INSS : un système hybride neuro-symbolique pour l'apprentissage automatique constructif

Fernando Santos Osorio

THÈSE

présentée à

L'Institut National Polytechnique de Grenoble - I.N.P.G.
Laboratoire LEIBNIZ - IMAG

pour obtenir le titre de

DOCTEUR

Spécialité:

Informatique
(Arrêté ministériel du 30 mars 1992)

INSS : UN SYSTEME HYBRIDE NEURO-SYMBOLIQUE POUR L'APPRENTISSAGE AUTOMATIQUE CONSTRUCTIF

par

Fernando Santos Osório

Soutenue le 3 février 1998 devant le jury composé de:

MM.	Philippe JORRAND	Directeur
	Alain GRUMBACH	Rapporteur
	Christian PELLEGRINI	Rapporteur, Président
	Bernard AMY	Examinateur (co-directeur)
	Frédéric ALEXANDRE	Examinateur
	Vincent RIALLE	Examinateur

A Denise et Felipe,

mes deux compagnons de toutes les heures

embarqués dans cette aventure avec moi.

Remerciements:

Je tiens tout d'abord à remercier Bernard Amy, qui m'a accepté dans son équipe, a consacré beaucoup de son temps à mon travail, et a fait de nombreuses remarques, toujours intéressantes, sur cette thèse. Bernard a toujours su diriger mes recherches dans la bonne direction; je dois beaucoup à lui pour ses nombreuses contributions à la réussite de cette thèse.

Je tiens à remercier aussi Alain Grumbach, Professeur à l'ENST de Paris, et à Christian Pellegrini, Professeur à l'Université de Genève, pour avoir accepté de juger ce travail.

Je tiens également à remercier Vincent Rialle, Maître de Conférences à l'Université Joseph Fourrier de Grenoble, et à Fréderic Alexandre, Chercheur à l'INRIA Lorraine, qui ont gentiment accepté de participer à mon jury de thèse.

Merci beaucoup à Philippe Jorrand, Directeur du Laboratoire Leibniz - IMAG/INPG, qui a accepté de m'accueillir dans ce laboratoire, et aussi, qui a accepté de passer toutes ces années à mon côté, comme mon directeur de thèse.

Je remercie à la CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nivel Superior) et plus particulièrement au programme de coopération Franco-Brésilien CAPES-COFECUB, de m'avoir accordé un financement pour la réalisation de cette thèse. Le financement ainsi que le support administratif fourni par ces deux organismes a été une pièce fondamentale pour permettre mon séjour en France et la réalisation de cette thèse. Merci à Brigitte Plateau et Claudio Geyer, coordinateurs du projet de coopération CAPES-COFECUB entre l'UFRGS et l'INPG, de m'avoir permis de participer de son projet de coopération. Je tiens aussi à remercier particulièrement à Philippe Navaux de l'UFRGS.

Je remercie aussi à l'UNISINOS (Universidade do Vale do Rio dos Sinos, Brésil), pour m'avoir accordé un détachement de mes fonctions d'enseignant de 1993 à 1998, et principalement, pour son soutien constant qui m'a permis de mener mes travaux de doctorat jusqu'à la fin.

Je tiens aussi à remercier :

Les membres des laboratoires LIFIA et Leibniz qui m'ont beaucoup aidé dans la réalisation de cette thèse. Plus particulièrement, je remercie aux 'anciens de l'équipe' Arnaud Giacometti, Bruno Orsier, Abderrahim Labbi, Nicolas Szilas, Maria Malek, Françoise Fessant et Emmanuel Duriez pour leurs inestimables contributions sur le plan scientifique et aussi pour leur amitié. Je remercie aussi à tous ceux qui ont travaillé a mon côté dans le développement et l'utilisation du système INSS (Loïc Decloedt, Pierre-Yves Blond, Gerardo Reyes et Benjamin Deuker), ainsi qu'à Vincent Danel et Vincent Rialle pour leurs contributions dans l'application médicale. Je remercie particulièrement à Henri Saya, notre ingénieur système, à Jaime Sichman et à Remis Balaniuk, qui m'ont beaucoup

aidé à "m'insérer" dans ce laboratoire d'un point de vue technique et aussi humain. Je remercie ~~beaucop~~ beaucoup ;^) aux *héros* qui ont fait la révision de ce document, les critiques sur son contenu, et la correction de mes innombrables fautes commises dans la langue de Molière: Nico, Eric Gauthier, Daniel Memmi, Sophie Rouzier et Bernard.

Tous les membres de l'équipe RESEAUX dirigée par Bernard Amy au laboratoire Leibniz pour leur accueil très chaleureux et leur disponibilité constante, avec qui j'ai eu des rapports aussi divers qu'enrichissants. *Une thèse est principalement le résultat d'un vrai travail d'équipe!*

Merci aussi à tous mes amis des autres groupes de recherche de Grenoble : l'équipe LAPLACE (en particulier à Olivier Lebeltel pour sa inestimable aide avec le petit Khepera), les équipes MAGMA et PRIMA, les amis des laboratoires TIMA, TIMC et LMC, aux membres de l'association InCognito, et tant d'autres compagnons de recherche et d'amitié.

Merci à Scott Fahlman, à Geoffrey Towell, à Olivier Michel, et à tous les autres chercheurs qui ont mis leurs outils, articles et travaux accessibles à travers l'Internet, et merci à tous les membres du projet Européen MIX. Vos contributions ont beaucoup apporté au développement de cette thèse.

Et bien sûr, merci à toute la communauté brésilienne de Grenoble et à tous mes amis avec qui j'ai garde des souvenirs très agréables des bonnes moments vécus ensemble. Nos échanges de messages (et des "flames") à travers la "penduick-fm" resteront à jamais enregistrés dans la mémoire non-volatile des nos âmes... "SaluTCHE et Obrigado!".

Parmi la communauté brésilienne de Grenoble, je garderai un souvenir très particulier de tous ceux qui ont été toujours prêts à m'encourager dans ce long parcours : l'unique et inestimable João Paulo Kitajima, Silvio Nabeta, Ricardo Duarte et Silvana, Alvaro Guarda et Vera, Gilson Braviano et Teresinha, Paulo Fernandes et Marilia, José Celso et Néia, Gilberto Marchioro (Giba) et Camila, Alexandre Ribeiro et Helena.

Je remercie aussi à tous ceux et celles qui sont là-bas de l'autre côté du monde, au Brésil, et qui ont donné leur soutien à la réalisation de cette thèse. Merci particulièrement à mes beaux-parents Egydio et Eneida, à mes grands-parents Luiz Carlos et Maria, et à ma mère Beatriz, je n'aurais pas pu arriver jusqu'ici sans vous.

Finalement, merci, merci et encore merci à mon épouse qui a toujours su me soutenir (dans tous les sens de ce mot, et principalement dans le sans brésilien - "suportar, agüentar") et me donner les forces nécessaires pour arriver jusqu'à la fin de cette thèse. Je n'aurais jamais écrit chacune des 300 pages qui composent cette thèse sans son inestimable aide et amour. DeniSe, ma vraie DS à moi, tu es for me... formidable!

Merci à Denise et Felipe, sources de mon inspiration.

Merci à toute ma famille et plus particulièrement à mes parents Beatriz et Luiz Carlos.

Résumé:

Plusieurs méthodes ont été développées par l'Intelligence Artificielle pour reproduire certains aspects de l'intelligence humaine. Ces méthodes permettent de simuler les processus de raisonnement en s'appuyant sur les connaissances de base disponibles. Chaque méthode comporte des points forts, mais aussi des limitations. La réalisation de systèmes hybrides est une démarche courante qui permet de combiner les points forts de chaque approche, et d'obtenir ainsi des performances plus élevées ou un champ d'application plus large. Un autre aspect très important du développement des systèmes hybrides intelligents est leur capacité d'acquérir de nouvelles connaissances à partir de plusieurs sources différentes et de les faire évoluer.

Dans cette thèse, nous avons développée des recherches sur les systèmes hybrides neuro-symboliques, et en particulier sur l'acquisition incrémentale de connaissances à partir de connaissances théoriques (règles symboliques) et de connaissances empiriques (exemples). Un nouveau système hybride, nommé système INSS - *Incremental Neuro-Symbolic System*, a été étudié et réalisé. Ce système permet le transfert de connaissances déclaratives (règles symboliques) d'un module symbolique vers un module connexionniste (réseau de neurones artificiel - RNA) à travers un convertisseur de règles en réseau. Les connaissances du réseau ainsi obtenu sont affinées par un processus d'apprentissage à partir d'exemples. Ce raffinement se fait soit par ajout de nouvelles connaissances, soit par correction des incohérences, grâce à l'utilisation d'un réseau constructif de type Cascade-Correlation. Une méthode d'extraction incrémentale de règles a été intégrée au système INSS, ainsi que des algorithmes de validation des connaissances qui ont permis de mieux coupler les modules connexionniste et symbolique. Le système d'apprentissage automatique INSS a été conçu pour l'acquisition constructive (incrémentale) de connaissances. Le système a été testé sur plusieurs applications, en utilisant des problèmes académiques et des problèmes réels (diagnostic médical, modélisation cognitive et contrôle d'un robot autonome). Les résultats montrent que le système INSS a des performances supérieures et de nombreux avantages par rapport aux autres systèmes hybrides du même type.

Abstract:

Various Artificial Intelligence methods have been developed to reproduce intelligent human behaviour. These methods allow to reproduce some human reasoning process using the available knowledge. Each method has its advantages, but also some drawbacks. Hybrid systems combine different approaches in order to take advantage of their respective strengths. These hybrid intelligent systems also present the ability to acquire new knowledge from different sources and so to improve their application performance.

This thesis presents our research in the field of hybrid neuro-symbolic systems, and in particular the study of machine learning tools used for constructive knowledge acquisition. We are interested in the automatic acquisition of theoretical knowledge (rules) and empirical knowledge (examples). We present a new hybrid system we implemented: INSS - Incremental Neuro-Symbolic System. This system allows knowledge transfer from the symbolic module to the connectionist module (Artificial Neural Network - ANN), through symbolic rule compilation into an ANN. We can refine the initial ANN knowledge through neural learning using a set of examples. The incremental ANN learning method used, the Cascade-Correlation algorithm, allows us to change or to add new knowledge to the network. Then, the system can also extract modified (or new) symbolic rules from the ANN and validate them. INSS is a hybrid machine learning system that implements a constructive knowledge acquisition method. We conclude by showing the results we obtained with this system in different application domains: ANN artificial problems (The Monk's Problems), computer aided medical diagnosis (Toxic Comas), a cognitive modelling task (The Balance Scale Problem) and autonomous robot control. The results we obtained show the improved performance of INSS and its advantages over others hybrid neuro-symbolic systems.

TABLE DES MATIERES

TABLE DES FIGURES

1. INTRODUCTION

De nos jours, l'informatique à pris une place importante dans notre société. Les systèmes informatiques deviennent de plus en plus complexes, comme les tâches qui leur sont confiées. Les systèmes d'Intelligence Artificielle (I.A.) ont été créés avec l'espoir de pouvoir aider l'homme dans les tâches les plus complexes, celles qui requièrent l'utilisation de l'intelligence[1] ou la réalisation de procédés dits intelligents. Les résultats obtenus jusqu'à présent sont encore assez limités par rapport à notre conception d'un vrai système intelligent. A quelques rares exceptions près, l'homme reste toujours supérieur à la machine pour la réalisation de tâches complexes.

Plusieurs domaines de recherche en I.A. ont pour but la reproduction de certains aspects de l'intelligence humaine, et plusieurs méthodes ont été développées à cette fin. Ces méthodes nous permettent de simuler les processus de raisonnement humain (par exemple : l'inférence, l'analogie et la déduction) en s'appuyant sur les connaissances de base disponibles. Chaque méthode comporte des points forts, mais aussi des limitations. La réalisation de systèmes hybrides est une démarche courante qui nous permet de combiner les points forts de chaque approche, et d'obtenir ainsi des performances plus élevées ou un champ d'application plus large. Un autre aspect très important du développement des systèmes intelligents est leur capacité d'acquérir de nouvelles connaissances (parfois à partir de plusieurs sources différentes) et de les faire évoluer.

Dans cette thèse, nous avons développé des recherches sur les système hybrides neuro-symboliques (SHNS), et en particulier sur l'acquisition incrémentale de connaissances à partir de connaissances théoriques (règles symboliques) et de connaissances empiriques (exemples). Un nouveau système hybride, nommé système INSS - *Incremental Neuro-Symbolic System*, a été étudié et réalisé. Ce système permet le transfert de connaissances déclaratives (règles symboliques) d'un module symbolique vers un module connexionniste (réseau de neurones artificiel - RNA) à travers un convertisseur de règles en réseaux. Les connaissances du réseau

---

[1] La définition de "l'intelligence", ainsi que les tentatives de la quantifier, demeurent des sujets controversés. Nous considérons l'homme comme notre modèle d'intelligence, et pour nous, les systèmes dits "intelligents" sont ceux capables de reproduire certaines caractéristiques propres au raisonnement humain.

ainsi obtenu sont affinées par un processus d'apprentissage à partir d'exemples. Ce raffinement se fait soit par ajout de nouvelles connaissances, soit par correction des inconsistances, grâce à l'utilisation d'un réseau constructif. Une méthode d'extraction incrémentale de règles a été intégrée au système INSS, ainsi que des algorithmes de validation des connaissances qui ont permis de mieux coupler les modules connexionniste et symbolique. INSS a été conçu de façon à créer un système d'apprentissage automatique pour l'acquisition constructive (incrémentale) de connaissances. Le système a été testé sur plusieurs applications, en utilisant des problèmes académiques et des problèmes réels (diagnostic médical et contrôle d'un robot autonome).

Ce mémoire est organisé de la manière suivante :

- Le Chapitre 2 est consacré à une description des systèmes intelligents (systèmes experts) et des méthodes d'acquisition et de représentation des connaissances. Nous présentons plus particulièrement les systèmes d'apprentissage automatique, que nous avons divisés en deux grands groupes : les approches symboliques d'apprentissage et les approches connexionnistes d'apprentissage. Nous détaillons le mode de fonctionnement des principales méthodes d'apprentissage qui sont liées directement aux études développées dans le cadre de cette thèse. A l'issue de ce chapitre nous présentons une liste de solutions hybrides utilisées dans le domaine de l'apprentissage automatique, et nous expliquons notre choix des systèmes hybrides neuro-symboliques.

- Le Chapitre 3 est consacré à un état de l'art critique sur les systèmes hybrides neuro-symboliques. Nous présentons une liste d'exemples de ce type de systèmes, en les classifiant par rapport à leurs principales propriétés. Nous avons choisi d'étudier plus en détails deux d'entre eux en raison de leurs propriétés et de leurs caractéristiques : le système SYNHESYS et le système KBANN. Nous décrivons leur implémentation, car ces deux systèmes ont beaucoup influencé le développement du système INSS. Nous présentons aussi une introduction aux systèmes connexionnistes de raffinement de connaissances, une classe de systèmes dont les systèmes SYNHESYS et KBANN sont des exemples typiques.

- La Chapitre 4 concerne la description du système hybride que nous avons développé, le système INSS. Nous présentons l'organisation générale du système INSS et ensuite nous détaillons le mode de fonctionnement de chacun des modules : le module d'inférence symbolique, le module d'insertion de règles symboliques dans un réseau, le module de simulation du réseau connexionniste, le module d'extraction de règles à partir des réseaux et le module de validation de connaissances. Puis nous concluons par une analyse des propriétés de notre système.

