une quantité de phéromone.
d) Puisque le phéromone s'évapore, les longues pistes finiront par disparaître, seule la piste la plus courte (entre le nid et la source de nourriture) reste car suivie par presque toute la colonie..
e) A terme du processus, l'ensemble des fourmis de la colonie a donc déterminé d'une manière collaborative et par stigmergie (communication locale et indirecte de l'information) la piste la plus courte qui conduit à la source de nourriture.

8.2 Modélisation mathématique du comportement de la colonie de fourmis pour la recherche de la nourriture (Le modèle artificiel)

- **La règle de déplacement d'une fourmi k**

Est gérée par une fonction de transition aléatoire $\mathbf{P_{ij}^{k}(t)}$ qui à l'itération t , donne la probabilité que la fourmi k , initialement au point i, se déplace au point j. Elle est définie de la manière suivante :

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{\tau_{ij}(t)^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{l \in J_i^k} \tau_{il}(t)^{\alpha} \cdot \eta_{il}^{\beta}} & \text{si } j \in J_i^k \\ 0 & \text{si } j \notin J_i^k \end{cases}$$

(Relation 1)

- $J_k(i)$: l'ensemble des points à visiter par la fourmi k , en étant au point i.

- η_{ij} : Souhait d'ajouter l'arc (i,j) au chemin d'une fourmi, en étant au point i.
- $\tau_{ij}(t)$: quantité de phéromone sur l'arc (i,j) lors de l'itération t.
- α : coefficient d'intensité de phéromone.
- β : coefficient de visibilité des points j

- **Règle de dépôt de phéromone par une fourmi k**

La quantité déposée sur l'arc (i,j) du trajet de la fourmi k est donnée par la formule :

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \frac{Q}{L^{k}(t)} & \text{si } (i,j) \in T^{k}(t) \\ 0 & \text{si } (i,j) \notin T^{k}(t) \end{cases}$$

Relation (2)

Tels que :

Q : est un paramètre fixe
$T^k(t)$: Le trajet de la fourmi k lors de l'itération t
$L^k(t)$: La longueur du trajet de la fourmi k lors de l'itération t.

- **Règle d'évaporation de la phéromone**

La quantité de phéromone sur l'arc (i, j) du trajet de la fourmi k pour l'itération suivante (t+1) est mise à jour selon la relation mathématique suivante :

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t)$$

Relation(3)

ρ: Taux d'évaporation de la phéromone.
$\tau_{ij}(1)$: quantités initiales de phéromone sur les arcs (i,j) , >= 0 , et non toutes nulles.

11.5 Pseudo code de l'algorithme de colonies de fourmis

Pseudo-Algo –ACF

Lecture (données) ;
Lecture (paramètres de l'ACF)
Pour t=1 …. Tmax
Pour chaque fourmi k=1 … m
Choisir un nœud au hasard
Pour chaque nœud non visité
Choisir un nœud j de la liste J_i^k (relation (1))
Finpour
Déposer une quantité de phéromone $\Delta \tau_{ij}^k(t)$ sur le trajet $T^k(t)$ (relation (2))
Finpour
Evaporer la phéromone selon (la relation (3))
Finpour
Sortir (trajet optimal)
Fin Pseudo

Exemple 44

Le graphe ci- après dans la figure 44 montre les chemins possibles avec distances reliant le nid d'une colonie de fourmis (N) à une source de nourriture (F). Dérouler l'algorithme de colonie de fourmis pour trouver le chemin le plus court entre N et F sous les hypothèses suivantes :

- Nombre de fourmis m = 2 ;
- Q = 1 ; ρ = 0.5 ;
- Phéromone initial sur toutes les arêtes du graphe = 0.4
($T_{NA}(0) = T_{AB}(0) = T_{BF}(0) = T_{NC}(0) = T_{CD}(0) = T_{DF}(1) = T_{AD}(0) = 0.4$)

- Critère d'arrêt : Une (1) itération ;
- $\alpha = \beta = 1$.

Solution

Itération : 1

Fourmi k=1

- **Le trajet Aller**

A la sortie du nid, la fourmi « 1 » a deux chemins possibles à prendre NA ou NC. Calculons la probabilité sur chaque chemin.

P_{NA}= 0.4 x (1/15)/(0.4x(1/15)+0.4x(1/15))= 0.5 , P_{NC}=0.4 x (1/15)/(0.4x(1/15)+0.4x(1/15))= 0.5 Les deux probabilités sont égales, les deux chemins ont les mêmes chances d'être pris.

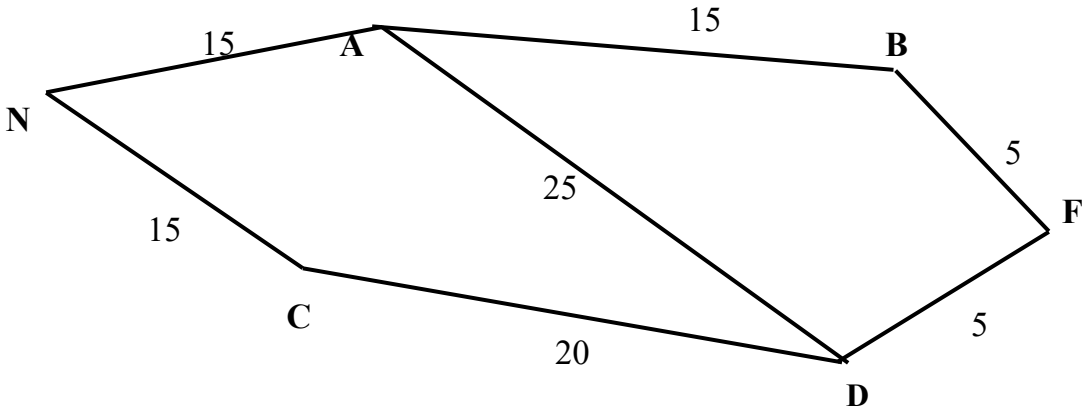


Figure 44 : Graphe d'exploration de la source de nourriture

Supposons que la fourmi « 1 » choisit NA.

Au point A, la fourmi « 1 » a deux possibilités AB ou AD :
P_{AB} = 0.4 x (1/15)/ (0.4 x (1/15) +0.4x (1/20)) = (1/15) / (1/15 +1/20)= (1/15)/(7/60)=60/105=0.57
P_{AD} = (1/20)/(7/60) = 1/20 x 60/7 = 60/140 = 3/7=0.42 , puisque P_{AB}> P_{AD} donc la fourmi « 1 » se dirige vers le point B. De B , la fourmi « 1 » trouve la source de nourriture F.

Le trajet Aller de la fourmi « 1 » est : N— A— B— F

- **Trajet Retour**

En F , la fourmi « 1 » a deux possibilités FB ou FD avec les mêmes probabilités = 0.5 , puisque même distance (5m) et même taux de phéromone 0.4. Supposons qu'elle prend le FB. De B elle prendra BA sûrement, et de A elle aura deux possibilités AN ou AD, calculons les deux probabilités.
P_{AN} = (0.4 x 1/15)/ (0.4 x (1/15) + 0.4 x (1/20)) = 0.57
P_{AD} = (0.4 x 1/20)/ (0.4 x (1/15) + 0.4 x (1/20)) =0.43

Le trajet de retour de la fourmi « 1 » est donc : F—B—A—N

Ajout de la phéromone sur **le trajet de retour** de la fourmi « 1 » : F—B—A—N
Longueur du trajet de la fourmi (NA+AB+BF) = (15+15+5)= 35 m Donc la quantité déposée par la Fourmi « 1 » sur son trajet est =1 /35 sur chacune des arrêtes du chemin de retour.

Fourmi k= 2

- **Le trajet Aller**

P_{NA}= (0.4+ 1/35) x (1/15) / ((0.4+1/35)x(1/15)+ 0.4x (1/15)) =0,41x0,06 /0,41x0,06+0,4x0,06=41/81=0,516
P_{NC}= (0.4 x (1/15) / ((0.4+1/35)x(1/15)+ 0.4 x(1/15))= 0,4 / 0,828 = 0.484 . Puisque P_{NA} > P_{NC} Donc la fourmi « 2 » choisit NA. De A , la fourmi « 2 » a deux directions possibles AB ou AD.
P_{AB} = (0.4+ 1/35)x (1/15) / ((0.4+1/35)x(1/15)+ 0.4x(1/20)) = 0.428x0,066 /(0.428x0,066 +0,02)=0,028/(0,028+0,02)
= 0,028/0,048 =0.58
P_{AD} = (0.4)x (1/20) / ((0.4+1/35)x(1/15)+ 0.4x(1/20)) = 0,020/0,048 =0,42

Puisque $P_{AB} > P_{AD}$, alors de A, la fourmi « 2 » se dirige vers B. De B elle se dirige sûrement vers la source de nourriture F.
Le trajet Aller de la fourmi « 2 » est : N— A— B— F

- **Trajet Retour**

P_{FB} = (0.4+ 1/35)x (1/5) / ((0.4+1/35)x(1/5)+ 0.4x(1/5) =0,428x0,2/(0,428x0,2+0,08)=0,085/0.165 =0.51
P_{FD} =0.49. Donc la fourmi « 2 » choisit B. de B elle choisira sûrement A. Au point A, la fourmi « 2 » aura à choisir entre N et D. Calculons les probabilités correspondantes:
P_{AN} = (0.4+ 1/35) x (1/15) / ((0.4+1/35)x(1/15)+ 0.4x(1/20)) =0,028/(0,428x0,03)+0,02)= 0,028/0,032=0,87
P_{AD} = (0.4 x 1/20) / ((0.4+1/35)x(1/15)+ 0.4x(1/20)) =0.13. Donc de A , la fourmi « 2 » se dirige vers son nid N.
Le trajet de retour de la fourmi k=2 et le même que la fourmi k=2 : F—B—A—N

- **Ajout de la phéromone par la fourmi « 2 »:** ajout de 1/35 sur toutes les arêtes du trajet de retour

- **Évaporation de la phéromone**

$T_{NA}(1) = T_{AB}(1) = T_{BF}(1)$= (1-0.5)*0.4 +2/35 = 0.20+0.57=0.77
$T_{NC}(1) = T_{CD}(1) = T_{DF}(1) = T_{AD}(1)$= (1-0.5)*0.4 = 0.20

- Fin de l'itération 1 …… Arrêt (car critère d'arrêt atteint)

Le chemin le plus court est : N—A—B---F , sa longueur est de 35 m.