- Le Chapitre 5 est consacré à la présentation des résultats pratiques obtenus avec l'utilisation du système INSS. Ce chapitre est divisé en quatre sous-sections, qui décrivent les principales applications que nous avons développée avec l'aide d'INSS : les problèmes du Moine (problème académique d'apprentissage), le système d'aide au diagnostic des comas toxiques (système expert d'aide au diagnostic médical développé en coopération avec le CHU de Grenoble), l'étude du problème de la balance (un problème de modélisation du développement cognitif chez l'enfant), et une application de contrôle d'un robot autonome (en utilisant d'abord un simulateur puis le vrai robot). Les résultats obtenus dans les différents tests réalisés montrent bien les propriétés et les qualités de notre système.

Les chapitres 2 et 3 constituent un état de l'art respectivement dans le domaine de l'apprentissage automatique et dans celui des systèmes hybrides neuro-symboliques. C'est pourquoi nous conseillons aux lecteurs qui ont des connaissances bien fondées sur l'apprentissage automatique et les réseaux de neurones de passer directement au chapitre 3. Par contre, nous pensons que la lecture du chapitre 3 (et en particulier de la section 3.3) est très importante pour la bonne compréhension des sujets qui seront abordés par la suite. De plus, nous avons ajouté à la fin de ce mémoire un glossaire et une liste des abréviations utilisées, qui pourront aider les lecteurs à mieux parcourir cet ouvrage.

2. APPRENTISSAGE AUTOMATIQUE

Dans ce chapitre, nous allons présenter un aperçu des différents outils d'apprentissage automatique, parmi lesquels on distingue deux axes principaux de recherche : les approches symboliques et les approches connexionnistes (sub-symboliques). Nous allons tout d'abord proposer une brève introduction sur l'importance et sur le rôle de ces algorithmes d'apprentissage automatique dans le domaine de *l'Intelligence Artificielle* (I.A.), des *Systèmes Experts* et des *Systèmes Intelligents* [KAN 92]. Ensuite, nous allons présenter plus en détail les systèmes d'apprentissage symbolique et les systèmes d'apprentissage connexionniste, pour aboutir au concept de système d'acquisition de connaissances hybride. Nous nous intéresserons plus particulièrement aux Systèmes Hybrides Neuro-Symboliques (S.H.N.S. [ORS 95]).

2.1 LES SYSTEMES EXPERTS ET L'ACQUISITION DE CONNAISSANCES

Le terme Intelligence Artificielle (I.A.) a été introduit par McCarthy en 1956 pendant une conférence au Dartmouth College, et depuis, ce terme a été retenu pour représenter le domaine. L'I.A. est directement liée aux concepts de Systèmes à bases de Connaissances, Systèmes Experts, Systèmes Intelligents, Acquisition de Connaissances, Apprentissage Automatique, parmi d'autres sujets d'étude et d'application pratique. (voir le Tableau 1 - Applications de L'I.A.). Dans cette thèse, nous allons nous concentrer plutôt sur le sujet IV indiqué dans le tableau ci-dessous, avec des études complémentaires à propos des items II et V.

Domaines de l'I.A. : Sujets d'étude	**Exemples d'Applications Pratiques**
I - Vision par Ordinateur : Reconnaissance d'Images, Analyse d'Images et Traitement de Signaux	Reconnaissance de formes, classification d'images, reconnaissance de textes, conduite autonome de véhicules.
II - Robotique Intelligente et Evolution Artificielle	Systèmes réactifs, robotique autonome, robots intelligents, contrôle et planification de trajectoires, *animats*.
III - Traitement du Langage Naturel	Traduction automatique, reconnaissance de la parole, analyse automatique de textes, indexation et recherche automatique d'informations.

IV - Raisonnement Automatique et Acquisition de Connaissances : Systèmes Experts, Systèmes Intelligents et Systèmes Hybrides, Apprentissage Automatique, Représentation de Connaissances	Systèmes intelligents pour l'exploration de données (*data mining, knowledge data discovery-KDD*), systèmes d'aide au diagnostic (e.g. médical, industriel), systèmes pour la classification et/ou la prévision, systèmes d'aide à la décision, systèmes de raisonnement fondé sur des cas (*CBR*), systèmes à base de connaissances (*KBS*).
V - Modélisation Cognitive	Proposition, analyse et validation de modèles du comportement cognitif humain (modèles fondés sur des études psychologiques).
VI - I.A. Distribuée et Systèmes Multi-Agents	Modélisation et conception de systèmes fondés sur des interactions avec multiples modules, agents coopératifs et collaboratifs (e.g. modèles sociaux, prise de décisions en groupe, comportements collectifs en robotique, applications militaires), théorie des jeux.
VII - Logique Formelle	Logiques pour l'I.A. (e.g. logique temporelle), mécanisation des logiques, langages d'I.A. (e.g. Prolog), preuve automatique de théorèmes.

Tableau 1 - Applications de L'I.A. (Liste non exhaustive)

Le terme d'Intelligence Artificielle est un peu provocateur, puisque le concept d'un agent ou processus intelligent n'est pas très bien définie : il n'existe pas de vrai consensus sur la frontière entre ce qui est intelligent et ce qui ne l'est pas, ni au sujet de l'évaluation de l'intelligence d'un système.

D'une façon générale, on peut dire que, " le but de l'I.A. est de construire des systèmes qui puissent exhiber un comportement intelligent et réaliser des tâches complexes avec un niveau de compétence qui est équivalent, sinon supérieur, à celui exhibé par des experts humains " [NIK 97]. Quand les recherches en I.A. ont démarré, un grand effort a été déployé dans la recherche d'un outil capable de résoudre n'importe quel type de problème (les systèmes nommés "General Problem Solvers - GPS"). Ce type d'approche s'est avéré trop ambitieux, car on n'est pas capable de créer un seul et unique système qui puisse traiter en même temps toute une large gamme de problèmes complexes. En conséquence de cet échec, un changement de cap est survenu. Les chercheurs ont conclu que des systèmes spécialisés utilisés dans la résolution de certains problèmes pourraient être développés plus facilement, et quand même être appliqués

dans différents domaines (les connaissances sur le problème changent , mais les mécanismes de base restent à peu près les mêmes). A la place de chercher une seule et unique solution générale omnipotente de résolution de problèmes, il vaut mieux alors disposer d'outils spécialisés dans la résolution de tâches plus spécifiques.

Les systèmes spécialisés dans la solution de problèmes plus spécifiques sont donc apparus. Ces systèmes ont étés nommés des *systèmes experts* [HAY 83]. Ce type de système appartient à la classe des *systèmes à bases de connaissances (Knowledge-Based Systems - KBS)*.

2.1.1 Les Systèmes Experts

Feigenbaum [FEI 79] a défini un système expert comme étant " un logiciel intelligent qui utilise des connaissances et un procédé d'inférence pour résoudre des problèmes. Ces problèmes sont assez difficiles à résoudre et requièrent une expertise humaine significative pour arriver à une solution. Les connaissances d'un système expert sont composées de faits et de heuristiques ". Cette définition est d'une façon générale encore valable, comme on pourra le constater dans la suite de ce chapitre. Pourtant, elle est un peu dépassée quand on fait référence aux *Systèmes Experts de Deuxième Génération* [NIK 97, DAV 93]. Ce deuxième type de système expert est plus développé et possède des propriétés telles que l'intégration de différentes méthodes de raisonnement et l'automatisation de l'acquisition de connaissances. Comme on le verra dans la suite, notre étude a été dirigée plutôt vers ce type particulier de système expert.

Donc, les systèmes experts sont faits pour résoudre des problèmes dans un domaine spécifique d'expertise et ne sont pas des outils généraux de résolution de problèmes. Par exemple, un système expert ne va probablement jamais résoudre des problèmes d'un domaine aussi large que le domaine médical, mais peut toutefois s'adresser parfaitement à des sous-domaines spécifiques du domaine médical. Cela a été démontré par des systèmes comme MYCIN [SHO 76] et DENDRAL [LIN 80], des systèmes experts très connus utilisés pour le diagnostic de maladies infectieuses sanguines et pour l’identification de structures moléculaires respectivement. Actuellement, les systèmes experts couvrent un large domaine d’application [DUR 93, HAY 83], dans lequel on peut distinguer deux principales applications : les systèmes d’aide à la décision et

les systèmes d'aide au diagnostic. Ces systèmes doivent être conçus pour *aider* l'homme dans la réalisation de tâches difficiles et ne doivent jamais être utilisés pour le *remplacer* complètement.

Les systèmes experts ont des caractéristiques particulières qui les distinguent des autres systèmes informatiques. Parmi ces caractéristiques, on trouve des éléments tels que : l'architecture particulière de ses modules ; la codification des connaissances dans une base de connaissances ; la séparation des connaissances de la partie de contrôle ; la capacité à raisonner sur des données incomplètes, inexactes et floues (sur certains systèmes experts) ; les outils d'acquisition de connaissances ; la possibilité d'explication des résultats. Ces capacités sont principalement dues au fait que ces systèmes sont capables de dériver des solutions à partir d'une heuristique, plutôt que par les approches algorithmiques employées dans les systèmes informatiques conventionnels.

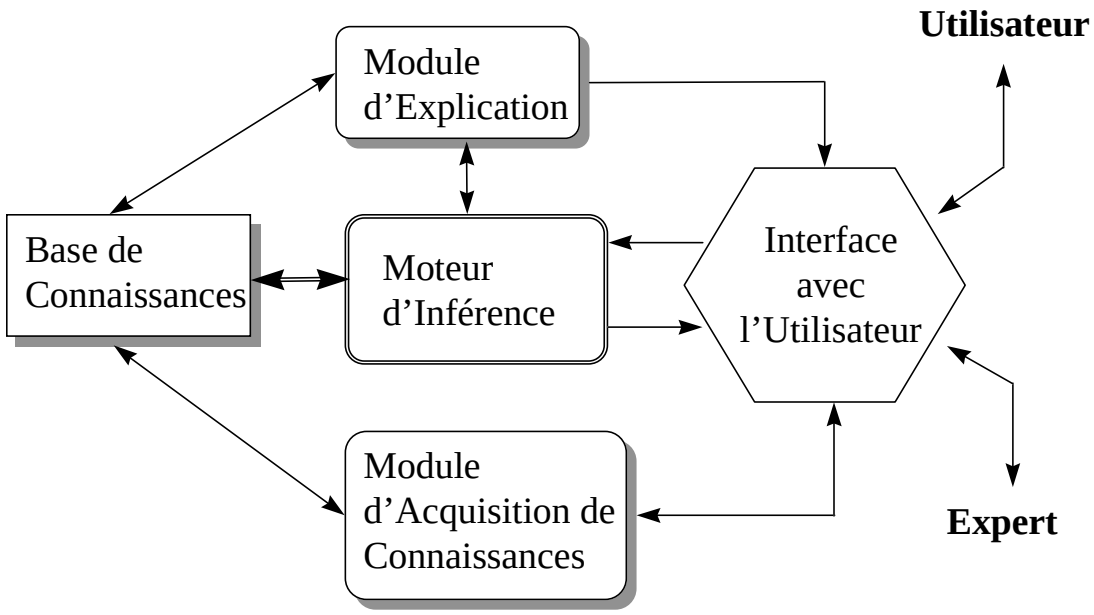


Figure 1 - L'Architecture d'un Système Expert

Les systèmes experts sont en général composés d'une architecture à cinq modules (voir Figure 1 - L'Architecture d'un Système Expert) :

1. ***Base de Connaissances*** : la base de connaissances contient les connaissances spécifiques du domaine qui sont utilisées pour résoudre un certain problème. Elle est composée

usuellement par des règles et des faits. Nous allons aborder plus en détail les aspects liés à ce module dans la section 2.1.2;

2. ***Moteur d'Inférence*** : le moteur d'inférence permet de manipuler les données stockées dans la base de connaissances de façon à pouvoir résoudre les problèmes posés au système. Le moteur d'inférence reste séparé des connaissances du domaine et généralement il n'est pas dépendant de son application à un problème particulier. Cela ne veut pas dire que l'on a un seul type de moteur d'inférence, car chaque formalisme de représentation possède ses propres techniques d'inférence et donc, le mécanisme d'inférence reste directement lié au type de représentation des connaissances employé. Un moteur d'inférence est typiquement fondé sur une règle d'inférence (permet de réaliser un raisonnement et de déduire de nouvelles connaissances à partir des connaissances de la base) et une stratégie de recherche [HAY 83, NIK 97];
3. ***Module d'acquisition de connaissances*** : ce module permet de créer, ajouter et maintenir les connaissances nécessaires pour la résolution d'un problème dans un domaine d'application. L'acquisition de connaissances peut être faite par explicitation de connaissances d'un domaine avec l'aide d'un ingénieur de connaissances et de l'expert, ou à travers des processus semi-automatiques ou totalement automatiques d'acquisition de connaissances. Ces méthodes automatiques sont connues comme *méthodes d'apprentissage automatique* ("machine learning methods"). La Figure 2 présente un schéma simplifié du processus d'acquisition de connaissances. Dans la section 2.1.3, nous allons présenter plus en détails les méthodes d'acquisition de connaissances;
4. ***Module d'explication*** : ce module permet à l'utilisateur de connaître les processus et le raisonnement qui ont amené le système à donner une certaine réponse. De cette façon, il est possible à l'utilisateur de poser des questions et interagir avec le système afin de comprendre les réponses fournies. Le système est capable de *justifier* ses réponses.
5. ***Interface avec l'utilisateur*** : l'interface permet l'échange d'informations entre l'utilisateur, l'expert, l'ingénieur de connaissances et les différents modules du système expert.

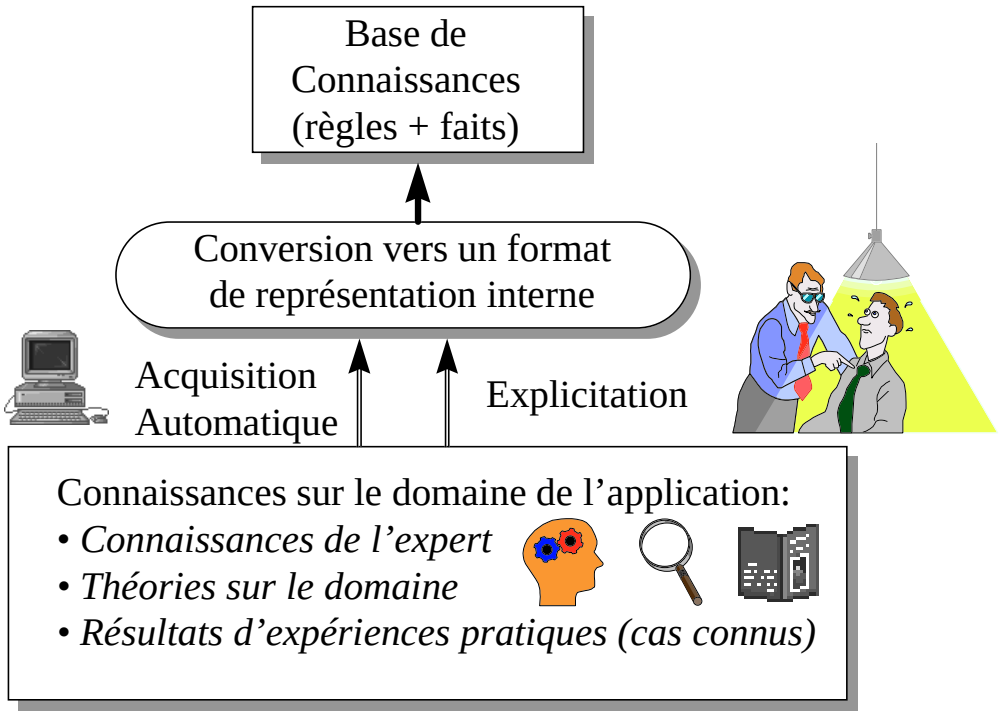


Figure 2 - Acquisition de Connaissances

Les systèmes experts permettent d'automatiser certains aspects du raisonnement humain, en utilisant des méthodes d'inférence telles que l'induction, la déduction et le raisonnement par analogie. Ces méthodes nous permettent d'inférer des propositions en utilisant des règles d'inférence (e.g. modus ponens, modus tollens, double négation, règle de la chaîne, résolution) [NIL 80, STE 93]. Certains types de moteurs d'inférence sont très connus, comme : la méthode de chaînage avant et la méthode de chaînage arrière.

Dans les méthodes de chaînage avant, c'est à partir des prémisses (faits) qu'on arrive aux conclusions. Nous avons utilisé dans le développement de nos recherches un système exploitant ce type de méthode d'inférence. Ce système s'appelle le langage **CLIPS** (*C Language Integrated Production System*) [GIA 93, CUL 93], et il sert à la construction de systèmes experts. CLIPS a été développé par le *Software Technology Branch* de la NASA aux U.S.A. Il inclut un moteur d'inférence par chaînage avant fondé sur l'algorithme *Rete* [FOR 82]. Nous l'avons utilisé pour implémenter le module symbolique de notre système hybride INSS (voir Section 4.1).

La méthode de chaînage arrière part des conclusions. Elle cherche à trouver, à partir des conclusions, les prémisses nécessaires à leur déclenchement. Un autre exemple d'outil informatique utilisé pour la construction de systèmes experts est le langage Prolog (Programmation en Logique) [COL 85, BRA 90]. Ce langage utilise une méthode de chaînage arrière dans l'implémentation du moteur d'inférence par une méthode de résolution. Ce type de méthode automatique de raisonnement, employé par les langages comme Prolog, repose sur l'idée que "tout ce qui n'est pas vrai est faux et tout ce qui n'est pas faux est vrai". C'est pour cela que de tels systèmes ont besoin d'une théorie (base de règles et faits) qui soit à la fois correcte et complète sur un domaine.

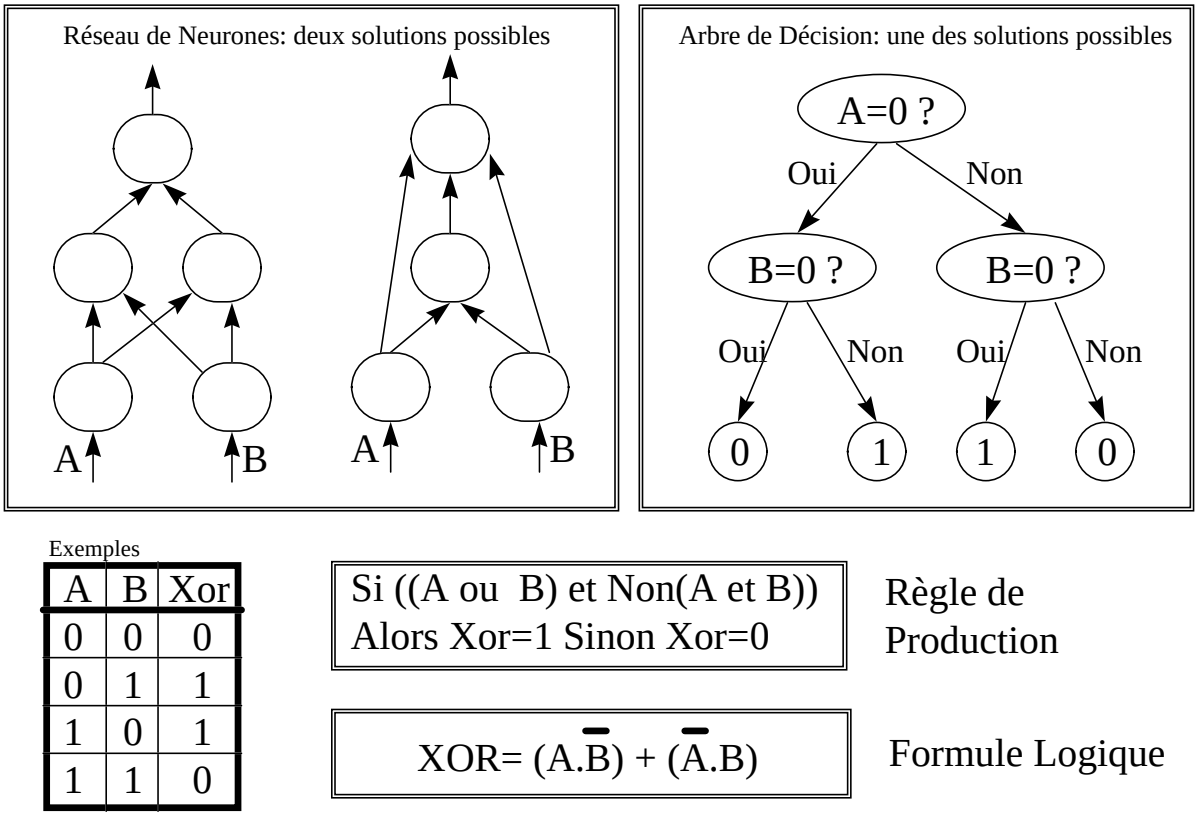
En conclusion, les systèmes experts utilisent des méthodes de raisonnement automatique. Ils nécessitent un moteur d'inférence spécifique et une base de connaissances. Cette base doit être composée par des structures de données bien adaptées au traitement automatique des informations et au moteur d'inférence employé par le système.

2.1.2 La Représentation des Connaissances

De nombreuses activités "intelligentes" de l'être humain, aussi bien dans sa vie quotidienne que dans son activité professionnelle, reposent sur l'exploitation d'une masse importante d'informations, de faits, d'expériences et de connaissances plus ou moins spécifiques d'un domaine particulier. De ce fait, une partie importante des travaux de recherche et de développement en I.A. concerne la conception de formalismes permettant la mise en oeuvre de systèmes à bases de connaissances, et plus spécifiquement l'étude des différentes façons de spécifier et de créer les bases de connaissances. Nous allons nous concentrer ici sur les problèmes de la représentation des connaissances et de leur façon de s'adapter aux différents systèmes de raisonnement par ordinateur.

La représentation de connaissances est un sujet délicat à aborder car il n'existe pas un unique formalisme de représentation de connaissances accepté par tous. Les connaissances sur un domaine peuvent être exprimées de plusieurs façons. Par exemple, dans la Figure 3 nous présentons les différentes façons de représenter les connaissances sur un problème très

spécifique : la fonction logique *Ou Exclusif* (XOR). Bien que la fonction XOR soit une fonction logique très simple, cette figure montre qu'il existe déjà différentes façons de la représenter.



Exemples

A	B	Xor
0	0	0
0	1	1
1	0	1
1	1	0

Si ((A ou B) et Non(A et B))
Alors Xor=1 Sinon Xor=0

Règle de Production

$XOR = (A.\overline{B}) + (\overline{A}.B)$

Formule Logique

Figure 3 - Différentes façons de représenter les connaissances sur la fonction logique XOR

La Figure 3 permet de montrer qu'on peut coder les connaissances à propos d'un sujet particulier de différentes façons. L'important est donc qu'on puisse représenter les connaissances de façon à pouvoir mieux les exploiter par la suite. Ce processus de conversion des connaissances sur un sujet dans un format particulier est ce que l'on appelle le "*processus de représentation de connaissances*". Une fois codifiées, elles pourront être manipulées et exploitées par des outils informatiques.

Nous montrons, dans la Figure 2, que les connaissances utilisées dans un système expert appliqué à un domaine spécifique peuvent être obtenues à partir de différentes sources, à savoir : l'explicitation directe par l'expert de ses connaissances ; l'interrogation de l'expert de façon à

extraire ses connaissances; l'analyse et la représentation des connaissances contenues dans de livres, manuels, etc.; l'observation des résultats obtenus dans des expériences pratiques; et aussi, à travers l'emploi d'outils d'analyse, de traitement et de manipulation automatique de connaissances qui permettent d'induire ou de déduire de nouvelles connaissances.

Nous allons tout d'abord classer selon leur source les connaissances en deux grands groupes principaux :

1. **Connaissances Empiriques :** les connaissances empiriques, c'est-à-dire les connaissances expérimentales, sont représentées par l'ensemble des cas pratiques observés sur un sujet (ensemble d'exemples). Ce sont des connaissances "*pures*" qui n'ont pas été traitées, analysées ou modifiées. Ces connaissances représentent les résultats d'expériences ou les exemples de cas pratiques ; elles n'ont pas encore subi de transformations en vue d'obtenir une théorie plus générale sur le domaine. On peut dire que ce sont des connaissances de *bas niveau*. Ce type de connaissances peut être utilisé dans les systèmes experts à travers l'application de processus de raisonnement par induction (obtention de connaissances de plus haut niveau à partir des connaissances empiriques de base - la généralisation), ou à travers des processus de raisonnement par analogie (rapprochement des cas nouveaux aux cas anciens connus les plus proches - la similarité).

2. **Connaissances Théoriques :** les connaissances théoriques modélisent les connaissances sur un sujet à l'aide d'une théorie correspondant au problème posé. Elles sont des connaissances "*traitées*" qui ont été obtenues à partir de l'analyse des connaissances de base. Ce type de connaissances représente une généralisation du savoir. Ce sont des connaissances dites de *haut niveau*. Elles sont habituellement représentées par des structures symboliques, telles que les règles de production, les modèles mathématiques, les réseaux sémantiques, les objets structurés, les propositions.

Donc, les connaissances dont on peut disposer sur un domaine constituent une combinaison des connaissances empiriques et des connaissances théoriques disponibles. *Il faut souligner que*

ces deux types de connaissances forment deux ensembles ayant une intersection, mais qui peuvent aussi être totalement complémentaires [TOW 91]. Il faut préciser aussi qu'habituellement les connaissances sur un domaine ne sont ni parfaitement correctes, ni complètes. Par conséquent, on doit envisager d'exploiter au maximum les connaissances disponibles tout en tirant profit de ces deux types de connaissances.

Les différents systèmes, à savoir les systèmes experts de première génération ou les systèmes d'apprentissage symbolique tels que les réseaux connexionnistes, les arbres d'induction ou les systèmes de raisonnement fondé sur des cas (voir la section suivante sur l'acquisition de connaissances), ne permettent pas en général de bien exploiter ces deux types de connaissances. Un des principaux buts de notre recherche dans cette thèse est de proposer des nouvelles solutions pour pouvoir travailler à la fois sur ces deux types de connaissances.

Les connaissances empiriques et théoriques doivent être converties dans une représentation compatible avec les outils informatiques de raisonnement automatique afin de pouvoir être exploitées. Parmi les différentes approches de représentation de connaissances, on trouve deux grands groupes opposés : *les approches connexionnistes* et *les approches symboliques.*

2.1.2.1 Approches Connexionnistes

Les approches connexionnistes, aussi dénommées approches sub-symboliques, sont inspirées des études de l'architecture du cerveau. La représentation des connaissances dans les réseaux connexionnistes est fondée, avec beaucoup de simplification, sur les structures de notre cerveau telles que les neurones et leurs connexions. Dans cette approche, toutes les connaissances sont représentées par des liaisons entre les unités (neurones artificiels) et leurs poids synaptiques (valeurs) associés. Les *Réseaux Connexionnistes* sont aussi appelés *Réseaux d'Automates, Réseaux Neuronaux Artificiels* (RNA) ou *Réseaux Neuromimétiques*. Même si l'étude des réseaux connexionnistes est encore jeune, on peut trouver des nombreux ouvrages décrivant le sujet [AMY 96, JOD 94a, JOD 94b, AMA 95, ADB 94, HER 94, NAD 93, BOU 91]. La Figure 4 montre un exemple de réseau connexionniste.

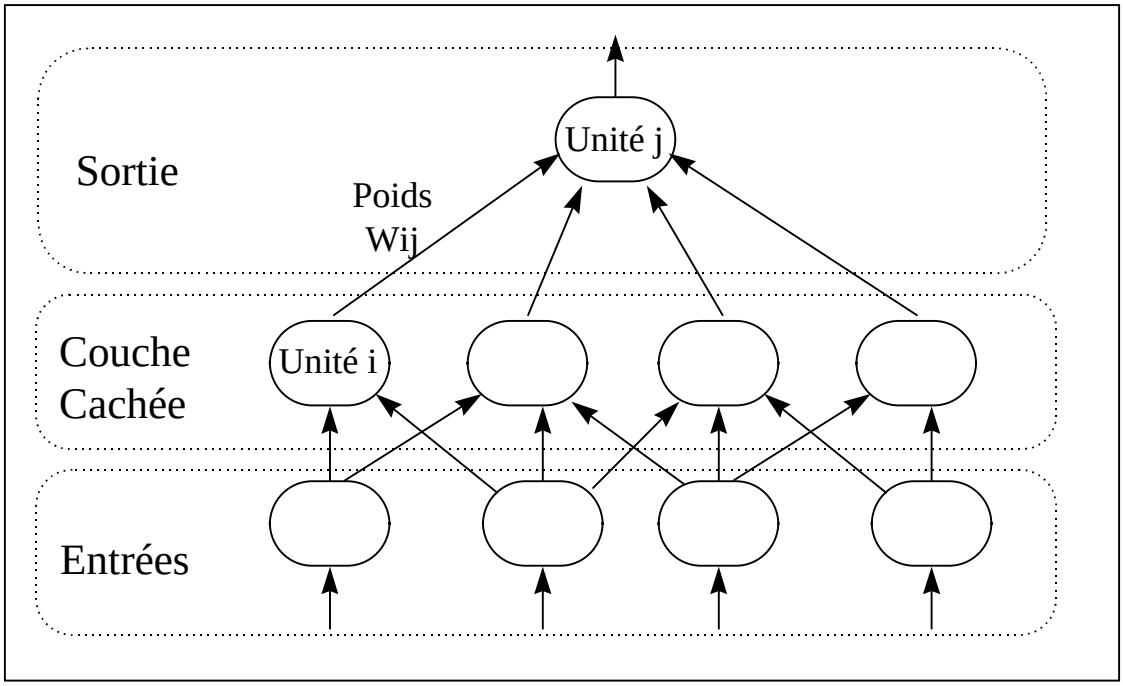


Figure 4 - Les Réseaux Connexionnistes

Les réseaux connexionnistes utilisent des méthodes d'apprentissage à partir d'exemples qui leur permettent d'ajuster leurs poids synaptiques. La modification des poids est faite de façon à modifier le comportement résultant de l'activation du réseau ; on cherche ainsi à induire (généraliser) des connaissances à partir d'une base d'apprentissage. Une bonne définition des réseaux connexionnistes est donnée par Giacometti [GIA 92] : il décrit ces réseaux comme capables de représenter le "savoir-faire" (comportement) dans un domaine d'application. Le "savoir-faire", qui représente les connaissances pratiques, apparaît en opposition au "savoir-que", qui lui représente les connaissances théoriques.

Lorsque les réseaux connexionnistes ont leurs connaissances codées sur la structure d'interconnexion et les poids des connexions, il est très difficile d'interpréter directement cette forme de codage des connaissances. Les connaissances ne forment qu'un ensemble de valeurs numériques décrivant les connexions et par conséquent le comportement d'activation du réseau. Dans la section 2.3, nous allons étudier les réseaux connexionnistes avec plus de détails.