Chapitre 12 : L'intelligence artificielle Distribuée (I.A.D)

12.1 I. A et I.A.D

Le domaine de l'Intelligence Artificielle (I.A) cherche surtout à décrire et à résoudre des problèmes complexes identifiés par des experts. Dans ce domaine, il est possible de construire des programmes informatiques, capables d'exécuter un nombre important de tâches en centralisant « l'intelligence » au sein d'un système unique [Erceau & Ferber, 1991]. Il est cependant difficile d'entrer dans une même « base », les connaissances et les compétences d'individus totalement différents qui communiquent entre eux.

L'apport de l'Intelligence Artificielle Distribuée (I.A.D) permet de « distribuer l'intelligence » entre plusieurs entités. Comme le souligne Bond et Gasser [Bond et Gasser, 1988]. L'intelligence Artificielle Distribuée présente les avantages suivants :

- L'IAD est bien adaptée à la distribution de problèmes spatiaux, logiques…
- L'héritage de modules permet aux différentes parties du système de développer de façon indépendante un système continu et extensible,
- Les processus distribués entre différents entités de calcul augmentent la vitesse de traitement et de raisonnement,
- Le contrôle du processus local peut être isolé ou séparé du système,
- Dans certain cas, les systèmes confèrent aux entités individuelles des ressources limitées pour résoudre les problèmes et la coopération et la coordination sont essentielles à la résolution de ces problèmes.

Une des branches de l'IAD est les Systèmes Multi-Agents (S.M.A), permet d'introduire dans un système, un ensemble d'individus, dits agents, dotés de connaissances, d'intentions, de capacités de communication entre eux pour des fins de coopération et de collaboration et éventuellement de capacités d'apprentissage et d'évolution.

12.2 Définition d'un agent

Pour certains auteurs [Wooldridge and Jennings 1995] [Jennings et al. 1998] [Weiss, 1999], un agent est défini comme étant une entité logicielle d'un système informatique qui possède les caractéristiques suivantes :

- **Autonomie :** les agents contrôlent leurs actions et leurs états internes. Le système dans son ensemble est capable de réagir sans l'intervention d'un humain ou d'un autre agent. Il n'y a pas de définition unique du terme agent, par contre, il y a un consensus général pour considérer l'autonomie comme notion centrale de l'agent.

- **Flexibilité :** le système doit être un système dans lequel (i) les agents perçoivent l'environnement et peuvent répondre dans le temps requis aux changements que celui-ci peut entraîner sur les agents (ii) les

agents prennent en considération leur comportement général pour permettre de prendre les initiatives appropriées aux changements de l'environnement (iii) les agents interagissent avec les autres agents afin d'accomplir leurs buts. Pour cela les agents ont donc les caractéristiques suivantes :

- **Réactivité** : ils perçoivent leur environnement et réagissent aux changements qui s'y produisent dans le temps requis ;

- **Proactivité** : ils exhibent un comportement proactif et opportuniste pour ne pas agir uniquement par réaction à leur environnement mais prendre des initiatives selon leurs buts individuels ;

- **Sociabilité** : ils sont capables d'interagir les uns avec les autres, veut dire avec leur environnement au sens large du terme quand la situation l'exige , afin d'accomplir leurs tâches ou d'aider les autres agents à accomplir leurs buts.

Pour Jacques Ferber [Ferber, 1995] , l'agent est «Une entité physique ou abstraite qui est capable d'agir sur elle-même et son environnement, qui dispose d'une représentation partielle de son environnement, et qui, dans un Système Multi-Agent, peut communiquer avec d'autres agents, et dont le comportement est la conséquence de ses observations, de sa connaissance et des interactions avec les autres agents ». Pour Ferber [Ferber, 1995], la distinction cognitif/réactif « définit un axe pratique d'évaluation de la capacité des agents à accomplir individuellement des tâches complexes et à planifier leurs actions. »

12.3 Types d'agents

Compte tenu de l'architecture interne des agents, on distingue trois (03) types d'agents :

- **Agent réactif**

Ce type d'agent est inspiré de la vie artificielle et possède un cycle de type perception-action. Un agent réactif n'a pas une représentation de soi-même ni de son environnement mais agit uniquement en fonction de ses perceptions. Il manifeste donc aucun comportement intelligent mais réalise uniquement un mapping entre sa fonction sensorielle et l'action à prendre.
Une communauté d'agents réactifs interagissant entre eux selon des règles locales et simples peut exhiber un comportement intelligent collectif connu par « émergence de comportement ».

Un exemple d'agents réactifs est celui du système MANTA (Modeling an ANThill Activity) dû à Drogoul [Drogoul, 1993] pour simuler une communauté de fourmis. « Dans ce système, l'architecture d'un agent comporte les opérateurs de perception, de sélection et d'activation qui manipulent un ensemble de tâches.

- **Agent cognitif ou délibératif**

Un agent de ce type possédé un cycle perception- délibération –action. On dit aussi que ces agents sont intentionnels car ils possèdent des buts et des plans explicites leurs permettant d'accomplir leurs buts. Le résultat de la planification est défini alors comme suit ; étant donné un but et un état courant, il s'agit de trouver une séquence d'actions qui permettent d'aller de l'état courant au but. Cette séquence d'actions est appelée plan. Contrairement aux agents réactifs, ce type d'agents a la capacité de raisonner sur des

représentations du monde, de mémoriser des situations, de les analyser, de prévoir des réactions possibles à toute action, d'en tirer des conduites pour les évènements futurs et donc de planifier son propre comportement [Ferber, 1995]. Un exemple de ce type d'architecture est l'architecture BDI [Bratman 1987]

- **Agent Hybride**

Un agent hybride permet d'allier un comportement réactif et un comportement délibératif. L'architecture de ce type d'agent est conçue sous forme de couches inter –agissantes :

- couche inférieure : au plus bas niveau de l'architecture, il y'a habituellement une couche purement réactive qui prend ses décisions en se basant sur des données brutes en provenance de l'environnement.
- La couche intermédiaire : fait abstraction des données brutes et travaille plutôt avec une vision qui se situe au niveau des connaissances de l'environnement
- La couche supérieure se charge des aspects sociaux de l'environnement, c'est-à-dire du raisonnement tenant compte des autres agents [Chaib-Draa et al., 2001]. Un exemple de type d'architecture est l'architecture InteRRaP [Fischer et al 995].

12.4 Définition d'un système multi agents

Un système multi agents est comme défini par J. Ferber [Ferber, 1995]

1. Un Environnement E, c'est-à-dire un espace disposant d'une métrique.
2. Un ensemble d'objets O. Ces objets sont situés, c'est- à- dire que, pour tout objet, il est possible, à un moment donné, d'associer une position dans E. Ces objets sont passifs, c'est à dire qu'ils peuvent être perçus, créés, détruits et modifiés par les agents.
3. Un ensemble A d'agents qui sont des objets actifs du système.
4. Un ensemble de relations R qui unissent des objets (et donc des agents) entre eux.
5. Un ensemble d'opérations Op permettant aux agents A de percevoir, produire, consommer, transformer et manipuler des objets de O.
6. Des opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification, que l'on appellera les lois de l'Univers.

12.5 Types d'interactions entre agents

Dans un système multi agents, l'interaction entre les agents est un concept clé dans le paradigme multi agents qui permet aux agents d'interagir via un langage de communication dans le but de résoudre collectivement un problème de nature complexe. Autrement dit, La résolution distribuée d'un problème complexe est le résultat surtout de l'interaction coopérative entre les différents agents constituant le système. Les relations d'interactions entre agents sont de natures différentes et varient en fonction des buts, des compétences de ces agents ainsi que les ressources offertes ou disponibles J. Ferber [Ferber 1995] les résument dans le tableau 14 ci-dessous.

Buts	Ressources	Compétences	Type interactions	Catégorie
compatibles	suffisantes	suffisantes	indépendance	indifférence
compatibles	suffisantes	insuffisantes	collaboration simple	coopération
compatibles	insuffisantes	suffisantes	encombrement	coopération
compatibles	insuffisantes	insuffisantes	collaboration coordonnée	coopération
incompatibles	suffisantes	suffisantes	compétition individuelle pure	antagonisme
incompatibles	suffisantes	insuffisantes	compétition collective pure	antagonisme
incompatibles	insuffisantes	suffisantes	conflit individuel pour des ressources	antagonisme
incompatibles	insuffisantes	insuffisantes	conflit collectif pour des ressources	antagonisme

Tableau 14 : Interactions entre agents [Ferber 95]

Lorsque les agents sont en situation de conflit, ils entrent en négociation pour résoudre cette situation. Il existe plusieurs méthodes de négociation mais les plus utilisées sont la technique de l'appel d'offres, et la technique de la vente aux enchères. Il faut noter que toute interaction entre agents est contrôlée par un ensemble de règles définissant l'initiation, le déroulement et la terminaison de cette interaction, cet ensemble de règles est dit protocole d'interaction.

Exemple : Le protocole Contract-Net

, le protocole Contract-Net [Smith 1980] définit un mécanisme de négociation entre deux types d'agents : Contractant et gestionnaire. Il est le protocole d'interaction le plus utilisé dans les SMA. Il repose sur un mécanisme d'allocation de tâches régi par le protocole d'appel d'offres qui est utilisé dans les organisations humaines. Le modèle est basé sur une communication par envoi de messages. Un agent peut jouer un de deux rôles : Initiateur ou Participant. Dès qu'il est invoqué, l'agent initiateur envoie un call-for-proposal à un agent participant. Avant une date limite (deadline) donnée, l'agent participant peut émettre à l'agent initiateur une proposition (propose), refuser de soumettre une proposition (refuse), ou lui indiquer qu'il n'a pas compris (not-understood). La proposition formulée par l'agent participant peut être acceptée ou rejetée par l'agent initiateur. Quand il reçoit un avis d'acceptation de sa proposition (accept - proposal), l'agent informe l'agent initiateur de l'exécution de la proposition.