2.1.2.2 Approches Symboliques

Les approches symboliques sont employées pour permettre de représenter les connaissances de type "savoir-que" [GIA 92]. Dans ce type d'approche, on va représenter les connaissances par des propositions qui vont exprimer le savoir d'un expert. Les propositions peuvent être représentées de différentes façons, dont la plus générale est le langage naturel. Donc, il s'agit de représenter les connaissances (savoir de l'expert) exprimées dans le langage naturel, par des propositions ou par des formules construites selon une syntaxe précise. Cette forme d'expression du savoir a été fortement influencée par la logique mathématique.

Le "savoir-que" d'un expert d'un domaine donné sera ainsi formalisé par un ensemble de propositions de même forme syntaxique, pouvant être ensuite manipulées selon des règles de la logique classique (règles d'inférence). Les connaissances représentées à travers des approches symboliques sont facilement interprétables étant donné le codage utilisé qui est assez proche du langage naturel humain. Nous allons nous concentrer ici sur les différents modes de représentation de connaissances proposés dans le cadre des approches de l'I.A., à savoir :

a) Description de Cas Pratiques

b) Règles de Production et Formules Logiques

c) Réseaux Sémantiques

d) Objets Structurés et Frames

e) Méta-Connaissances

2.1.2.2.a Description de Cas Pratiques

C'est la forme de représentation de connaissances la plus simple. Les connaissances sont décrites par une liste d'attributs et ses respectives valeurs associées, tel qu'on les a observée dans une expérience pratique. Le simple fait de sélectionner quelques attributs spécifiques et d'associer une classification au cas peut donner l'origine à une information plus structuré comme les règles de production. Exemple : Température = 36.5 °C, Age = 38 ans, Pression Artérielle = 80, ...

2.1.2.2.b Règles de Production et Formules Logiques

Les propositions, originellement en langage naturel, doivent être converties dans une syntaxe bien précise, pouvant donner ainsi l'origine à des règles de production et des formules logiques. Une règle de production est un quantum de connaissance, déclaratif et autonome, de la forme générale :

Si *[conditions]* ***Alors*** *[conclusion]*

La partie *conditions* (antécédent) est constituée d'une formule logique qui doit être vérifiée pour que la règle s'applique. La partie *conclusion* (conséquent) correspond habituellement au déclenchement d'une action ou d'une autre règle. Les formules peuvent être des types : atomique, moléculaire ou généralisé [GIA 92] :

- Formules atomiques : utilisent des symboles atomiques et des relations unaires, binaires, ternaires, etc. Exemples : Froide(neige); Couleur(neige, blanche); Lancer(Philippe, neige, Denise);
- Formules moléculaires : peuvent être définies à partir des formules atomiques combinés à l'aide de connecteurs logiques tels que '$\wedge$' (ET logique), '$\vee$' (OU logique), '$\neg$' (NON logique). Exemple : Froide(neige) $\wedge$ Couleur(neige,blanche);
- Formules généralisées : peuvent être construites à partir des formules moléculaires. Il s'agit tout d'abord de substituer à des constantes (symboles atomiques) d'une formule moléculaire par des variables. Nous obtenons ainsi des fonctions sur les formules. Ces fonctions sont représentées par des quantificateurs tels que '$\exists$x' (il existe au moins une valeur possible pour la variable x), '$\forall$x' (pour toutes les valeurs possibles que la variable x peut assumer). Exemple : ($\exists$x)(Couleur(x,blanche))

Les règles de production sont classées selon leur pouvoir d'expressivité en règles d'ordre 0, d'ordre 0^+, ou de premier ordre (ordre 1). Les règles d'ordre 0 contiennent uniquement des formules atomiques ou moléculaires telles que nous les avons définies. Les règles d'ordre 0^+ contiennent en plus des formules du type 'x $\in$ I' (où 'x' est une variable décrivant une situation ou

un événement, et 'I' est un ensemble de valeurs possibles, c.-à-d. un intervalle auquel x peut s'appliquer). Les règles de premier ordre sont des règles qui peuvent contenir des formules généralisées et par conséquence des quantificateurs. Ce dernier type de règle est aussi appelé de prédicat. Un type particulier de prédicat de premier ordre simplifié, nommé clause de Horn, est utilisé par des langages comme le Prolog.

Les règles de production sont ensuite traitées par les moteurs d'inférence. Ceux-ci permettent d'appliquer des méthodes de raisonnement formel déductif en utilisant les règles disponibles. Le calcul propositionnel est une des méthodes (système formel) qui permet de raisonner sur des formules moléculaires. Le calcul des prédicats à son tour est un système formel de raisonnement sur des prédicats (formules généralisées). Une description plus détaillée de ces méthodes de raisonnement formel déductif est proposée par Giacometti dans sa thèse [GIA 92].

Enfin, des règles de production et des méthodes de raisonnement automatique plus particulières ont été proposées afin d'augmenter le pouvoir de représentation et d'inférence des systèmes experts. On peut citer comme exemples [REY 97] :

- Les règles floues (e.g. règles de Mamdani ou de Sugeno [JAN 96]).

  Exemple : **Si** x $\in$ Fuzzy_Set_I **Alors** y = a.x + b ('a' et 'b' sont des valeurs constantes)

- Les règles obtenues à partir des réseaux connexionnistes (e.g. règles M-of-N [TOW 91]).

  Exemple : **Si** *2-parmi-4* (F_1(a), F_2(b),F_3(c),F_4(d))

  Alors *conclusion_est_vraie* **Sinon** *conclusion_est_fausse*

- Les règles bayésiennes (probabilistes) et les coefficients de vraisemblance (facteurs de certitude)

  Exemple : **Si** *possède _propriété_X* (exemple$_i$)

  Alors *probabilité_d'appartenir_à_classe_A* (exemple$_i$) = Valeur1 et

  probabilité_de_non_appartenir_à_classe_A (exemple$_i$) = Valeur2

- Les règles d'inférence graduelle.

  Exemple : (**Si**) plus *X* est un oiseau typique (**Alors**) plus on est certain que *X* vole

- Les règles de "haut niveau" (règles de modulation ou de contexte)

  Exemple : **Si** *Xi* est plus grand par rapport à *Xj* **Alors** moins important sera *Xj*

Ces règles de production jouent sur l'incertitude, sur les types de données utilisées (données discrètes ou continues), sur les relations entre données et valeurs constantes (appartenance à un intervalle, supérieur, inférieur), sur les relations entre une donnée et une autre et même sur la *force* des relations. Nous reviendrons sur ce sujet des règles de production particulières (règles de *haut niveau*) dans la section 5.3.

2.1.2.2.c Réseaux Sémantiques

Un réseau sémantique est un graphe étiqueté dans lequel les noeuds représentent des objets ou des concepts, et les arcs des relations entre eux. Issus des travaux de psychologie cognitive sur l'organisation de la mémoire (mécanismes d'association), les réseaux sémantiques ont donné lieu à de nombreux systèmes et langages de représentation des connaissances, comme par exemple : NETL et KL_ONE [FIN 79]. Ces réseaux permettent notamment d'exprimer une hiérarchie ou une taxonomie de concepts et les relations de proximité entre concepts [HAT 90, GIA 92]. Les réseaux sémantiques constituent donc un moyen de structuration des bases de connaissances.

Grâce au mécanisme d'héritage de propriétés, un noeud d'un réseau sémantique peut hériter des propriétés des noeuds qui le subsument (c.-à-d. qui sont plus hauts que lui dans la hiérarchie). Ce mécanisme constitue un mode économique de raisonnement, permettant notamment à un concept particulier de posséder automatiquement les propriétés des concepts plus généraux dont il est issu. On retrouve ce mécanisme dans les représentations par objets, actuellement beaucoup plus utilisées que les réseaux sémantiques.

2.1.2.2.d Objets Structurés et Frames

Les représentations à base d'objets structurés, telles que les *schémas, les frames, les scripts* ou *les scénarios*, permettent de regrouper ensemble des connaissances relatives à un objet, un concept, une situation ou un événement. A partir des travaux de psychologie cognitive sur

l'organisation de la mémoire chez l'homme, la notion de schéma a été proposée comme modèle de représentation prototypique d'expériences passées mises à profit pour résoudre un problème nouveau. Ce concept à été repris en I.A. sous la forme de *frames*, proposés par Minsky [MIN 75] et de *scripts* ou *scénarios*, introduits par Schank [SCH 77]. Un exemple de langage basé sur les *frames* est le langage Shirka développé au LIFIA [REC 89].

```
< frame vin :
  sorte de : boisson
  appellation : (domaine : AOC/VDQ/vin de pays)
                (défaut : vin de pays)
                (si besoin : demander appellation)
  nature : (domaine : sec/demi-sec/doux/liquoreux/corsé)
  degré alcool : (intervalle : 9 à 15)
                (défaut : 12)
  producteur : (si besoin : trouver nom sur étiquette)
  robe : ... >
```

Figure 5 - *Frame* décrivant le concept de vin

Frames et scénarios sont des entités de granularité importante regroupant de façon structurée l'ensemble des connaissances relatives à un objet, un concept ou une situation typique. Un *frame* est composé d'un ensemble d'attributs (*slots*) correspondant aux diverses notions relatives au concept représenté. A titre d'exemple, la Figure 5 donne une description possible du concept de vin [HAT 89].

2.1.2.2.e Méta-Connaissances

Une forme importante de connaissances est la méta-connaissance ou connaissance sur la connaissance [PIT 90]. Elle correspond au recul que l'on prend par rapport à un certain domaine d'activité et représente de ce fait souvent une part notable de l'expertise humaine. Elle intervient souvent, car elle est liée à la façon d'utiliser un ensemble de connaissances, aux stratégies de raisonnement et aussi à l'acquisition de nouvelles connaissances. Ce type de connaissances peut

être représenté par des structures comme celles décrites dans les items ci-dessus ; cependant nous pensons qu'il mérite d'être classé dans un groupe à part, du fait de ses caractéristiques très particulières.

2.1.3 L'Acquisition de Connaissances

Le processus du développement d'un Système Expert se compose d'une étape d'acquisition de connaissances appelée Ingénierie de Connaissances (*Knowledge Engineering*). Elle est réalisée par l'ingénieur de connaissances, le responsable de l'obtention, de la vérification, de la validation et de la structuration des connaissances. Nous discuterons de la vérification et la validation de connaissances dans la section 4.5.

L'acquisition de connaissances est un processus de rassemblement des connaissances nécessaires à la résolution d'un problème lié à une application spécifique. Ce processus inclut la codification (structuration) des connaissances dans un formalisme adapté aux outils informatiques utilisés. Il y a plusieurs façons de coder les connaissances afin de créer la Base de Connaissances, comme on l'a vu dans la section 2.1.2. En ce qui concerne l'obtention des connaissances qui vont être placées dans cette base, on peut diviser les processus d'acquisition de connaissances en deux grands groupes : obtention par explicitation de connaissances (*Knowledge Elicitation*) et obtention par apprentissage automatique ou semi-automatique (*Machine Learning*).

L'explicitation de connaissances est réalisée par l'ingénieur de connaissances, qui va disposer de différents moyens pour la réalisation de cette tâche. Les connaissances sont obtenues à partir d'interviews avec des experts et d'informations disponibles dans des livres, manuels, rapports, etc. Cette approche est très intéressante, puisqu'on profite des connaissances déjà acquises et bien élaborées. Malheureusement, on constate un certain nombre d'inconvénients :

- Le processus d'extraction de connaissances d'un expert est extrêmement ennuyeux, il prend beaucoup de temps et sa mise en place présente de sérieuses difficultés ;

- Les experts humains utilisent souvent des activités mentales d'un niveau "*subcognitif*" pour résoudre les problèmes : ils ne savent pas verbaliser ce type de connaissances ;
- Parfois les experts ne sont même pas conscients d'avoir utilisé certaines connaissances dans la résolution d'un problème et par conséquent ne les explicitent pas.

L'explicitation de connaissances par des ingénieurs de connaissances et des experts a un problème très connu des systèmes experts : il s'agit du *goulot-d'étranglement du processus d'acquisition de connaissances (The Knowledge Acquisition Bootleneck)* [NIK 97, HAY 93]. Ce processus d'acquisition de connaissances, qui est indispensable au bon fonctionnement d'un système expert, est aussi le responsable des plus grands problèmes qui surviennent lors de la construction d'un tel système. Par conséquent, les chercheurs du domaine de l'I.A. ont beaucoup investi dans l'étude et implémentation de systèmes d'acquisition automatique de connaissances. Les recherches sur l'acquisition automatique de connaissances sont à l'origine des méthodes d'apprentissage automatique, l'un des domaines de l'I.A. en plus grande expansion actuellement.

L'apprentissage automatique est une partie très important de l'Intelligence Artificielle et doit être une des principales caractéristiques des systèmes intelligents. Une des meilleures définitions de l'apprentissage automatique est celle donné par H. Simon : "L'apprentissage dans un système est indiqué par les changements qu'il subit. Ces changements sont adaptatifs dans le sens où ils rendent possible au système de réaliser une même tâche, ou des tâches tirées d'une même population, d'une façon plus efficace et plus efficiente la prochaine fois qu'elle sera réalisée" [SIM 83]. Donc, l'apprentissage automatique se fait par des outils qui permettent d'acquérir, élargir et améliorer les connaissances disponibles au système. En général, l'apprentissage implique des processus d'adaptation et de modification des structures de contrôle et/ou de représentation de connaissances du système en question.

Etant donné que ces outils cherchent à implementer des méthodes d'apprentissage automatique, le problème du goulot-d'étranglement de l'acquisition de connaissances peut être réduit, voire parfois complètement résolu. Par conséquent, l'apprentissage automatique se présente comme une alternative prometteuse pour améliorer le processus d'acquisition de connaissances. Les systèmes experts de première génération, du fait des problèmes liés à

l'acquisition de connaissances, ont évolué vers des systèmes dits de deuxième génération qui ont intégré, parmi d'autres techniques, des outils d'apprentissage automatique [NIK 97].

Les différentes méthodes d'apprentissage automatique sont classées en deux groupes : les méthodes d'apprentissage empirique (*Empirical Learning*) et les méthodes d'apprentissage fondées sur l'explication (*Explanation Based Learning*) [TOW 91].

2.1.3.1 Méthodes d'Apprentissage Empirique

Les méthodes d'apprentissage empirique sont fondées sur l'acquisition de connaissances à partir d'exemples. Elles incluent aussi des méthodes connues sous les noms d'*Instance based learning* ou *Exemplar based learning*. On peut citer comme exemples de méthodes d'apprentissage empirique les techniques suivantes : le raisonnement fondé sur des cas, les méthodes de construction de concepts et prototypes, les réseaux de neurones artificiels, les arbres de décision, les algorithmes génétiques d'induction de règles ou les méthodes du type ILP. Ces méthodes se divisent entre les méthodes d'apprentissage par analogie et les méthodes d'apprentissage par induction.

2.1.3.1.a Apprentissage par Analogie

Les approches fondées sur l'analogie essayent de faire le transfert des connaissances sur une tâche bien connue vers une autre moins connue. Ainsi, il est possible d'apprendre des nouveaux concepts ou de dériver des nouvelles solutions à partir de concepts et solutions similaires connues. Ainsi, deux notions deviennent très importants dans la définition de l'apprentissage par analogie : le *transfert* et la *similarité*. Ce type d'approche est aussi dénommée apprentissage fondé sur la similarité.

Les approches empiriques fondées sur l'analogie, telles que les systèmes de raisonnement fondé sur des cas (CBR), peuvent être considérées plutôt comme des méthodes de raisonnement que comme des méthodes d'apprentissage. Certains de ces systèmes implementent uniquement des fonctions de calcul de similarité, sans utiliser de mécanismes d'adaptation.

2.1.3.1.b Apprentissage par Induction

L'apprentissage par induction reste toujours une des principales méthodes étudiées dans le domaine de l'apprentissage automatique. Dans cette approche, on cherche à acquérir des règles générales qui représentent les connaissances obtenues à partir d'exemples. Les règles ainsi obtenues peuvent être représentées d'une façon explicite (facilement interprétables) ou d'une façon implicite avec un codage qui n'est pas toujours facile à interpréter. L'algorithme d'apprentissage par induction reçoit un ensemble d'exemples d'apprentissage et doit produire des règles de classification, permettant de classer les nouveaux exemples. Le processus d'apprentissage cherche a créer une représentation plus générale des exemples, selon une *méthode de généralisation de connaissances*. Cet type de méthodes est aussi appelé apprentissage de concepts ou bien acquisition de concepts. Parmi les approches d'apprentissage empirique par induction les plus connues, on trouve les réseaux de neurones artificiels et les arbres de décision.

L'algorithme d'apprentissage par induction peut fonctionner de façon *supervisée* ou *non-supervisée* :

- Apprentissage supervisé (*supervised learning*) : les exemples d'apprentissage sont étiquetés afin d'identifier la classe à laquelle ils appartiennent. Le but de l'algorithme de classification est de correctement classifier les nouveaux exemples dans les classes définies dans la phase d'apprentissage.

- Apprentissage non-supervisé (*unsupervised learning, clustering, discovery*) : l'algorithme d'apprentissage cherche à trouver des régularités dans une collection d'exemples, puisque dans ce type d'apprentissage on ne connaît pas la classe à laquelle les exemples d'apprentissage appartiennent. Une technique employée consiste à implémenter des algorithmes pour rapprocher les exemples les plus similaires et éloigner ceux qui ont le moins de caractéristiques communes. Ces groupes d'exemples similaires sont parfois appelés des *prototypes*.

2.1.3.2 Méthodes d'Apprentissage fondées sur l'Explication

Les méthodes inductives, telles que les réseaux de neurones ou les arbres de décision, ont besoin d'un nombre significatif d'exemples pour pouvoir bien généraliser les connaissances (induire des règles ou des concepts). Ceci restreint les possibilités d'application de ces méthodes, puisqu'on n'a pas toujours une base d'exemples assez grande et complète sur le domaine traité. Les méthodes d'apprentissage fondées sur l'explication (*Explanation-Based Methods - EBL*) [MIT 93, NIL 97] utilisent des connaissances préexistantes et un raisonnement déductif pour augmenter l'information fourni par des ensembles d'exemples. Ces méthodes sont connues sous le nom d'apprentissage par analyse (*Analytical Learning*) [MIT 97].

Dans les méthodes EBL, les connaissances sont dérivées à partir d'un simple cas par explication des raisons pour lesquelles il représente un exemple du concept appris. La méthode EBL utilise les connaissances préexistantes pour analyser, ou expliquer, comment chaque exemple observé lors de l'apprentissage satisfait les concepts existants. Ensuite, cette explication est utilisée pour différencier les attributs pertinents de l'exemple d'apprentissage de ceux qui ne le sont pas. De cette façon, l'exemple pourra être généralisé par un raisonnement logique, à la place des raisonnements statistiques souvent utilisés par les autres méthodes. Ces méthodes servent donc à améliorer les performances du système, grâce à des traitements qui rendent l'utilisation des connaissances plus efficace. *Les méthodes d'apprentissage fondées sur l'explication considèrent que les connaissances préexistantes sont à la fois correctes et complètes.*

2.1.3.3 Considérations Finales sur l'Acquisition de Connaissances

Nous avons vu dans cette section (2.1.3) les méthodes d'acquisition de connaissances classées selon les techniques d'obtention des connaissance (explicitation ou apprentissage automatique) et nous avons présenté les différentes classes de méthodes d'apprentissage automatique selon le type prédominant de connaissances employé dans l'apprentissage : les méthodes empiriques (exemples) et les méthodes d'explication (théorie). Dans les sections suivantes, nous allons présenter certaines méthodes d'apprentissage automatique de façon plus détaillée. Elles sont divisées en deux grands groupes selon le type de représentation des connaissances utilisé : les approches symboliques et les approches connexionnistes.

2.2 APPRENTISSAGE AUTOMATIQUE - APPROCHES SYMBOLIQUES

Nous allons présenter dans cette section des méthodes d'apprentissage automatique symbolique, très connues dans le domaine de l'I.A. Notre but est de mieux connaître les algorithmes, leurs avantages ainsi que leurs problèmes particuliers, afin de pouvoir proposer par la suite des solutions plus performantes. Dans la section 2.4, nous allons montrer que ces différentes méthodes peuvent être combinées afin de donner naissance à des systèmes hybrides. Nous allons tout d'abord nous concentrer sur trois méthodes principales : les arbres de décision, les algorithmes génétiques d'induction de règles et les systèmes de type CBR.

2.2.1 Arbres de Décision

Les arbres de décision sont composés d'une structure hiérarchique en forme d'arbre. Cette structure est construite grâce à des méthodes d'apprentissage par induction à partir d'exemples. L'arbre ainsi obtenu représente une fonction qui fait la classification d'exemples, en s'appuyant sur les connaissances induites à partir d'une base d'apprentissage. En raison de cela, ils sont aussi appelés arbres d'induction (*Induction Decision Trees*). Une définition un peu plus formelle des arbres de décision est la suivante : un arbre de décision est un graphe orienté, sans cycles, dont les noeuds portent une question, les arcs des réponses, et les feuilles des conclusions, ou des classes terminales.

2.2.1.1 L'Apprentissage dans les Arbres de Décision

Traditionnellement, un arbre de décision se construit à partir d'un ensemble d'apprentissage par raffinements de sa structure. Un ensemble de questions sur les attributs est construit afin de partitionner l'ensemble d'apprentissage en sous-ensembles qui deviennent de plus en plus petits

jusqu'à ne contenir à la fin que des observations relatives à une seule classe. Les résultats des tests forment les branches de l'arbre et chaque sous-ensemble en forme les feuilles. Le classement d'un nouvel exemple se fait en parcourant un chemin qui part de la racine pour aboutir à une feuille : l'exemple appartient à la classe qui correspond aux exemples de la feuille. La Figure 6 donne un exemple d'arbre de décision pour le classement d'un ensemble de cas, avec un test d'appartenance à une classe. Dans ce cas particulier, les cas dits *positifs* sont ceux qui appartiennent à la classe et les cas dits *négatifs* sont ceux qui n'y appartiennent pas.

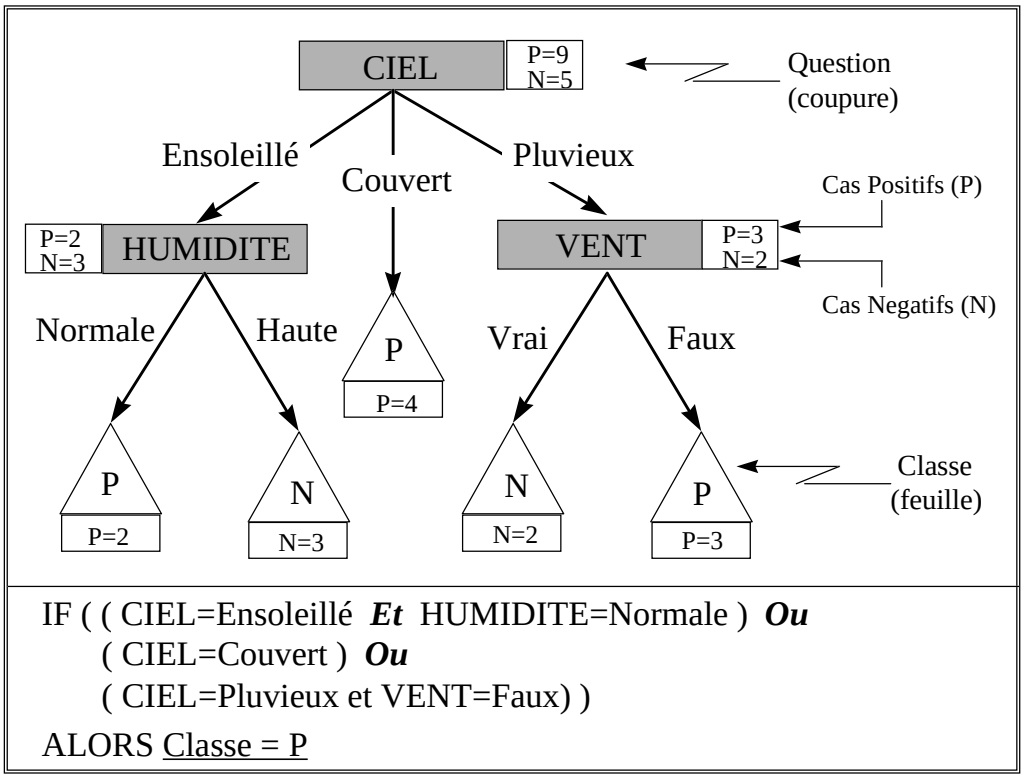


Figure 6 - Exemple d'Arbre de Décision Simple

L'apprentissage est fait à partir d'une base d'exemples qui possèdent un certain nombre d'attributs significatifs, e.g. la température, le vent, l'humidité, etc. Chaque exemple associe des valeurs particulières à chaque attribut, et comme cette méthode est une *méthode d'apprentissage supervisé*, chaque exemple est associé à une classe particulière. Nous présentons ci-dessous une base d'exemples afin de mieux expliquer le processus de construction d'un arbre (ce tableau est extrait de l'article de Quinlan [QUI 86]).

NUMERO	*CIEL*	*TEMPERATURE*	*HUMIDITE*	*VENT*	*CLASSE*
1	ensoleillé	élevé	forte	non	N
2	ensoleillé	élevé	forte	oui	N
3	couvert	élevé	forte	non	P
4	pluvieux	moyenne	forte	non	P
5	pluvieux	basse	normale	non	P
6	pluvieux	basse	normale	oui	N
7	couvert	basse	normale	oui	P
8	ensoleillé	moyenne	forte	non	N
9	ensoleillé	basse	normale	non	P
10	pluvieux	moyenne	normale	non	P
11	ensoleillé	moyenne	normale	oui	P
12	couvert	moyenne	forte	oui	P
13	couvert	élevé	normale	non	P
14	pluvieux	moyenne	forte	oui	N

Tableau 2 - Ensemble d'Apprentissage : Conditions Météorologiques

Le principe de construction des arbres est le suivant : on choisit un attribut parmi les attributs non sélectionnés, et on crée un noeud portant un test sur cet attribut. Pour chaque classe d'équivalence ainsi induite, on opère le traitement suivant : si tous les exemples de cette classe d'équivalence appartiennent à la même classe (les classes météorologiques décrites dans le tableau ci-dessus), alors on crée une feuille correspondante à cette classe, reliée au test précédent par un arc étiqueté par la valeur de l'attribut correspondant ; si tous les exemples de la classe d'équivalence considérée ne sont pas dans la même classe, alors on réitère ce processus en enlevant l'attribut précédemment considéré des attributs à sélectionner.

Par exemple, on peut construire de la manière suivante l'arbre de la Figure 6 : le premier attribut sélectionné est le 'ciel'; cet attribut sépare la base en trois ensembles : "ciel ensoleillé", "ciel pluvieux" et "ciel couvert". Ensuite, pour les exemples qui ont l'attribut "ciel ensoleillé", on prend un autre attribut qui va permettre de les distinguer, par exemple l'humidité. Une fois que tous les exemples qui ont l'attribut "humidité forte" appartiennent à la même classe et qu'il en est de même avec ceux qui ont l'attribut "humidité normale", on peut arrêter le processus de division des branches de l'arbre à ce niveau. Il faut alors terminer la construction de l'arbre pour les autres branches qui n'ont pas encore été traités jusqu'à la fin.

On peut remarquer que l'ordre dans lequel ces attributs sont sélectionnés est primordial pour la construction de l'arbre. On peut avoir différents arbres résultant de l'apprentissage d'une même base d'exemple, comme le montre la Figure 7. En effet, si l'on commence par prendre la 'température', alors l'arbre considéré sera plus grand. On suppose que les arbres qui généralisent le mieux sont les arbres de petite taille, puisque ils vont tout d'abord sélectionner les décisions plus importantes et qui séparent le plus grand nombre de cas différents. De plus, la préférence pour la construction d'arbres plus simples est une application du principe du "rasoir d'Occam" [BLU 87].

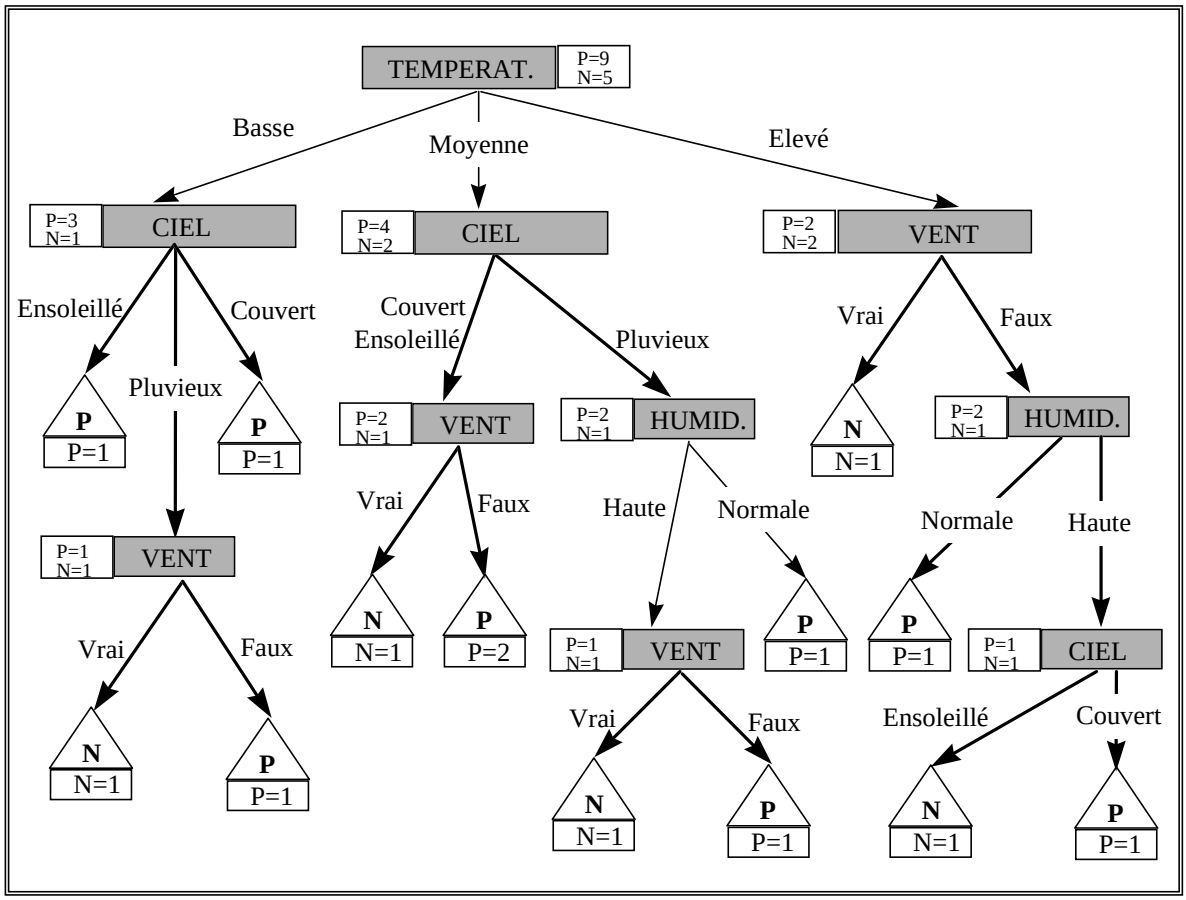


Figure 7 - Exemple d'Arbre de Décision Complexe

Quinlan a proposé une méthode pour traiter ce problème du choix de l'attribut. Il utilise une technique fondée sur la théorie de l'information de Shannon [QUI 86]. L'idée consiste à appliquer une méthode qui permet de maximiser le gain d'information apporté par chaque test.