12.6 L'organisation des agents dans un système multi agents

- **Définition**

Une organisation peut être définie comme un agencement de relations entre composants ou individus qui produisent une unité, ou système, dotée de qualités inconnues au niveau des composants ou individus. L'organisation lie de façon interrelationnelle des éléments ou événements ou individus divers qui dès lors deviennent les composants d'un tout. Elle assure solidarité et solidité relative, donc assure au système une certaine possibilité de durée en dépit de perturbations aléatoires [Ferber95].

- **Une autre définition**,

L'organisation désigne un ensemble d'agents travaillant ensemble au cours de la résolution d'une ou de plusieurs tâches. Le concept d'organisation peut être exprimé à partir des concepts plus élémentaires d'agent et de tâche. Par rapport au concept de tâche, l'organisation désigne les processus qui permettent : la décomposition des tâches en sous-tâches, l'allocation des tâches aux agents et l'accomplissement des tâches dépendantes de façon cohérente. Par rapport au concept d'agent, l'organisation détermine les statuts et les comportements sociaux d'agents(les rôles) et les relations qui permettent d'unir les agents au sein d'un groupe, que ce soit vis à vis de la décision (les liens d'autorité) ou vis à vis de la coordination (les liens d'engagement) [Bouron, 1992]

Une autre définition plus concrète qualifie l'organisation de « Quel agent fait Quoi et Comment ? » [Hannoun 2000].
Autres définitions existent aussi mais une chose à retenir c'est que l'organisation est directement liée la répartition du travail, des tâches et des rôles sur un ensemble d'agents tout en assurant leur coordination

- **Organisation des agents**

La dimension organisationnelle d'un système multi agents est un concept centrale dans la conception d'un système multi agents car permet de cerner le coté dynamique du système, c'est-à-dire le rôle de ses membres et les relations qui existent entre eux. La situation des agents d'un système multi agents vis-à-vis de l'organisation diffère selon le cas :
Dans le premier cas , l'organisation existe et les interrelations entre agents qu'elle engendre sont prises en compte lors de la conception des agents. Cette organisation est dite statique.

Dans le deuxième cas, les agents n'ont aucune représentation de l'organisation car non directement représentée mais par contre elle émerge des interactions des agents. Cette organisation est dite dynamique.

- **Topologie des organisations**

Selon Gasser [Gasser 92], il existe différentes architectures d'organisations :

- ***Topologies à structure hiérarchique*** **:** ces organisations sont souvent rigides. Le contrôle est centralisé sur un agent qui communique les ordres aux autres agents qui se contentent de les exécuter (Voir figure 45 ci-dessous).

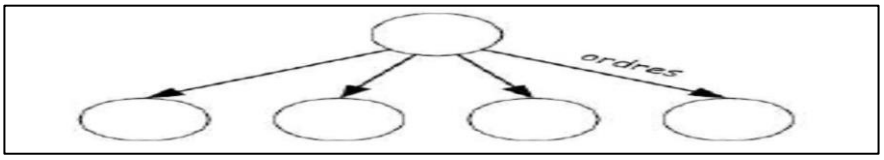


Figure 45 : Topologie à structure hiérarchique

- ***Les topologies de type marché :*** les organisations respectant ce type de topologie (voir figure 46 ci-dessous) sont composées d'agents coordinateurs et d'agents exécutants. Cette structure est un peu plus décentralisée autour de plusieurs objectifs opérationnels.

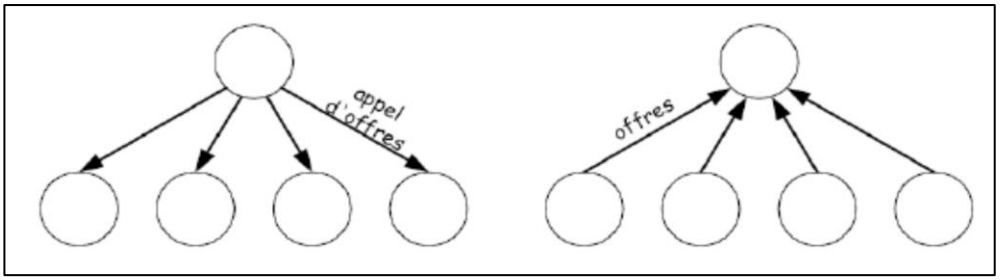


Figure 46 : Topologie de type marché

- ***Les topologies de type communauté :*** le contrôle est fortement distribué et dont les membres possèdent les mêmes capacités (Voir figure 47 ci-dessous).

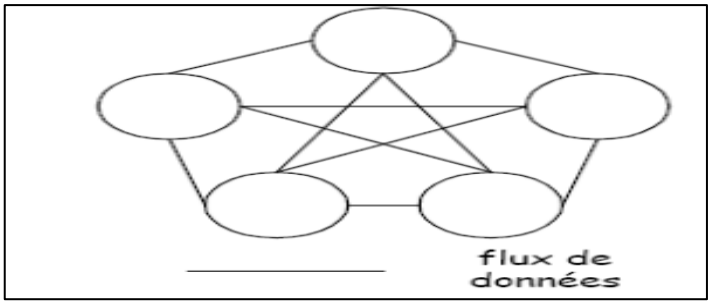


Figure 47 : Topologie de type communauté

- ***Les topologies de type société*** **:** dans ces organisations, le contrôle est décentralisé. Les agents poursuivent chacun un objectif opérationnel et le comportement global est ajusté par des principes de négociation (Voir figure 48 ci-dessous).

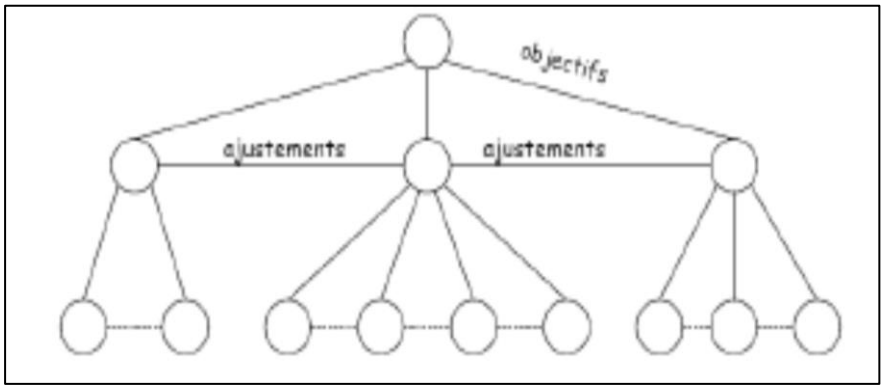


Figure 48 : Topologie de type société

12.7 Réorganisation et auto- organisation des systèmes multi agents

La réorganisation est un vocable relatif aux systèmes multi agents orientés organisation c'est-à-dire les SMA dont le volet organisation est explicité par la définition des rôles entre agents et les groupes constituant les agents. Par opposition, l'auto organisation caractérise les systèmes multi agents qui n'ont aucune représentation explicite de l'organisation mais dont l'interaction entre ses agents permet de faire émerger une organisation, c'est pourquoi de tels systèmes sont connus par systèmes multi agents avec organisations émergentes.

La réorganisation ou l'auto-organisation est un processus permettant de modifier explicitement ou implicitement l'organisation d'un système multi agents le menant à s'adapter le plus à son environnement. [Picard et al. 2009].

Cette réadaptions du système multi agents à son environnement passe par les étapes suivantes : [Picard et al. 2009]. figure 49 suivante :

- Détection de la partie du système défaillante ou freinant sa performance ou l'interdisant à mieux s'adapter à son environnement, ou le situant dans des états indésirables.
- Définition de toutes les alternatives pour résoudre cette inadaptation.
- Choisir parmi la liste des alternatives, celle qui est la plus optimale.
- Appliquer l'alternative choisie à l'organisation courante.

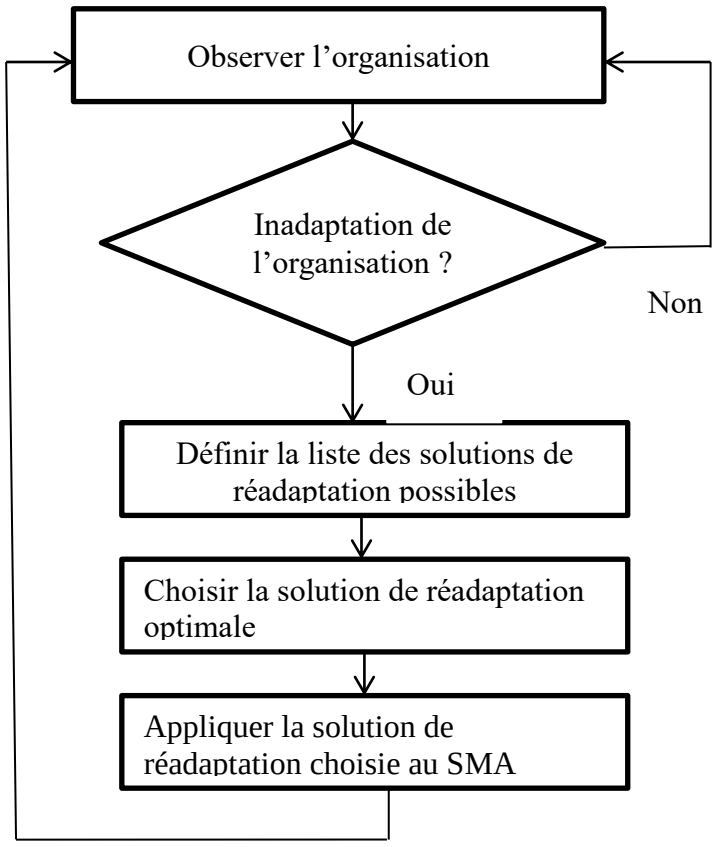


Figure 49 : Cycle du moniteur de réadaptation d'une organisation

12.8 Environnement des agents

Un agent est situé dans un environnement. Pour modéliser la structure de l'agent il faut avoir un modèle de l'environnement. L'environnement peut être vu comme étant dans un état parmi un ensemble d'états $E=\{e_1, e_2 .., e_k\}$. L'environnement peut changer son état soit d'une manière spontanée soit comme résultat des actions de l'agent. L'évolution de l'environnement se modélise différemment selon les caractéristiques que l'on prend en compte. Les principales distinctions à faire sur les types d'environnements sont [Ferber, 1995] [Russel et Norvig, 1995] :

- **Accessible vs inaccessible** : un environnement accessible est un environnement dans lequel l'agent peut obtenir une information complète sur l'état de cet environnement. Le plus accessible des environnements est celui construit pour que les agents puissent y agir.
- **Déterministe vs non-déterministe** : un environnement déterministe est un environnement dans lequel chaque action d'un agent a un effet. Il n'y a pas d'incertitude sur l'état qui résultera de l'action en cours. Dans un environnement non déterministe chaque action provenant d'un agent pourrait avoir plusieurs possibles effets avec des probabilités différentes.
- **Episodique vs non-épisodique** : Un environnement est épisodique si les prochains états de l'environnement ne dépendent pas des actions déjà réalisées.
- **Statique vs dynamique** : un environnement est statique est un environnement dans lequel aucun autre processus, excepté les actions des agents, n'agit sur lui.
- **Discret vs continu** : un environnement est discret s'il y a un nombre fini d'actions possibles et un nombre fini de perceptions.

12.9 Méthodologies de conception des systèmes multi agents

Dans ce qui suit, nous décrivons succinctement les plus importantes des méthodologies d'analyse et de conception des SMA.

- CASSIOPEE

Dans le travail de Collinot et son équipe [Collinot et al., 1996], les auteurs présentent la méthode Cassiopée comme « une façon d'appréhender un type de résolution de problèmes qui suppose la mise en œuvre de comportements collectifs par un ensemble d'agents logiciels ». Afin d'aider le concepteur, cette méthode propose une analyse de l'approche d'un problème par le paradigme mutli-agents. Les auteurs modélisent ce cadre méthodologique suivant trois étapes :

- **Définition des Agents** : le concepteur répartit les connaissances et les compétences nécessaires aux systèmes entre les agents.

- **Définition des Interactions** : le concepteur définit le mode de communication entre les agents et la nature de leurs échanges.

- **Définition de l'Organisation** : le concepteur définit les liens entre les agents qu'il a conçus, comment travailleront-ils en commun afin d'atteindre l'objectif global que le système dans son ensemble doit atteindre.

- GAIA

Gaia est une méthodologie permettant la modélisation de l'agent et de son système [Wooldridge et al. 2000]. Elle suppose que les actions des agents sont toutes concurrentes vis à- vis à d'un but global à atteindre. Les relations entre les agents d'un système construit à l'aide de Gaia sont statiques (elles ne varient pas au cours de la vie du système). Comme la plupart des méthodologies de modélisation, Gaia découpe la construction du système en trois phases :
Analyse des besoins, conception architecturale du système et conception détaillée du même système mais elle ne prend en compte que les deux dernières phases. Cette méthodologie peut être :
(i) Abstraite (qui n'a aucune représentation directe dans le système final). Les entités abstraites sont le modèle de rôles et le modèle d'interactions.
(ii) Concrète (qui se retrouve dans le système construit). Les entités concrètes quant à elles sont le modèle d'agents, le modèle de services et le modèle d'accointances.

- PROMETHEUS

Prometheus [Padgham & Winikoff, 2002] est une méthodologie complète de spécification, conception et implémentation de systèmes d'agents intelligents. Elle est adaptée à la conception des systèmes fermés contenant des agents contrôlés et fiables, par contre, ne l'est pas pour la conception de systèmes ouverts.
A la spécification des besoins, les actions, perceptions et fonctionnalités du système sont définies. (i) les actions et les perceptions définissent l'interface entre les agents et leur environnement. (ii) les fonctionnalités décrivent en un sens plus large ce que devrait faire le système. (iii) Des scénarii de cas d'utilisation sont créés pour fournir une vue plus globale de l'interconnexion entre actions, perceptions, et fonctionnalités. La phase de conception se décompose en deux sous phases :

- La première appelée conception architecturale consiste à définir les agents du système et leurs fonctionnalités. Au cours de cette sous-phase, on définit également les événements auxquels les agents réagissent, les messages qu'ils peuvent recevoir ou émettre. Les protocoles d'interaction sont donc spécifiés sur la base des diagrammes d'interaction. A la fin de cette étape, les données partagées sont identifiées.
- La deuxième sous-phase est la conception détaillée. Elle se préoccupe de la structure interne des agents et de la façon dont ils exécutent leurs tâches.

- VOYELLES

La méthodologie voyelles (AEIO : Agent, Environnement, Interactions, Organisation) [Demazeau, 1995] [Demazeau, 2001a][[Demazeau, 2001b] Se décompose en quatre parties :

13 La facette A permet de représenter l'ensemble des fonctionnalités du raisonnement interne (planification par exemple) de l'agent.
14 La facette E permet de définir l'ensemble des capacités de perception et d'action d'un agent sur son environnement.
15 La facette I permet de définir l'ensemble des interactions avec les autres agents (protocoles de communication par exemple).
16 La facette O est liée aux capacités de structuration et de gestion des relations des agents entre eux.

Toutes ces méthodologies trouvent leur origine dans l'approche orientée objet mais autres méthodologies, telles que CoMoMAS [Norbert, 1996] et MAS-CommonKADS [Carlos et al., 1998], non décrites ici, sont issues de l'ingénierie des connaissances.
Autre remarque c'est que le volet organisationnel du SMA est pris en charge par les méthodologies GAIA, CASSIOPEE et VOYELLES.

12.10 Plates-formes de développement des systèmes multi agents

Les plates-formes multi-agents permettent aux développeurs de concevoir et réaliser leurs applications sans perdre de temps à réaliser des fonctions de base pour la création des agents, de décrire leurs interactions et leur environnement ainsi que de spécifier l'objectif ou les objectifs à atteindre par le système multi agents. Ces plateformes ont l'avantage aussi de surpasser la contrainte d'exiger au développeur d'être familier avec les différents concepts théoriques des systèmes multi-agents. [Ri 01].

Dans ce qui suit, nous allons décrire d'une manière succincte quelques importantes plateformes de développement multi agents utilisées par la communauté SMA.

MACE [Gasser et al., 1987] est le premier environnement de conception et d'expérimentation de différentes architectures d'agents dans divers domaines d'application. Dans MACE, un agent est un objet actif qui communique par envoi de messages. Les agents existent dans un environnement qui regroupe tous les autres agents et toutes les autres entités du système. Un agent peut effectuer trois types d'actions : changer son état interne, envoyer des messages aux autres agents et envoyer des requêtes au noyau MACE pour contrôler les événements internes. Chaque agent est doté d'un moteur qui représente la partie active de l'agent. Ce moteur détermine l'activité de l'agent et la façon dont les messages sont interprétés. MACE a été utilisé pour développer des simulations d'applications distribuées.

SWARM [Minar et al., 1996] est une plate-forme multi-agent avec agents réactifs. L'inspiration du modèle d'agent utilisé vient de la vie artificielle. SWARM est l'outil privilégié de la communauté américaine et des chercheurs en vie artificielle. L'environnement offre un ensemble de bibliothèques qui permettent l'implémentation des systèmes multi-agent avec un grand nombre d'agents simples qui interagissent dans le même environnement.

ZEUS [Nwana et al., 1999] est une plate-forme multi-agent conçue et réalisée par British Telecom (Agent Research Programme of BT Intelligent Research Laboratory) pour développer des applications collaboratives. ZEUS est écrit dans le langage Java et il est fondé sur les travaux de la FIPA. L'architecture des agents ZEUS est similaire à la majorité des agents collaboratifs. Elle regroupe principalement les composantes suivantes :

- une boîte aux lettres et un gestionnaire de messages qui analyse les messages de la boîte aux lettres et les transmet aux composantes appropriées ;
- un moteur de coordination ;
- un planificateur qui planifie les tâches de l'agent en fonction des décisions du moteur de coordination, des ressources disponibles et des spécifications des tâches ;

- plusieurs bases de données représentant les plans connus par l'agent, les ressources et l'ontologie utilisée ;
- un contrôleur d'exécution qui gère l'horloge locale de l'agent et les tâches actives.

L'environnement comporte trois bibliothèques : une avec des agents utilitaires, une avec des outils pour la construction des agents, et une avec des composants agents.

ZEUS met un fort accent sur la méthodologie de développement, fondée sur la notion de rôle. ZEUS a été utilisé pour développer plusieurs applications réelles comme les ventes aux enchères et la simulation de la fabrication des ordinateurs. Les caractéristiques des domaines d'applications de ZEUS ont été définies par les concepteurs ; parmi ces caractéristiques, on peut mentionner :

- chaque agent crée un plan qui nécessite un raisonnement explicite pour atteindre son but ;
- la résolution de problèmes nécessite une coopération entre agents ;
- le rôle de chaque agent consiste à contrôler un système externe qui réalise une tâche du domaine d'application, la résolution de problème est ainsi effectuée par ce système externe et contrôlée par les agents.

JADE (Java Agent Development Framework) [Bellifemine et al., 1999] est une plate-forme multi-agent développée en Java par CSELT (Groupe de recherche de Gruppo Telecom, Italie) qui a comme but la construction des systèmes multi-agent et la réalisation d'applications conformes à la norme FIPA [FIPA, 1997]. JADE comprend deux composantes de base : une plate-forme agents compatible FIPA et un paquet logiciel pour le développement des agents Java.