2.2.1.2 Les Systèmes fondés sur les Arbres de Décision

Parmi les méthodes d'apprentissage fondées sur les arbres de décision les plus connues, on trouve l'algorithme ID3 [QUI 79, QUI 86] et la méthode CART [BRE 84]. Ces systèmes sont assez semblables ; ils ont été développés par deux groupes de recherche séparés et presque à la même époque. La principale différence entre ces deux systèmes réside dans le choix de la mesure utilisée pour la sélection des attributs pendant la construction de l'arbre. Cette mesure est généralement fondée sur la théorie de l'information de Shannon (entropie et gain d'information - utilisées dans ID3).

La méthode ID3 est à l'origine de plusieurs autres systèmes. L'un des plus connus est le système C4.5, développé plus récemment par Quinlan [QUI 93]. Les logiciels qui implementent ce système peuvent être obtenus avec l'ouvrage de Quinlan, ce qui a fait de C4.5 une référence de base dans les études sur les arbres de décision. A l'origine du système C4.5 se trouve la volonté de résoudre les différents problèmes rencontrés dans son prédécesseur, l'algorithme ID3. Ces problèmes sont assez caractéristiques des arbres de décision en général, à savoir :

A) Les premiers systèmes ont été conçus pour traiter des attributs avec des valeurs nominales et discrètes (*variables qualitatives*) et ne pouvaient pas traiter des attributs avec des valeurs continues (*variables quantitatives*). Il manquait des méthodes bien adaptées pour trouver les bonnes questions à poser sur les valeurs continues et qui seront postérieurement associées aux noeuds de l'arbre ;

B) Le choix du meilleur attribut qui va diviser les exemples présente lui aussi certains problèmes. Comment peut-on nous assurer que la méthode employée va nous amener à la construction de l'arbre le plus simple possible ? Malheureusement, on peut pas être sûr de l'obtenir, puisque ce type de problème devient intraitable d'un point de vue calculatoire [SHA 90]. De plus, il n'est pas difficile de trouver des exemples pour lesquels une variable a un très grand pouvoir de discrimination, mais n'aide pas beaucoup à la solution du problème (e.g.: l'âge ou le sexe d'une personne permet de discriminer les gens d'une

façon très efficiente ; toutefois est-elle une bonne variable par rapport à tous les types de problèmes de classification ?) ;

C) Parfois on ne connaît pas toutes les valeurs de chacun des attributs considérés pour la classification. Le fait que l'on doive travailler sur des domaines représentés par des information incomplètes ouvre la problématique du traitement des *attributs avec valeurs manquantes* ;

D) Les arbres de décision qui résultent de l'algorithme décrit ci-dessus souffrent du problème de surapprentissage (*overtrainning/ovefitting*). Le fait que la construction de l'arbre de décision ne s'arrête que lorsque tous les exemples de la feuille appartiennent à une même classe, suppose des ensembles d'apprentissage contenant uniquement des exemples "parfaits" (ils doivent être corrects et doivent bien représenter tout l'espace des entrées). L'arbre de décision finit par trop se spécialiser dans les exemples d'apprentissage, ce qui peut poser de graves problèmes au niveau de la généralisation aux nouveaux cas. Souvent, les bases d'apprentissage sont incomplètes et ont des exemples incorrects, et donc, cette méthode d'apprentissage ne fonctionnera pas correctement ;

E) Sur un arbre de décision, les connaissances restent "cryptées". Bien qu'il s'agisse d'une représentation symbolique des connaissances, elle peut être parfois un peu difficile à lire et à interpréter ;

F) Les arbres de décision, tels que ID3, doivent être complètement reconstruits pour prendre en compte un nouvel exemple ajouté à la base d'apprentissage. Ce type d'arbre n'est pas incrémental du point de vue de l'acquisition de données, même si sa structure est construite d'une façon incrémentale (par ajout de noeuds). Il faut recommencer tout l'apprentissage pour prendre en compte un nouvel exemple ;

G) Malgré le fait de que ces méthodes cherchent à construire des arbres simples, certaines branches peuvent être répliquées. La structure en arbre n'est pas toujours la forme la plus simple et économique pour représenter les informations. Les graphes de décision restent une alternative à la structuration en forme d'arbres ;

H) Les arbres de décision, comme leur nom l'indique, servent à obtenir une classification des donnés et ont donc une ou plusieurs sorties binaires. On ne peut pas avoir une sortie qui représente une valeur continue. Ce type d'arbres de décision n'est pas utilisable pour l'approximation de fonctions (régression) ;

I) Enfin, ces arbres de décision n'exploitent pas de méthodes permettant l'utilisation des connaissances théoriques disponibles sur le problème. Dans leur forme initiale, les arbres d'induction travaillent uniquement sur les connaissances empiriques.

Plusieurs systèmes fondés sur les arbres de décision ont pris en compte ces différentes remarques décrites ci-dessus, en essayant d'apporter des solutions plus performantes :

- **C4.5** : c'est un système dérivé de l'ID3. Il présente des propositions pour traiter et améliorer les items A (discrétisation des variables quantitatives), B (prise en compte des coûts associés au choix de chaque attribut), C (prise en compte par la fonction de sélection d'attributs), D (élagage de l'arbre à la fin du processus d'apprentissage, à partir d'un ensemble d'exemples de test de généralisation) et E (explicitation de règles symboliques du type Si-Alors à partir des arbres de décision) [QUI 93, QUI 96];
- **Assistant Professional** : il utilise une autre fonction de choix d'attributs qui permet l'élagage de l'arbre de décision (*post-pruning*), et implemente une méthode de binarisation des attributs avec des valeurs continues [CES 87, THR 91];
- **ID5R** : Il s'agit d'une méthode de construction incrémentale d'arbres de décision qui permet d'apprendre des nouveaux exemples sans avoir besoin de recommencer tout l'apprentissage [UTG 89];
- **ITI** : système d'apprentissage incrémental d'arbres de décision, développé à partir de l'ID5R. Il apporte des améliorations par rapport à l'utilisation de variables continues et de valeurs manquantes, et il permet aussi d'élaguer les arbres (*virtual pruning process*) [UTG 95];
- **IDL** : Il s'agit d'un algorithme de construction incrémental d'arbres de décision fondé sur la méthode ID5R [VAN 90, THR 91];
- **PRISM** : méthode d'apprentissage fondée sur l'algorithme ID3 de Quinlan. Il se distingue d'ID3 par le type de méthode employé pour la sélection des attributs discriminants [CEN 87, THR 91];
- **SIPINA** : système de construction de graphes de décision. Il utilise une fonction particulière de sélection d'attributs, implemente une méthode de discrétisation de

variables continues, implemente une méthode qui évite le surappentissage et permet aussi l'explicitation de règles représentées dans les graphes de décision [ZIG 92, ZIG 96b, ZIG 96c].

2.2.1.3 Discussion sur les Arbres de Décision

Les arbres de décisions sont largement utilisés, car ce type d'approche est assez stable et donne des bons résultats dans un grand nombre d'applications. Cependant, dans certains types d'applications, les arbres de décision donnent des résultats décevants. Les principaux problèmes sont ceux énumérés dans la section précédente en relation avec l'algorithme ID3.

Nous avons présenté toute une série de problèmes liées aux algorithmes comme l'ID3. Plusieurs systèmes ont été conçus afin de résoudre ou de minimiser ces problèmes. Cependant, certains aspects sont directement liés à la façon dont les arbres de décision sont construits et à leurs propriétés. Les techniques que nous avons décrites apportent des améliorations, mais les problèmes de base demeurent, à savoir :

- La difficulté de manipulation de variables quantitatives (valeurs continues). La discrétisation des données reste une solution à considérer, mais elle ne résout pas tous les problèmes de traitement d'informations non-symboliques ;

- Le choix des attributs à considérer dans les divisions (noeuds de l'arbre) ne garantit pas une solution toujours parfaite. De plus, la méthode la plus utilisée, fondée sur la théorie de l'information de Shannon, impose de fortes contraintes aux méthodes incrémentales (on est obligé de tout reconstruire) ;

- Les méthodes incrémentales, telles que l'ID5R et l'ITI, restent des méthodes assez intéressantes à étudier, mais elles imposent d'autres contraintes au niveau de la construction de l'arbre. Dans ces deux cas, le processus d'élagage semble être difficile à implementer et la complexité de la tâche de construction et de maintenance des arbres augmente ;

- Les arbres de décision n'utilisent pas de connaissances théoriques disponibles sur le problème posé. Ils exploitent uniquement des connaissances empiriques.

Enfin, la méthode de représentation des connaissances utilisée dans les arbres de décision est très dépendante du type des questions associées aux noeuds. Nous verrons dans la section 5.3 que les arbres de décision (du moins ceux qu'on a présenté ici) ne sont pas capables de traiter correctement des problèmes qui mettent en relation directe deux variables avec des valeurs continues.

2.2.2 Les Algorithmes Génétiques

La notion d'Algorithme Génétique (AG) a été introduite par John H. Holland en 1975 [HOL 75] et a été considérablement développée au cours des années quatre-vingt (cf. les ouvrages [DAV 91, MIC 92, GOL 94]). Plus récemment, une autre voie de recherche, la programmation génétique [KOZ 92], a été dérivée des AG. Nous allons ici nous focaliser sur les algorithmes génétiques et leur application à l'apprentissage de règles par induction à partir d'une base d'exemples.

2.2.2.1 Principes de Base

Les algorithmes génétiques sont des algorithmes d'exploration fondés sur les mécanismes de la sélection naturelle et de la génétique. Ils utilisent à la fois les principes de survie des structures les mieux adaptées, et de modification pseudo-aléatoire d'informations, pour former un algorithme d'exploration qui possède des caractéristiques intéressantes. Les AG ont une capacité de s'adapter et de se transformer sans avoir à imposer beaucoup de restrictions sur la recherche d'une solution, ce qui leur permet de mieux explorer le problème et de trouver des solutions originales.

Le principe de l'algorithme est simple et repose sur un codage des problèmes et de leurs solutions sous la forme de chaînes d'éléments de base. Les chaîne peuvent être rompues entre chaque élément de base, à l'image des chromosomes, qui eux constituent de véritables listes de caractéristiques d'un individu. Le codage prend habituellement la forme d'une chaîne binaire très structurée, de longueur fixe ou variable selon le type de problème. On génère tout d'abord une *population* de solutions potentielles à un problème donné, sous la forme de telles chînes, puis on sélectionne, au moyen d'une mesure d'ajustement, les éléments de la population qui satisfont au mieux les contraintes (ou caractéristiques) de la solution recherchée. *Des opérateurs génétiques* sont ensuite appliqués à cette population de manière à obtenir une nouvelle population, possédant, dans son ensemble, de meilleures solutions que la précédente génération. On réitère le processus sur plusieurs générations, jusqu'à l'obtention d'une génération de solutions qui satisfait les critères de qualité imposés, et qui est beaucoup plus adapté au traitement du problème en question. Il suffit alors de choisir parmi la population la meilleure solution.

Les AG mettent en œuvre des mécanismes assez simples pour l'évolution vers la solution. Ils ne font que des copies et des échanges de morceaux de chaînes, en plus des opérations d'évaluation et de sélection des éléments d'une population. Les manipulations des chaînes sont faites en utilisant trois opérateurs basiques : la *reproduction*, le *croisement* et la *mutation*.

2.2.2.2 Induction de Règles à partir d'Exemples

L'utilisation des algorithmes génétiques dans les systèmes d'apprentissage automatique s'est largement développé à partir des années quatre-vingt. Plusieurs applications des AG à la construction de classifieurs ont été déjà expérimentées avec des bons résultats. La première application a été réalisée par celui qui est à l'origine même des AG, J. Holland [HOL 86, HOL 87]. Plusieurs autres expérimentations ont fait suite à ces premiers travaux [DAV 91, GOL 89, DEJ 88, DEJ 90, DEJ 93b, GRE 93, EIC 96].

Pour pouvoir réaliser un apprentissage de règles avec des AG, il est nécessaire d'avoir une représentation interne de l'espace de recherche. Le codage utilisé par les AG est donc très

important vis-à-vis de l'implémentation des outils d'apprentissage automatique. Le choix d'un type de codage spécifique doit essayer de garder les propriétés des AG, sans imposer des restrictions qui peuvent nuire ou restreindre la convergence de l'algorithme vers une bonne solution.

Deux approches assez connues pour générer automatiquement un classifieur à partir d'un AG sont : l'approche de Pitt et l'approche de Michigan [DEJ 93a]. L'approche de Pitt, développée initialement à l'Université de Pittsburgh, postule qu'un élément (chromosome) correspond à une base de règles et qu'une population de chromosomes est un ensemble de bases de règles. L'autre approche, celle de Michigan, à été développée à l'Université de Michigan par l'équipe de J. Holland. Cette approche postule qu'un chromosome représente une règle et qu'une population correspond donc à une base de règles. L'algorithme, au cours de ses itérations cherche à construire et améliorer une seule base de connaissances.

2.2.2.3 Discussion sur les Algorithmes Génétiques

Les algorithmes génétiques se présentent comme une approche très intéressante. Ils sont capables d'exploiter l'espace de recherche sans avoir à imposer de restrictions (*biais*) liées au type d'algorithme choisi, ce qui n'est pas toujours le cas des autres algorithmes d'apprentissage automatique. Par contre, les choix faits au niveau du codage employé, du type de règles manipulées, ou du type de méthodes de sélection et de transformation utilisées, vont beaucoup influencer l'apprentissage. Une des tâches les plus difficiles va être la définition de ces choix d'implémentation des AG. Ce type de problème est d'autant plus délicat qu'il n'existe pas de méthodologie générale à suivre.

La difficulté du choix d'un codage et d'une bonne fonction de sélection et adaptation font des AG une technique très particulière qui doit être bien maîtrisée pour donner des bons résultats. Dans le cas contraire, ces algorithmes n'arriveront pas à trouver de bonnes solutions aux problèmes posés. Le manque d'une garantie de convergence vers une bonne solution s'agit de l'un

des principaux désavantages des AG. De plus, les AG sont des algorithmes très lourds à exécuter, d'où l'intérêt d'exploiter le parallélisme dans ce type d'approche.

Comme d'autres algorithmes d'optimisation, ils connaissent les problèmes liés à la convergence et au blocage dans des *minima locaux*. Par contre, si les fonctions de mutation, de reproduction et de croisement sont bien définies, les AG peuvent constituer une solution assez intéressante pour s'échapper des minima locaux. Un autre avantage des AG est le fait qu'ils admettent l'introduction de connaissances théoriques préexistantes dans la population initiale à traiter. Donc, cette population initiale nous permet d'avoir un point de départ pour l'apprentissage qui peut être plus proche de la solution finale recherché.