MADKIT [Gutknecht & Ferber, 2001] est une plate-forme développée par le Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM) de l'Université Montpellier II. MADKIT est libre pour l'utilisation dans le domaine de l'éducation. MADKIT est écrit en Java et est fondé sur le modèle organisationnel ALAADIN Il utilise un moteur d'exécution où chaque agent est construit en partant d'un micro-noyau. Chaque agent a un rôle et peut appartenir à un groupe. Il y a un environnement de développement graphique qui permet facilement la construction des applications.

JADEX [Pokhar et al, 2005]
Est une extension de la plateforme JADE qui permet d'implémenter des systèmes multi agents sur la base d'agents est de type BDI [Bratman, 1987] [Bratman, 1992] (Belief, Desire, Intention). Cette plateforme nous propose un environnement joignant l'architecture middleware de JADE basée sur le « matchmaking » avec la prise en compte de l'architecture interne d'agents intelligents BDI ayant la capacité de raisonner sur ses actions. Le projet JADEX est hébergé sur le site GitHub est sa dernière version (v.4) date de 2018 [Ri,03]

12.11 Dans quels cas doit-on utiliser les systèmes multi agents ?

Selon [Ri 02], les systèmes multi agents sont utilisables dans les cas suivants :

- Le problème est très complexe et ne peut être résolu par un seul agent. Ici le système est composé d'un nombre très important d'entités hétérogènes ou homogènes en interaction. Les systèmes naturels et vie artificielle, en sont l'exemple (colonie de fourmis, essaim d'oiseaux etc.)
- Le système est naturellement distribué (distribution des données et des procédures)
- Le système est soumis à des contraintes distribuées par exemple le problème de satisfaction de contraintes distribuées.
- Le système est évolutif et donc doit s'adapter dynamiquement aux divers changements de l'environnement,
- Le système est ouvert où ses composants peuvent à tout moment quitter ou intégrer le système sans pour autant influer négativement sur son comportement global.
- Une combinaison de cas précédemment cités.

12.12 La planification automatique par l'approche multi agents

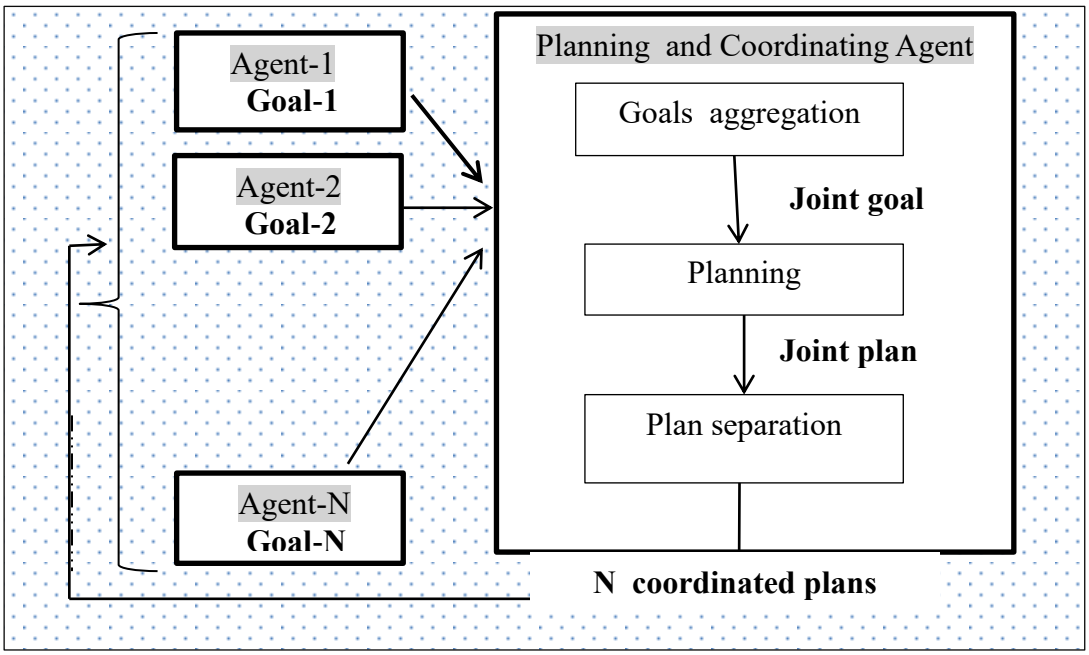
La planification est une faculté cognitive complexe pratiquée par les êtres humains dont le but est de réaliser leurs objectifs. En Intelligence Artificielle, on parle de la planification automatique, qui signifie trouver automatiquement (calculer) un ensemble fini d'actions ordonnées permettant à partir d'un état initial, d'arriver à un état final, appelé état but.
La planification classique dite aussi « mono agent » est la recherche de la séquence d'actions permettant d'atteindre l'état but, dans un environnement totalement observable, déterministe, statique et discret, ce qui est n'est pas souvent le cas. Plusieurs algorithmes de planification automatique classique existent et sont tous de complexité non polynomiale. [Rusell & Norvig 2010]

La planification multi agent est motivée lorsque l'environnement des agents est indéterministe, partiellement observable par les agents et/ou dynamique et/ou continu. En plus, de ces conditions relatives à l'environnement, l'utilisation de l'approche multi agents est justifiée lorsque les agents ont un but global commun mais leurs compétences et les ressources qui leur sont disponibles sont insatisfaisantes pour chercher une solution (un plan) au problème de planification posé.
La planification multi agents c'est surtout comment coordonner les actions des agents pour atteindre un but commun [Olivier 2011]. Il existe trois différentes approches de planification multi agents [Rusell & Norvig 2010] . Nous en donnons ci-dessous une description succincte.

12.12.1 Planification centralisée pour agents multiples

Le système multi agents est constitué d'un seul agent planificateur et de N agents simples exécuteurs de plans. Ces derniers expriment leurs buts et les soumettent à l'agent planificateur qui à son tour élabore un plan global coordonné puis affecte à chaque agent un sous plan pour exécution.

Cette approche est schématisée par la figure 50 ci-dessous.



50: Planification centralisée pour agents multiples

Cette approche présente comme avantage la simplicité mais en revanche tout le système est doté d'un seul agent délégué pour effectuer le planificateur au profit des autres agents qui ne sont que de simples exécuteurs de plans. Si les buts des agents ne sont pas contradictoires alors l'agrégation des buts locaux des agents se résume par une opération d'union sinon l'agent planificateur contacte les agents concernés et les invite à négocier afin de résoudre les conflits de buts entre eux

12.12.2 Planification centralisée pour plans distribués

Dans cette approche de planification, nous avons N agents planificateurs et un agent d'intégration et de coordination de plans. Chaque agent planificateur construit un plan en se basant sur ses buts puis le soumet à l'agent d'intégration. A la réception des plans des agents, l'agent d'intégration, comme son nom l'indique, procède à l'intégration de ces plans puis les sépare pour produire N sous plans coordonnés. (Voir Figure 51 ci-dessous).

Si les agents arrivent à résoudre leurs conflits, l'agent délégué passe à l'étape de planification pour produire un plan global puis procède a le séparer en N sous plans coordonnés et les alloue aux agents pour exécution. Cette approche est recommandée lorsque le nombre d'agents d'exécution de plans n'est pas important et que les buts de ces agents ne sont pas trop conflictuels.

Ici les agents ne communiquent pas durant la planification (plans partiels) ceci pourrait donner naissance à des conflits majeurs et freiner donc l'opération d'intégration. En outre, le processus de planification en entier se base sur le succès de l'opération d'intégration des sous plans qui risque de ne pas aboutir à cause de conflits non résolvables. Néanmoins, cette approche est populaire car beaucoup d'algorithmes d'intégration de plans existent dans la littérature.

.

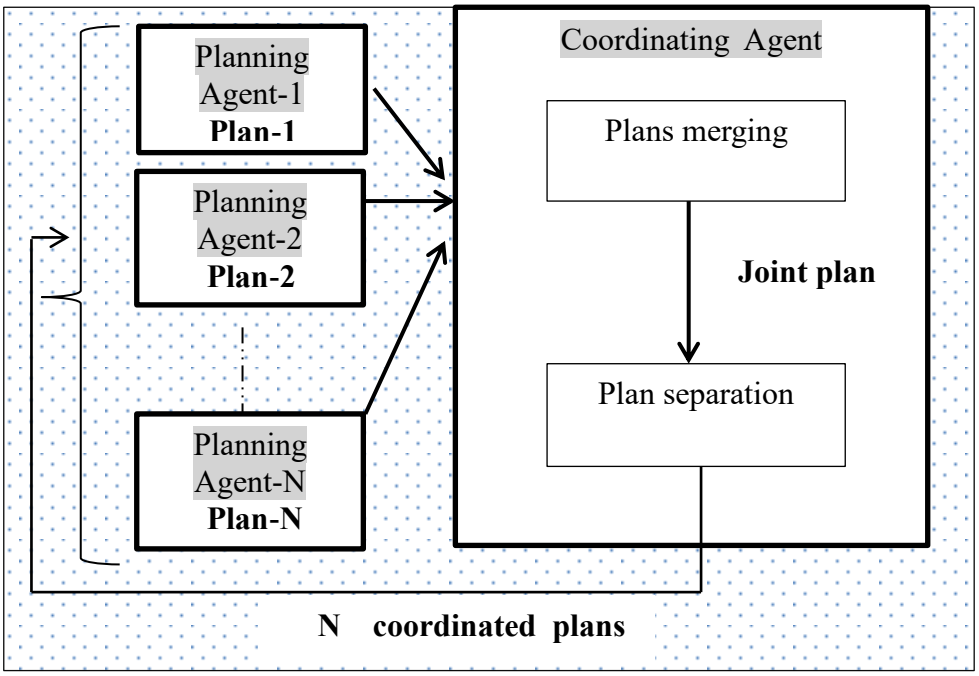


Figure 51: Planification centralisée pour plans distribués

12.12.3 Planification distribuée pour plans distribués

Dans cette approche, chaque agent élabore localement un plan en fonction de ses buts puis le partage avec les autres agents dans le but de le coordonner avec les autres plans (Voir Figure 52 ci-dessous).

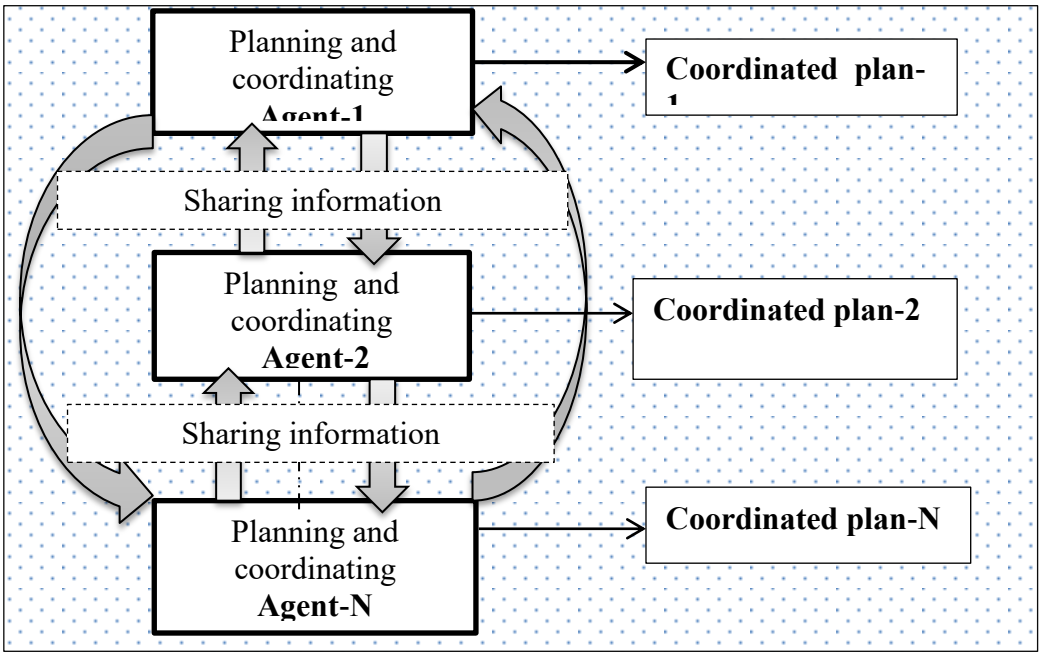


Figure 52 : Planification distribuée pour plans distribués

La coordination de plans dans cette approche se fait donc durant la planification. Pour ce faire, trois méthodes de coordination sont à envisager.

a) Un agent maître appelé « coordinateur » reçoit les informations concernant les plans partiels de tous les agents, analyse ces informations puis envoie des directives de mise à jour aux agents subordonnés ou « esclaves ».

b) Chaque agent peut jouer le rôle de maitre et esclave en même temps selon une architecture hiérarchique.

c) Tous les agents sont maîtres de leurs destins. Chaque agent planifie et coordonne son plan avec les autres agents en toute liberté selon une architecture multi points .

Exemple 45

Un agent A se déplace dans une grille rectangulaire constituée de 06 cellules identiques et communicantes comme montré sur la figure 53. Chaque cellule contient une fenêtre et toutes les fenêtres sont initialement ouvertes. L'agent se trouve initialement dans le centre de la grille et a pour but de fermer toutes les fenêtres. Les actions possibles de l'agent sont :

- move (cellule_source, cellule_destination) : Permet de se déplacer d'une cellule source vers une cellule destination communicantes. Si la cellule source et la cellule destination ne sont pas communicantes l'action n'a aucun effet. La cellule source ou la cellule destination peuvent prendre comme valeur 0.
- closew(x) : consiste à fermer la fenêtre de la cellule x si elle est ouverte.

- Les prédicats nécessaires sont :
- at-position(x) : désigne la cellule ou se trouve l'agent actuellement. Prend la valeur vraie si l'agent est dans la cellule x dans (1,2,...6) , fausse sinon.
- grid(x) : vraie si x est une cellule, fausse sinon.
- opend(x) ; vraie si la fenêtre de la cellule x est ouverte, fausse sinon.
- closed(x) : vraie si la fenêtre de la cellule x est fermée, fausse sinon.
- hall (x) : vraie si x est un hall , fausse sinon.

a) Spécifier ce problème de planification dans le langage PDDL (Planning Domain Description Language) ?
b) Donner deux plans possibles générés par le module de planification de cet agent permettant d'atteindre son but ?

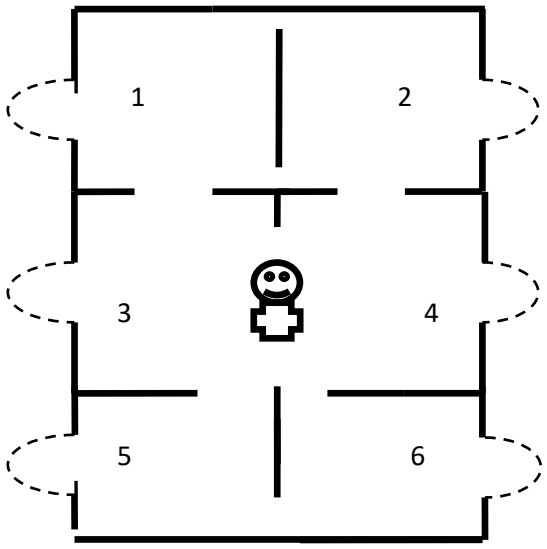


Figure 53 : Grille a six (6) cellules et un agent

c) On suppose maintenant que la tâche la fermeture des fenêtres est effectuée par deux agents agentA et agentB (Figure 54) qui se trouvent initialement au milieu de la grille. Les buts de ces agents sont :

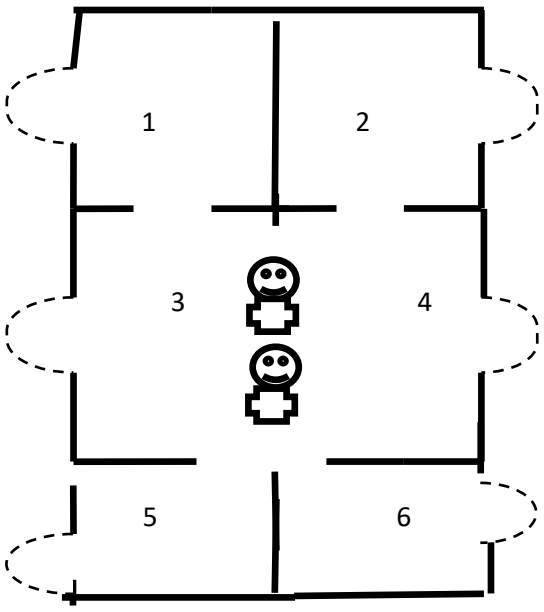


Figure 54 : Grille à six (6) cellules et deux agents

BUT_AGENTA = { closed(1) , closed(2),closed(3),closed(4)}; BUT_AGENTB = { closed(3),closed(4), closed(5),closed(6)} ; Les Fentress sont initialement supposées toutes ouvertes

c1) Décrire les situations conflictuelles possibles entre les deux agents
c2) Donner deux plans partiels locaux conflictuels de l'agent A et l'agent B ?

c3) Que proposeriez- vous au sujet de la coordination entre les deux agents afin qu'ils arrivent à résoudre ou à anticiper ces conflits entre eux ?
c4) Décrire un scénario de coordination distribuée pour plans partiels

d) Supposons maintenant que les buts des deux agents sont:
BUT_AGENTA = {closed(1) , closed(2),closed(3),closed(4)};
BUT_AGENTB = {closed(5),closed(6), opend(3),opend(4)}.
L'agent B a la possibilité de fermer et d'ouvrir les fenêtres. Ainsi les actions possibles de l'agent A restent les mêmes que dans la partie (c) mais l'agent B possède en plus une action d'ouverture d'une fenêtre de la cellule x , exprimée par : OPENW(x). Sachant que l'état initial est le même que précédemment, Dire quelle est la dans ce cas, la situation d'interaction entre les deux agents. ? Les deux agents arrivent-ils à satisfaire leurs buts ?

Solution

a) La spécification d'un problème de planification automatique par le langage PDDL se fait en définissant deux fichiers séparés nommés le fichier "domain" et le fichier "problem". Dans le premier fichier on spécifie les prédicats et les actions tandis que dans le second fichier on spécifie les objets, l'état initial et l'état but. La structure des deux fichiers est comme suit :

(define (domain <domain name>)
<PDDL code for predicates>
<PDDL code for first action>
[…]
<PDDL code for last action>
)
(define (problem <problem name>)
(:domain <domain name>)
<PDDL code for objects>
<PDDL code for initial state>
<PDDL code for goal specification>
)
Ainsi le contenu de ces deux fichers sera comme suit :

```
define (domain   windowbot
( : predicates  (at-position ?x)  (opend ?x) (closed ?x)  hall  ?x) ( grid ?x) )
(:action move  :parameters (?x,?y)
                :precondition  (and (at-position  ?x)  or(( hall  ?x)  (grid ?x))    (grid ?y ))
                :effect  ( and(  at-position(y)     or(( grid ?y)  ( hall  y?)))
:action  closew(x)   :parameters(?x)
                : precondition (and ( at-position ?x)  (grid ?x)   (opend ?x ))
                : effect  ( and (at-postion ?x)  (closed ?x))
        )

define (problem  windowsbot-prob)
```

(: domain windowsbot)

(: objects (g1, g2 ,g3,g4,g5,g6,hl)
(: init (grid g1) (grid g2) (grid g3) (grid g4) (grid g5) (grid g6) (hall hl)
(at-position hl) (opend g1) (opened g2) (opend g3) (opend g4) (opend g5) (opend g6))

(:goal and ((closed g1) (closed g2) closed g3) (closed g4) (closed g5) (closed g6) (at-position hl)))
)

b) Deux plans possibles générés par le module de planification de l'agent :

Plan1 = {move hl,g1 , closew g1 ,move g1 g2 , closew g2, move g2,g4 ,closew g4,move g4,g3,closew g3, move g3 g5 ,closew g5,move g5 g6, closew g6, move g6 hl}.