Pour conclure cette discussion, il nous reste à souligner que la majorité des AG sont des approches qui manipulent des informations purement symboliques, d'où la difficulté à travailler avec des variables continues et des données approximatives. C'est le type de représentation des règles symboliques qui va définir le pouvoir d'un tel algorithme d'apprentissage. On trouve dans la littérature des travaux qui cherchent à surmonter ces limitations en utilisant des règles plus robustes, comme le système DELVAUX, qui utilise des règles bayésiennes [EIC 96].

2.2.3 Le Raisonnement Fondé sur des Cas - CBR

Le raisonnement fondé sur des cas (CBR) recouvre un ensemble de méthodes de résolution de problèmes qui exploite les expériences passées, plutôt que les connaissances générales d'un niveau supérieur, telles que les règles de production. Les CBR sont des méthodes d'apprentissage totalement fondées sur les connaissances empiriques, au lieu de faire usage des connaissances théoriques d'un domaine.

2.2.3.1 Principes de Base

Un système CBR est capable d'utiliser la connaissance spécifique contenue dans son expérience passée pour résoudre les nouveaux problèmes. Cet expérience est représentée normalement sous la forme de cas. Ces cas, qui ont été corrigés et assignés par l'expert aux classes auxquelles ils appartiennent, constituent ainsi la mémoire d'un système CBR.

Quand un nouveau problème est présenté à un système CBR, il va *se rappeler* des cas passés stockés dans son mémoire, similaires au problème courant. Ensuite, le système adapte la meilleure solution mémorisée et la transfère au problème actuel. Le cas nouveau, qui a été traité par le système et reconnu, peut être à son tour mémorisé et donc ajouté comme une nouvelle expérience du système [MAL 96, KOL 93].

En général, un système de raisonnement fondé sur des cas contient les phases suivantes :

1. *Remémoration* des cas les plus similaires par rapport au cas posé en question;
2. *Réutilisation* de la connaissance du(ou des) cas remémore(s) pour la résolution du problème;
3. *Révision* de la solution donné afin de la valider;
4. *Mémorisation* de cette nouvelle expérience, pour une utilisation future.

2.2.3.2 Discussion sur le Raisonnement Fondé sur des Cas

Les systèmes de raisonnement fondé sur des cas font appel à des algorithmes d'apprentissage dits *paresseux* (*lazy learning* [AHA 97]). Ils diffèrent des autres algorithmes par le fait qu'ils retardent le traitement des informations, c'est-à-dire qu'ils ont un faible coût calculatoire pendant la phase d'apprentissage et des calculs plus intenses pendant la phase de test (reconnaissance, *recall*). Les systèmes CBR n'essaient pas de traiter les exemples fournis, alors que les autres algorithmes d'apprentissage font une compilation des exemples d'apprentissage et les remplacent par des abstractions concises (e.g.: arbres de décision, ensembles de règles).

Les CBR sont des systèmes purement fondés sur les connaissances empiriques et ne permettent pas l'utilisation de connaissances théoriques. Donc, on ne peut pas profiter des connaissances disponibles sur le domaine d'application sauf si celles-ci sont représentées par des cas pratiques. Toutefois, on trouve dans la littérature des méthodes alternatives que permettent de contourner ce problème d'une manière assez simpliste : la solution consiste en transformer les règles symboliques en exemples que les représentent (prototypes des concepts théoriques) [ALT 97,ORS 95].

Une des particularités de ce type d'approche est le rôle essentiel joué par la fonction de similarité employée. Elle va déterminer le bon fonctionnement du système, car il est nécessaire de bien retrouver les bons exemples proches de l'exemple traité. La fonction de remémoration la plus simple et une des plus utilisées est la méthode des K-Plus-Poches-Voisins (*K-PPV* ou *K-Nearest-Neighbor*), avec une mesure de similarité fondée sur la distance euclidienne. Ce type de mesure peut bien fonctionner avec des variables quantitatives, mais est mal adapté quand il s'agit de mesurer la distance entre deux valeurs d'une variable qualitative (e.g. peut-on mesurer la distance entre 'lundi' et 'jeudi', ou entre 'bleu' et 'rouge' ?).

L'une des avantages d'une telle approche est le traitement assez simple des cas particuliers et des exceptions, ce qui n'est pas toujours évident pour un système d'apprentissage inductif. Enfin, ce type de système n'est pas très performant en ce qui concerne l'explication de la solution trouvée puisqu'au mieux il va donner la liste des cas les plus semblables au cas présenté sans faire une analyse plus profonde de la solution fournie.

2.3 APPRENTISSAGE AUTOMATIQUE - APPROCHES CONNEXIONNISTES

Les réseaux connexionnistes sont des assemblages fortement connectés d'unités de calcul, les *neurones formels*. Ces derniers ont pour origine un modèle du *neurone biologique*, dont ils ne retiennent d'ailleurs qu'une vision fort simplifiée (voir Figure 8 et Figure 9). Le neurone, comme

toute cellule, est composé d'un corps (ou *soma*), qui contient son noyau et où se déroulent les activités propres à sa vie cellulaire. Cependant, il est aussi doté d'un *axone* et de *dendrites*, structures spécialisées dans la communication avec les autres neurones. Cette communication entre cellules nerveuses s'effectue via des impulsions nerveuses. Les impulsions sont générées à l'extrémité somatique de l'axone et vont vers les terminaisons axonales. Là, elles affecteront tous les neurones reliés au neurone générateur, par l'intermédiaire de jonctions entre les terminaisons axonales et les autres cellules. Cette jonction est appelée *synapse*.

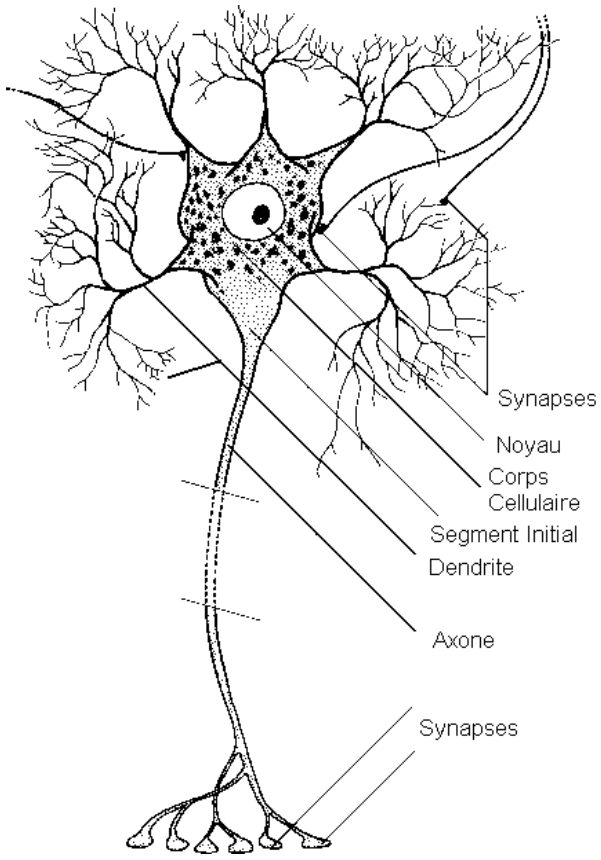


Figure 8- Exemple de Neurone Biologique

Cet héritage de la neurobiologie forme une composante importante de l'étude des réseaux connexionnistes, et le souci de maintenir une certaine correspondance avec le système nerveux humain a animé et continue à animer une part importante des recherches dans ce domaine. Malgré cet héritage, l'essentiel des travaux d'aujourd'hui ont pour objet les réseaux de neurones formels et

non son corrélât neurobiologique. Vu comme des systèmes de calcul, les réseaux de neurones possèdent plusieurs propriétés qui les rendent intéressants d'un point de vue théorique, et fort utiles en pratique. C'est cette approche - plus technique que biologique - qui est adoptée pour nous dans cette thèse.

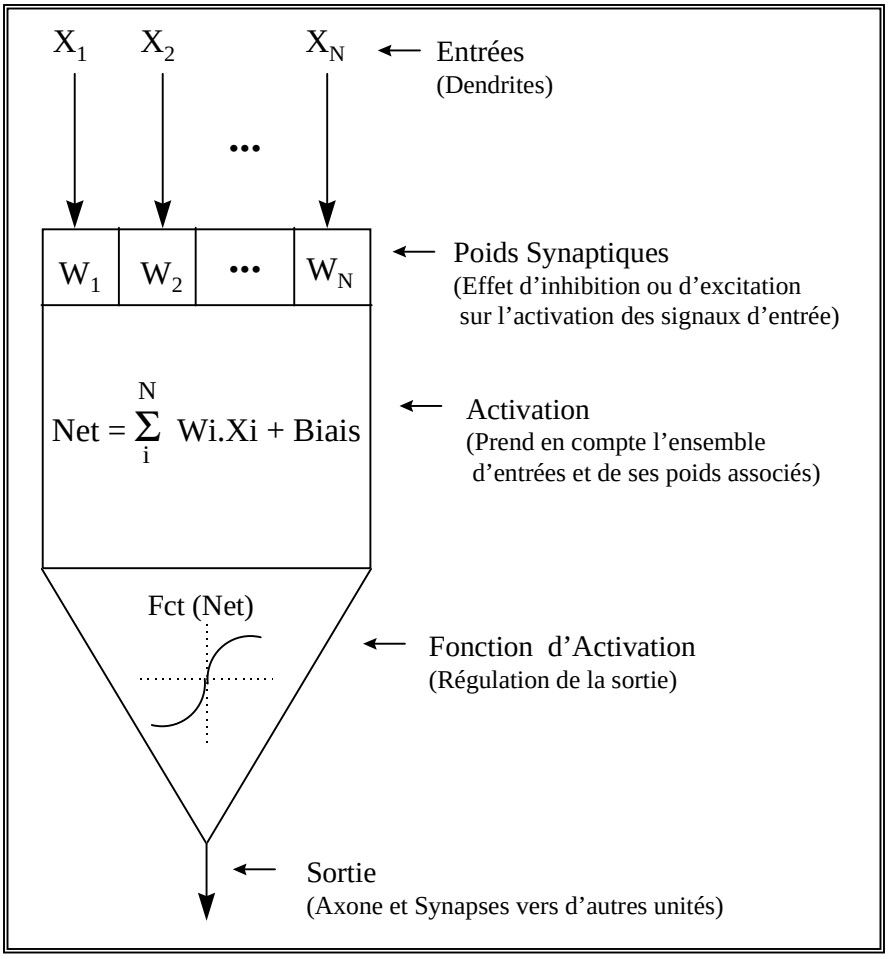


Figure 9- Exemple de Neurone Formel

Un réseau connexionniste est constitué par un graphe orienté et pondéré. Les noeuds de ce graphe sont des automates simples nommés *neurones formels* ou tout simplement *unités du réseau*. Les unités sont dotées d'un *état interne*, que l'on appelle *état d'activation*. Les unités peuvent propager son état d'activation aux autres unités du graphe en passant par des arcs pondérés appelés *connexions*, *liens synaptiques* ou bien *poids synaptiques*. La règle qui détermine l'activation d'un neurone en fonction de l'influence venue de ses entrées et de leurs poids

respectifs s'appelle *règle d'activation* ou *fonction d'activation*. Les *changements* apportés aux valeurs *des poids synaptiques* ou dans la structure d'interconnexion des unités du réseau sont responsables des changements de comportement d'activation de ce réseau, ce qui lui permet d'*apprendre* un nouveau comportement. Ainsi, le réseau est capable d'établir des associations entrée-sortie (stimulus et réponse) afin de bien résoudre un problème. La méthode utilisée pour modifier le comportement d'un réseau s'appelle règle d'apprentissage.

La grande quantité de modèles de réseaux connexionnistes rend difficile leur descritpion exhaustive. Le lecteur pourra se référer aux nombreux ouvrages sur le sujet pour une description générale des réseaux de neurones. En particulier, l'ouvrage édité par E. Fiesler et R. Beale, intitulé "Handbook of Neural Computation" [FIE 97] complétera les différents points abordés ici (voir aussi les références citées dans la section 2.1.2.1). Nous allons plutôt approfondir ci-dessous les différences entre les modèles, afin de mieux comprendre les choix qui ont été faits dans cette thèse.

2.3.1 Classification et Propriétés

La grande quantité de modèles connexionnistes existants [SAR 97, JOD 94b] nous oblige à tout d'abord les classer selon leurs différentes proprietés, afin de mieux comprendre les avantages et les inconvénients du choix d'un modèle plutôt que l'autre. Il n'existe pas une seule façon de classer les modèles existants, mais plusieurs, selon les attributs choisis, tels que : le type d'apprentissage, l'architecture d'interconnexion des unités du réseau, la forme pour traiter et représenter les données, etc. Une description plus detaillée des réseaux connexionnistes et de leur classification est donnée par E. Fiesler [FIE 97].

2.3.1.1 Apprentissage Connexionniste

L'apprentissage est en général un processus graduel, itératif, où les poids du réseau sont modifiés plusieurs fois selon une règle d'apprentissage avant d'atteindre leurs valeurs finales. Les

approches d'apprentissage connexionniste peuvent être reparties en trois grandes classes, selon le degré de contrôle donné à l'utilisateur :

- *Apprentissage Supervisé* : l'utilisateur dispose d'un comportement de référence précis qu'il désire faire apprendre au réseau. Le réseau est donc capable de mesurer la différence entre son comportement actuel et le comportement de référence, et de corriger ses poids de façon à réduire cette erreur. L'apprentissage supervisé utilise des connaissances empiriques, habituellement représentées par des ensembles d'exemples étiquetés par la classe à laquelle ils appartiennent.

- *Apprentissage Semi-Supervisé* : l'utilisateur ne possède que des indications imprécises (par exemple, échec/succès du réseau) sur le comportement final désiré. Les techniques d'apprentissage semi-supervisé sont aussi appelées apprentissage par renforcement (*reinforcement learning*). En effet, on dispose souvent tout au plus d'une évaluation qualitative du comportement du système.

- *Apprentissage non-supervisé* : les poids du réseau sont modifiés en fonction de critères internes comme la coactivation des neurones. Les comportements résultants de ces apprentissages sont en général comparables à des techniques d'analyse de données. On les appelle aussi *auto-organisation.*

L'apprentissage connexionniste nécessite en général une grande quantité de données, que l'on regroupe dans un *ensemble d'exemples d'apprentissage*. Selon la technique d'apprentissage utilisée, d'autres ensembles de données sont aussi employés, notamment pour mesurer la validité de la solution trouvée par le réseau. On appelle ces données supplémentaires de *ensembles d'exemples de test* ou de *généralisation*. On appelle généralisation la capacité du réseau à réagir correctement lorsqu'on lui présente des entrées qui n'ont pas été vues lors de l'apprentissage. Dans la pratique, ce sont les *capacités de généralisation* d'un réseau connexionniste qui vont établir les possibilités de son application à différents problèmes.

Un réseau peut trop se spécialiser à un ensemble d'apprentissage. Ce type de comportement d'apprentissage va probablement mener à un problème de mauvaise généralisation, ou surapprentissage (*overfitting*). Une façon d'éviter le surapprentissage est l'utilisation de la

procédure de la validation croisée (*cross-validation*). Cette procédure consiste en diviser les données en deux ensembles distincts : un ensemble d'apprentissage et un ensemble de test. L'ensemble de test nous permet de contrôler l'erreur de généralisation ; ainsi, on peut déterminer le bon moment pour arrêter le processus d'apprentissage. La Figure 10 montre deux courbes où sont représentées les erreurs de sortie d'un réseau par rapport aux ensembles d'apprentissage et de test.