Plan2 = { move hl,g2 , closew g2 ,move g2 g1, closew g1, move g1,g4 ,closew g4,move g4,g3,closew g3, move g3 g5 ,closew g5,move g5 g6, closew g6, move g6 hl}.

c.1) les deux agents ont des buts compatibles, des compétences suffisantes mais peuvent tomber dans une situation d'encombrement dans le cas de l'accès simultané à la cellule 3 et à la cellule 4. Le cas échéant, les deux agents doivent coopérer pour résoudre ces deux situations d'encombrement.

c.2) Le tableau 15. Ci-dessous donne deux plans partiels de l'agent A et l'agent en situation d'encombrement.

No. action	Plan partiel agentA	Plan partiel agent B
01	move hl,g1	move hl,g3
02	closew g1	closew g3
03	move g1 g2	move g3 g5
04	closew g2	closew g5
05	move g2,g4	move g5,g4
06	closew g4	closew g4
07	move g4,g3	move g4,g6
08	closew g3	closew g6
09	move g3 hl	move g6 hl

Tableau 15. Deux plans partiels avec une situation d'encombrement

Ici, la même action "closew g4" est la 6^{eme} action dans les deux plans partiels de l'agent A et de l'agent B, ce qui met ces deux agents en situation de conflit (on suppose bien sûr que le temps d'exécution des actions est le même partout)

c.3) la résolution de cette situation d'emcombrement peut être évité soit par anticipation ou soit par détection et résolution :

- Méthode par anticipation

ici les deux agents doivent coordonner avant l'établissement de leurs plans partiels en partageant leurs buts pour pouvoir déterminer les sous buts en double (dans notre cas fermer la fenêtre 3, fermer la fenêtre 4). Une fois déterminées, ces sous buts doubles sont soient affectés le plus équitablement possible entre les deux agents, soient pris en charge par un des agents uniquement en appliquant un tirage au sort par exemple.

- Méthode par détection et résolution

Ici les deux agents coordonnent après avoir établi chacun un plan partiel. Ces deux plans sont examinés pour voir s'ils sont sources de conflit (du genre du tableau 16). Une fois le conflit est détecté, un des agents ou les deux est (sont) appellé(s) à réviser son (leurs) plan(s) (replanification) pour enlever le conflit. (Le tableau 16 ci-dessous donne deux plans sans conflit avec une révision du sous plan de l'agent B)

No. action	**Plan partiel agentA**	**Plan partiel agent B**
01	move hl,g1	move hl,g3
02	closew g1	closew g3
03	move g1 g2	move g3 g5
04	closew g2	closew g5
05	move g2,g4	move g5,g6
06	closew g4	closew g6
07	move g4,g3	move g6,g4
08	closew g3	closew g4
09	move g3 hl	move g4 hl

Tableau 16. Deux plans partiels sans encombrement

d) Les deux agents ont des buts incompatibles, des compétences suffisantes mais peuvent tomber dans des situations de conflits individuels pour les ressources, ils sont donc deux agents antagonistes. Les buts des deux agents ne peuvent donc être jamais atteints en même temps. Les deux agents demeurent toujours en activité sans pouvoir pour autant atteindre simultanément leurs buts.

Conclusion générale

Ce support de cours est composé d'une douzaine de chapitres répartis sur les thèmes abordés par l'I.A symbolique, l' I.A connexionniste et l'I.A inspirée de la nature et de la biologie, sans oublier les techniques de résolution des problèmes de satisfaction de contraintes. Le dernier chapitre de ce document, nous l'avons consacré à l'étude des systèmes multi agents, une des approches de l'intelligence artificielle distribuée. Ce contenu donc peut être utilisé par les étudiants de la filière informatique des deux paliers Licence et Master. Nous l'avons organisé d'une manière à ce qu'il reflète l'évolution de l'intelligence artificielle depuis l'ère des systèmes experts jusqu'à l'avènement des systèmes multi agents. Aussi, par souci de pédagogie et de clarté, chaque chapitre étudié est illustré par des exemples.

Les systèmes experts sont des systèmes à base de connaissances. Ce sont des outils très importants, qui offrent une bonne approche pour coder, enregistrer et implémenter l'expertise humaine, où dans plusieurs domaines cette expertise est souvent très rare, en bénéficiant du développement progressif de la technologie. Ceci dans le but de remplacer les limites des experts humains, comme la fatigue, l'oubli et le décès qui constituent une grande perte d'expertise. Le développement d'un système expert est une procédure longue et dure. La réussite de la mise en place d'un système expert est basée sur la bonne construction de leur base de connaissances. Il faut signaler qu'un mauvais choix de représentation des connaissances, même face à un bon moteur d'inférence et une interface utilisateur attirante, influe négativement sur la fiabilité de la solution proposée par un système expert.

Comme nous l'avons déjà signalé dans l'introduction générale, à cause de ces difficultés rencontrées lors du développement des systèmes experts, les chercheurs et les développeurs de tels systèmes se sont trouvés contraints de combiner la technologie des systèmes experts avec d'autres technologies plus avancées de l'IA comme la logique floue et la théorie de l'incertitude pour donner naissance aux systèmes experts avec raisonnement incertain et raisonnement flou. Nous avons consacré une bonne partie de ce document aux trois types de systèmes experts : systèmes experts classiques, systèmes experts incertains et systèmes experts flous.

Dès l'année 1986, les chercheurs en IA ont commencé a constaté que les systèmes experts ne pouvaient pas résoudre des tâches plus complexes telles la vision artificielle et la lecture et la compréhension automatique des textes.. En outre, cette fameuse année coïncide avec la mise au point par Rumelhart de l'algorithme d'apprentissage dit par rétro propagation de l'erreur, pour les réseaux de neurones multi couches destinés à résoudre les problèmes non linéaires. Ces deux facteurs en plus de l'avancement technologique des ordinateurs ont accéléré le basculement de la recherches scientifique de l'intelligence artificielle symbolique vers la l'intelligence artificielle connexionniste puis après vers les systèmes inspirés de la nature et de la biologie.

Comme exemple de modèles de l'IA connexionniste, nous avons étudié deux modèles de réseaux de neurones : le modèle de perceptron multi- couches et le modèle récurent de Hopfield . Les algorithmes d'apprentissage dans les deux modèles ont été aussi abordés et illustrés par des exemples.

Dans le cadre de l'IA inspirée de la nature et de la biologie, nous avons étudié les algorithmes génétiques et les algorithmes de colonies de fourmis, deux modèles qui offrent chacun une solution optimale aux problèmes complexes et à explosion combinatoire.

Dès 1995, la volonté accrue de résolution de problèmes de plus en plus complexes et distribués telle effectuée dans la nature par des entités en interaction (comme les colonies de fourmis pour la recherche collective d'une source de nourriture ou le déplacement d'une importante nuée d'oiseaux dans le ciel sans le moindre incident) ont poussé la chercheurs à inventer une nouvelle approche basée agents qui est une extension de l'approche orientée objets.. Dans le contexte de cette nouvelle approche, tout problème complexe et distribué peut être modélisé par un ensemble d'agents en interaction dit système multi agents.

Actuellement, après que le Web et les réseaux sociaux soient devenus le moyen de communication et le support de stockage d'informations de taille colossale, on s'attend à ce que le génie logiciel basé agents ait sa place qu'il mérite, notamment avec la nouvelle révolution en technologie des ordinateurs qui se prépare déjà, nommée « Quantum Computing ».

Références bibliographiques

[Alison 1987] **:** Knowledge Acquisition for Expert Systems A Practical Handbook Edited by ALISON L. KIDD Hewlell Packard Laboratories Bristol, England, 1987 ISBN-13: 978-1-4612-9019-3 e-ISBN-13: 978-1-4613-1823-1, DOl: 10.1007/ 978-1-4613-1823-1

[Bellifemine et al., 1999] : Bellifemine, F., Poggi, A. and Rimassa, G. (1999) JADE—A FIPA-Compliant Agent Framework. C Selt Internal, 31, 103-128.

[Bellman, 1978] : Bellman, R. E. (1978). An Introduction to Artificial Intelligence: Can Computers Think? Boyd & Fraser Publishing Company.

[Bouzenita 2012] : Conception et Implémentation d'un système expert hybride pour le Diagnostic industriel. Mémoire de Magister. Auteur : Mohammed BOUZENITA, Université El Hadj Lakhdar Batna , 2012.

[Buchanan & Shortliffe & 1975] : Buchanan, B. G. and Shortliffe, E. H. (Eds.).(1984). *Rule-Based Expert Systems: The MYCIN. Experiments of the Stanford Heuristic ProgrammingProject*. Addison-Wesley.

[Bond et Gasser 1988] : Bond A. et Gasser L. (1988) Readings in Distributed Artificial Intelligence. Morgan Kaufman.

[Bouron, 1992] : Bouron T. (1992) Structures de communication et d'organisation pour la coopération dans un univers multi-agents. Th`ese de l'universit´e Paris 6.

[Bratman 1987] Bratman. M Intention, Plans, and Practical Reasoning. Harvard University Press, Cambridge, Massachusetts, 1987.

[Bratman 1992] : Bratman, M. E. (1992). Planning and the stability of intention. *Minds and Machines*, *2*(1), 1–16.

[Carlos et al., 98]: Carlos A. Iglesias, Mercedes Garijo, Jos´e C. Gonz´alez, and Juan R. Velasco. Analysis and design of multiagent systems using MAS-CommonKADS. In AAAI'97Workshop on Agent Theories, Architectures and Languages,

[Collinot et al., 1996]: Collinot, A., Drogoul, A. and Benhamou, P. (1996) Agent oriented design of a soccer robot team. In *Proceedings of the 2nd International Conference on Multi-Agent Systems (ICMAS-96)*, pp. 41-47, Kyoto, Japan.

[Chaib-Draa et al., 2001] : Chaib-Draa, B., Jarras, I., & Moulin, B. (2001). Systèmes multi-agents: principes généraux et applications. *Edition Hermès*, *242*, 1030-1044.

[Charniak & McDermott, 1985] : Charniak, E. and McDermott, D. (1985). Introduction to Artificial Intelligence. Addison-Wesley.

[Chomsky 1956]: Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, *2*(3), 113–124.

[Chomsky 1957] : Chomsky, N. (1957). *Syntactic Structures*. Mouton.

[Demazeau, 1995]: Y. DEMAZEAU : From Interactions to Collective Behaviour in Agent-Based Systems. Dans Proceedings of the First European Conference on Cognitive Science (ECCS'95), Saint Malo, France, pages 117–132, 1995.

[Demazeau, 2001a]: Y. DEMAZEAU : MAS Methodology. Présentation lors du Séminaire IRIT - Cycle Systèmes Multi-Agents, Institut de Recherche en Informatique de Toulouse - Université Paul Sabatier, 15 Novembre 2001.

[Demazeau, 2001b]: Y. DEMAZEAU : Voyelles. Mémoire d'habilitation à diriger des recherches, INP Grenoble, 2001

[Drogoul, 1993] : Drogoul A. (1993) De la simulation multi-agent `a la r´esolution collective de probl`emes. Une ´etude de l'´emergence de structures d'organisation dans les syst`emes multi-agents.. Th`ese de l'universit´e Paris 6.

[Erceau & Ferber, 1991]. : Erceau J. et Ferber J. (1991) "L'intelligence artificielle distribuée." In La Recherche, Juin.

[Ferber 1995] : Les Systèmes Multi Agents: vers une intelligence collective. Jacques Ferber.Intereditions ,1995.

[Fisher et al. 1995] : Fischer K., Müller J.P., Pischel M. "Unifying control in a layered agent architecture". , IJCAI95, Agent Theory, Architecture and Language Workshop 95, pp. 240- 252.Frasson. C.(eds),. Proceedings of the 5th international conference on autonomous agents, 324-331, Monreal-Canada, 1995.

[FIPA, 1997]. : Fipa, F. I. P. A. (1997). *specification part 2: Agent communication language*. Technical report, FIPA-Foundation for Intelligent Physical Agents.

[François & Laurent 2006] : Intelligence Artificielle : Les systèmes experts, Manuel de cours. (Janvier 2006), François Denis et Laurent Miclet, Université de Rennes I , ENSSAT , Lannion LSI 2.

[Gasser et al. 1987] : Gasser, L. (1987). MACE: A flexible testbed for distributed Al research. *Distributed artificial intelligence.*

[Gutknecht et Ferber, 2001]: Gutknecht, O., & Ferber, J. (2001). The M ad K it agent platform architecture. In *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems: International Workshop on Infrastructure for Scalable Multi-Agent Systems Barcelona, Spain, June 3–7, 2000 Revised Papers* (pp. 48-55). Springer Berlin Heidelberg.

[Frege 1879]: Frege, G. (1879). Begriffsschrift, eine der arithmetischennachgebildete Formelsprache des reinen Denkens. Halle, Berlin.
("Concept writing, one of the arithmetic modeled formula language of pure thinking". English translation appears in van Heijenoort ,1967).

[Hannoun 2000]. : Hannoun, M., Boissier, O., Sichman, J. S., & Sayettat, C.(2000). MOISE: An organizational model for multi-agentsystems. In Advances in Artificial Intelligence (pp. 156-165).Springer Berlin Heidelberg .

[Holland et al. 1975] Holland, J. H. (1975). *Adaption in Natural and Artificial Systems*. University of Michigan Press.

[Hopfield 1982]: Hopfield, J. J. (1982). Neurons with graded response have collective computational properties like those of two-state neurons. *PNAS, 79*, 2554–2558.

[Jennings et al. 1998] : N.R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. Autonomous Agents and Multi-Agent Systems, 1:7-38, 1998

[Marco et al. 1990] : Ant system: optimization by a colony of cooperating agents
M Dorigo, V Maniezzo, A Colorni IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 26 …

[Marco et al. 2006] : Ant colony optimization M Dorigo, M Birattari, T Stutzle. IEEE computational intelligence magazine 1 (4), 28-39.

[Minsky 1975] : Minsky, M. L. (1975). A framework for representing knowledge. InWinston, P. H. (Ed.), *The Psychology of Computer Vision*, pp. 211–277.McGraw-Hill.
Originally an MIT AI Laboratory memo; the 1975 version is abridged, but is the most widely cited.

[Minsky et Papert 1969] : Minsky, M. L. and Papert, S. (1969). *Perceptrons:An Introduction to Computational Geometry* (first edition). MIT Press.

[Minar et al. 1996] : Minar, N., Burkhart, R., Langton, C., & Askenazi, M. (1996). The swarm simulation system: A toolkit for building multi-agent simulations.

[Norbert 1996] : Contribution à l'acquisition et à la modélisation de connaissances dans un cadre multi-agents : l'approche CoMoMAS . Thése soutenue par Norbert Glaser. 1996. Université Nancy1.

[Nwana et al. 1999]: Nwana, H. S., Ndumu, D. T., Lee, L. C., & Collis, J. C. (1999). ZEUS: a toolkit for building distributed multiagent systems. *Applied Artificial Intelligence*, *13*(1-2), 129-185.

[Olivier 2011]: Olivier Boissier , 'Planification multi agents, support de cours' ENS des Mines Saint Etienne, France , 2011.

[Padgham & Winikoff, 2002] : Padgham, L., & Winikoff, M. (2002, July). Prometheus: A methodology for developing intelligent agents. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1* (pp. 37-38).

[Picard et al. 2009]: Picard, G., Hübner, J. F., Boissier, O., & Gleizes, M. P. (2009, October). Réorganisation et auto-organisation dans les systèmes multi-agents. In *Journées Francophones sur les Systèmes Multi-Agents (JFSMA'09)* (pp. pages-89).

[Pokhar et al. 2005] JADEX: A BDI REASONING ENGINE Alexander Pokahr,1 Lars Braubach,1 and Winfried Lamersdorf1 1University of Hamburg Distributed Systems and Information Systems 22527 Hamburg, Chapter Book.

[Quinlan 1961] Quillian, M. R. (1961). A design for an understanding machine. Paper presented at a colloquium: Semantic Problems in Natural Language, King's College,Cambridge, England.
[Quinlan 1968] : Quillian, M. Ross (1968), "Semantic Memory", in Marvin Minsky (ed.) Semantic Information Processing (Cambridge, MA: MIT Press): 227-270.

[Quinlan 1979] **:** Quinlan, J. R. (1979). Discovering rules from large collections of examples: A case study. In Michie,D. (Ed.), *Expert Systems in the Microelectronic*

[Reiter 1980] : Reiter, R. (1980). A logic for default reasoning. *AIJ,13*(1–2), 81–132.

[Rumelhart et al. 1986] : Rumelhart, D. E., Hinton, G. E., and Williams,R. J. (1986b). Learning representations by backpropagating errors. *Nature, 323,* 533–536.

[Russel et Norvig, 1995] : S.J. Russell and P. Norwig. Artificial Intelligence. A Modern Approach. Prentice Hall, Englewood Cliffs, New Jersey, 1995.

[Rusell & Norvig 2010] S. Russell, P. Norvig , Artificial intelligence : A modern approach, 3nd Edition, 2010 .ISBN-13: 978-0-13-604259-4 ISBN-10: 0-13-604259-7

[Rosenblatt 1957]: Rosenblatt, F. (1957). The perceptron: A perceiving and recognizing automaton. Report 85-460-1,Project PARA, Cornell Aeronautical Laboratory.

[Outellet & Tessier 1987]. Ouellet, M.//Tessier, J. C. « Intelligence artificielle et systèmes experts : principes et méthodes ». INRS-Eau, Rapport de recherche annuel 1986-1987. ISBN : 2-89146-223-8. Québec. Janvier 1987

[Smith 1980] : Smith R. G. (1980) "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver". IEEE Trans. on Computers. 29(12), p. 1104-1113.

[Turing 1950] : Turing, A. (1950). Computing machinery and intelligence. *Mind, 59,* 433–460.

[Weiss, 1999], Multiagent Systems,A Modern Approach to Distributed Artificial Intelligence edited by Gerhard Weiss The MIT Press. Cambridge, Massachusetts
London England, 1999. ISBN 0-262-23203-0

[Winston, 1992] : Winston, P. H. (1992). *Artificial Intelligence* (Third edition). Addison-Wesley.

[Wooldridge and Jennings 1995].Wooldridge, M. and Jennings, N. R. (1995) Intelligent agents: theory and practice. *The Knowledge Engineering Review,* 10(2), 115-1 52.

[Wooldridge et al., 2000]: M. WOOLDRIDGE, N. R. JENNINGS et D. KINNY : The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems, 3(3):285–312, 2000.

[Zadeh 1965] : Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, *8*, 338–353.
[Zadeh 1978] : Zadeh L. A. (1978). Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, *1*, 3–28.

Ressources bibliographiques sur Internet.

[Ri, 01] : http://theses.univ- Lyon2.fr/ documents/ getpart.php id=lyon2. 2008.nfaoui_eh&part = 152187.
(Consulté le 20.01.2022 à 16h05)

[Ri, 02] : https://www.emse.fr/~boissier/enseignement/sma01/pdf/introduction.pdf

(Consulté le 25.04..2022 à 22h20)

[Ri 03] : https://vsis-www.informatik.uni-Hamburg.de/getDoc. php/ publications/ 250/promasbook _jadex.pdf.
(Consulté le 01.01.2023 à 9h14)

[Ri 04] : https://perso.liris.cnrs.fr/nathalie.guin/Prolog/Cours/Cours2-Prolog1.pdf

(Consulté le 20.01.2023 à 15h32)

[Ri 05] : https://cs.union.edu/~striegnk/learn-prolog-now/

(Consulté le 1 01.01 .2023 à 11h35)
[Ri 06] : http://www8.umoncton.ca/umcm- cormier_gabriel /SystemesIntelligents/GIND5439_ Chapitre4.pdf

(Consulté le l0 02 .2023 à 16h35)

[Ri 07] https://www.linkedin.com/advice/0/what-advantages-challenges-using-default-logic-nonmonotonic
(Consulté le 06.03.2023 à 12h30)

.
.