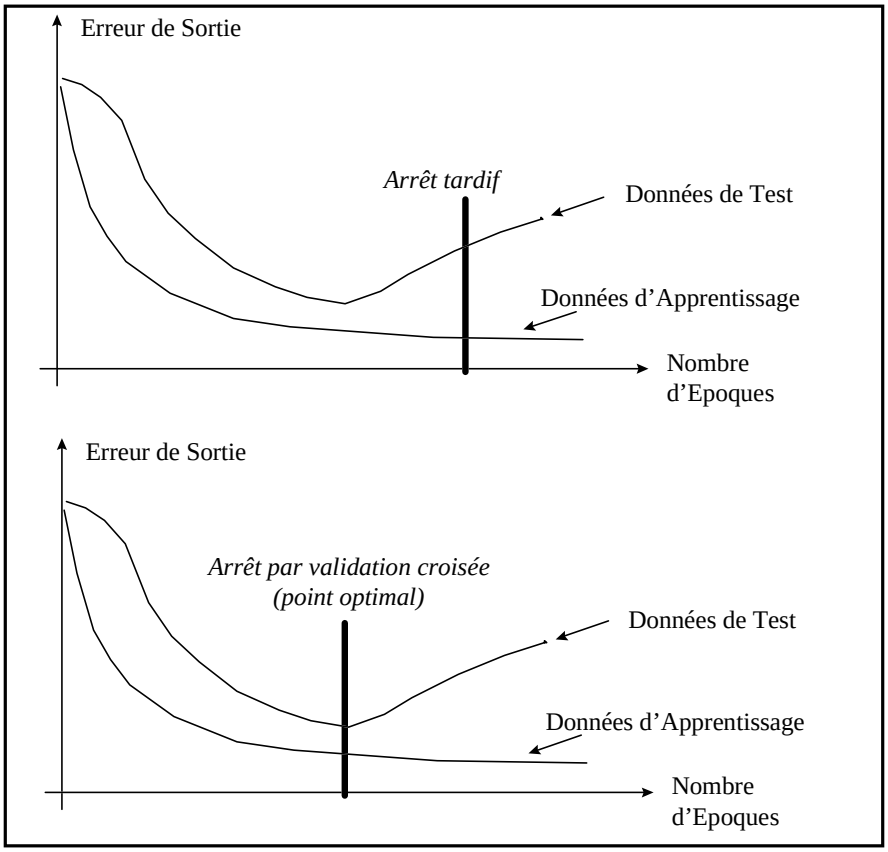


Figure 10- Apprentissage : Erreur dans l'ensemble d'apprentissage et de test

L'apprentissage d'un ensemble de données peut être réalisé de différentes façons, selon la manière dont le réseau est alimenté par les données :

• *Apprentissage Instantané* : l'ensemble de données d'apprentissage est analysé une seule fois et permet d'établir les poids du réseau d'une seul coup. Ce mode d'apprentissage requiert un

seul passage sur les données et il est aussi appelé *one-shot-learning*. Ce n'est pas une méthode très utilisée.

• *Apprentissage par Paquets* : l'ensemble des données d'apprentissage est présenté au réseau plusieurs fois, de façon à optimiser les poids du réseau et minimiser l'erreur en sortie. Chaque présentation de l'ensemble complet de données d'apprentissage est appelée une époque (*epoch*). L'algorithme d'apprentissage réduit petit à petit l'erreur de sortie à chaque présentation des données, et l'état du réseau doit converger vers la solution du problème. On peut aussi manipuler l'ordre des exemples de l'ensemble d'apprentissage, ce qui peut avoir des conséquences sur l'évolution de l'apprentissage (cf. l'apprentissage actif). Cette méthode est connue aussi comme *batch-learning* et constitue l'un des types d'apprentissage les plus utilisés.

• *Apprentissage Continu* : l'algorithme d'apprentissage prend en compte continuellement les exemples qui lui arrivent (*continuous/on-line learning*). Cette méthode est aussi appelée apprentissage incrémental (par rapport aux données), mais nous allons utiliser plutôt le terme apprentissage continu pour éviter toute confusion avec l'apprentissage incrémental dans le cas où des unités sont ajoutées à la structure du réseau. Un des principaux problèmes de l'apprentissage continu est la difficulté de trouver un bon compromis entre adaptation et stabilité, ce qui peut amener le réseau à 'oublier' les données apprises (trop d'adaptation) ou bien être incapable de s'adapter aux nouvelles données qui arrivent (trop de stabilité).

• *Apprentissage Actif* : cette approche est basée sur l'idée que l'algorithme d'apprentissage peut lui aussi imposer des choix par rapport aux données à utiliser. L'apprentissage passe d'un comportement passif, où le réseau ne peut pas intervenir sur les données qui lui arrivent, à un comportement actif, où le réseau peut déterminer quelles données vont être prises en compte et aussi dans quel ordre elles doivent être considérées. Il s'agit d'une voie de recherche relativement récente.

L'apprentissage peut être aussi implementé par différentes méthodes, selon le type de *règle d'apprentissage* sélectionné. Les règles d'apprentissage les plus utilisées sont [JOD 94b, SIM 90, CAU 92] :

- Les méthodes de *correction d'erreur*, telles que la descente de gradient sur une surface d'erreur. Exemples : Adaline, Perceptron, Rétro-propagation, Cascade-Correlation;

- Les méthodes d'apprentissage par *renforcement*. Exemples : AHC, ARC;
- Les méthodes d'apprentissage par *compétition* ou par *auto-organisation*. Exemples : Kohonen Feature Map, ART1;
- Les méthodes d'apprentissage par création de *prototypes* ou de *noyaux*. Exemples : RBF, ART1, ARN2;
- Les méthodes d'apprentissage basées sur des *mémoires associatives* (auto-associatives ou hétéro-associatives). Exemples : Modèle de Hopfield, BAM;
- Les méthodes *d'apprentissage temporel* (réseaux récurrents). Exemples : SRN, BPTT, RTRL.

Il existe des méthodes qui peuvent appartenir à plus d'une classe au même temps. Par exemple, les réseaux avec apprentissage du type ARN2 incluent dans cette approche des techniques d'apprentissage non-supervisé, supervisé, par compétition et aussi par création de prototypes. Dans cette thèse, nous allons nous focaliser plutôt sur les méthodes d'apprentissage supervisé fondées sur la descente du gradient. Le système que nous avons développé dans le cadre de cette thèse utilise un algorithme dérivé de la Rétro-Propagation - le Cascade-Correlation (CasCor) [FAH 90]. Nous allons étudier plus en détails les algorithmes d'apprentissage de type Rétro-Propagation dans la section 2.3.2.

2.3.1.2 Type des Unités

Les unités d'un réseau (neurones formels) peuvent être de différents types, selon la fonction interne utilisée pour calculer l'activation. Les principales différences résident dans le type de fonction d'activation employé (e.g.: linéaire, sigmoïde exponentielle - asymétrique, sigmoïde tangentielle - symétrique, gaussienne) [JOD 94a, JOD 94b], et dans la fonctionnalité des unités (unités à prototypes, unités du type Perceptron). Selon la fonctionnalité des unités nous pouvons avoir deux types majeurs de réseaux :

• *Réseaux à Prototypes* : ce type de réseau utilise des unités qui servent à représenter des prototypes d'exemples appris : les unités ont une représentation interne regroupant les caractéristiques typiques d'un ensemble d'exemples [CHE 97, MAL 96b, ORS 95, GIA 92]. Les réseaux à prototypes ont normalement une apprentissage non-supervisé, mais ils peuvent aussi avoir un apprentissage supervisé (avec un ou plusieurs prototypes associés à chaque classe). Une des avantages de ce type de réseau est la possibilité de faire un apprentissage continu (incrémental), puisqu'il n'est pas très difficile de concevoir des méthodes qui permettent d'augmenter le réseau par ajout de nouveaux prototypes. Les prototypes sont aussi appelés des noyaux.

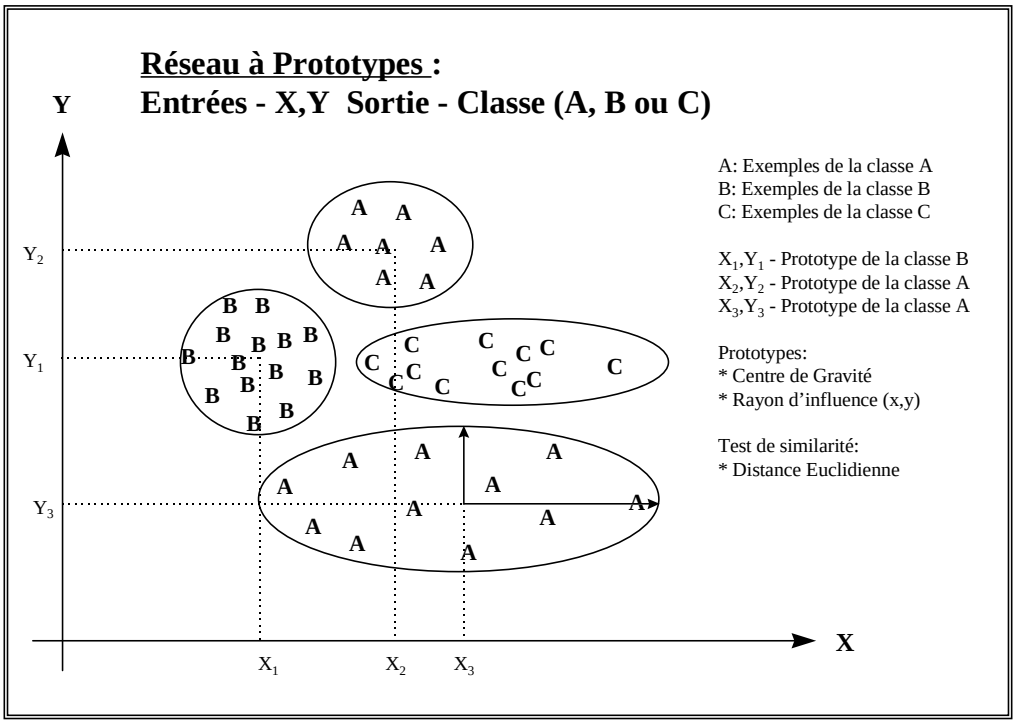


Figure 11 - Prototypes d'un réseaux à deux entrées

•*Réseaux de type "Perceptron"* : les unités du type "Perceptron" ont été proposées par Frank Rosenblatt en 1958. Il s'agit d'un des modèles d'unités les plus utilisés actuellement. Il est à l'origine de plusieurs autres réseaux de neurones avec apprentissage supervisé par correction d'erreur. Le modèle du Perceptron Multi-Couche (PMC) est devenu très connu, tout en étant

associé à la règle d'apprentissage de la Rétro-Propagation [JOD 94b, WID 90, RUM 86b]. Nous allons étudier ce type de réseau dans la section 2.3.2.

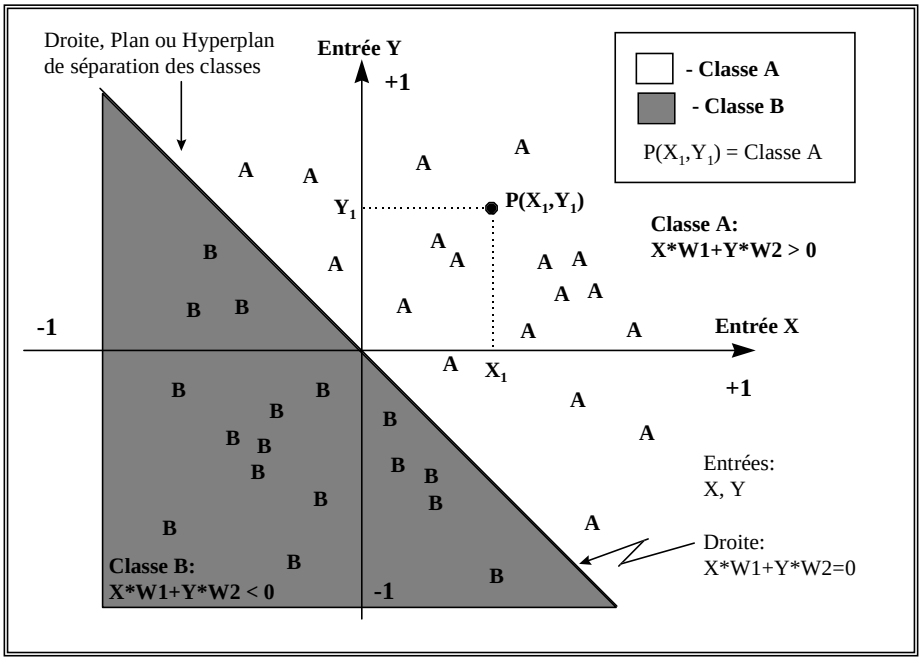


Figure 12 - Séparation des classes par un Perceptron à deux entrées

2.3.1.3 Type d'Architecture du Réseau

La façon d'interconnecter les unités d'un réseau va nous permettre d'obtenir différentes architectures. La Figure 13 montre des exemples de différentes façons d'interconnecter les unités. Les architectures les plus importantes sont :

- *Réseaux à une seule couche* : les unités sont toutes sur le même niveau. Dans ce type d'architecture, les unités sont connectées directement aux entrées et sont aussi les sorties du réseau. Les réseaux à une seule couche ont normalement des connexions latérales (entre les neurones d'une même couche). Un exemple de ce type d'architecture sont les réseaux du type "Kohonen Feature Map".

• *Réseaux à couches unidirectionnels* : les unités sont organisées en plusieurs niveaux bien définis, que l'on appelle des couches. Chaque unité d'une couche reçoit ses entrées à partir de la couche précédente et envoie ses sorties vers la couche suivante (*feed-forward nets*). La Figure 13(a) montre un exemple de réseaux à trois couches. Cette architecture à trois couches (entrée, couche cachée et sortie) est très utilisée dans la pratique. Le modèle du PMC [JOD 94b, WID 90] se compose en général d'une architecture de ce type, c'est-à-dire avec une seule couche cachée (*hidden layer*), mais rien n'empêche d'avoir plus d'une seule couche cachée dans ce modèle. Un autre type d'interconnexions dans les réseaux à couches sont les raccourcis (*short-cuts*) qui permettent de lier des unités en passant à travers des niveaux. Ainsi, on peut 'sauter' d'une couche à l'autre avec les raccourcis, à condition de ne pas créer une boucle (voir Figure 13(b)).

• *Réseaux récurrents* : les réseaux récurrents [SZI 95, JOD 94b, LAB 93] peuvent avoir une ou plusieurs couches, mais leur particularité est la présence d'interconnexions depuis la sortie d'une unité vers une autre unité de la même couche ou d'une couche inférieure. Ce type d'interconnexions permet de modéliser des aspects temporaux et des comportements dynamiques, où la sortie d'une unité dépend de son état antérieur. Les boucles internes rendent ce type de réseaux instable, ce qui nous oblige à utiliser des algorithmes plus spécifiques (et usuallement plus complexes) pour l'apprentissage. Un type particulier de réseau récurrent sont les réseaux totalement connectés, tels que le modèle de Hopfield représenté dans la Figure 13(d).

•*Réseaux d'ordre supérieur* : les unités de ce type permettent la connexion directe entre deux entrées ou plus, avant d'appliquer la fonction de calcul de l'activation de l'unité [FIE 94a]. Ce type de réseau sert à modéliser les synapses de modulation, c'est-à-dire quand une entrée peut moduler (agir sur) le signal provenant d'une autre entrée. Un modèle particulier de réseaux d'ordre supérieur (*high-order neural net*) est le réseau *Sigma-Pi* proposé par Rumelhart [RUM 86a], et représenté dans la Figure 13(e).

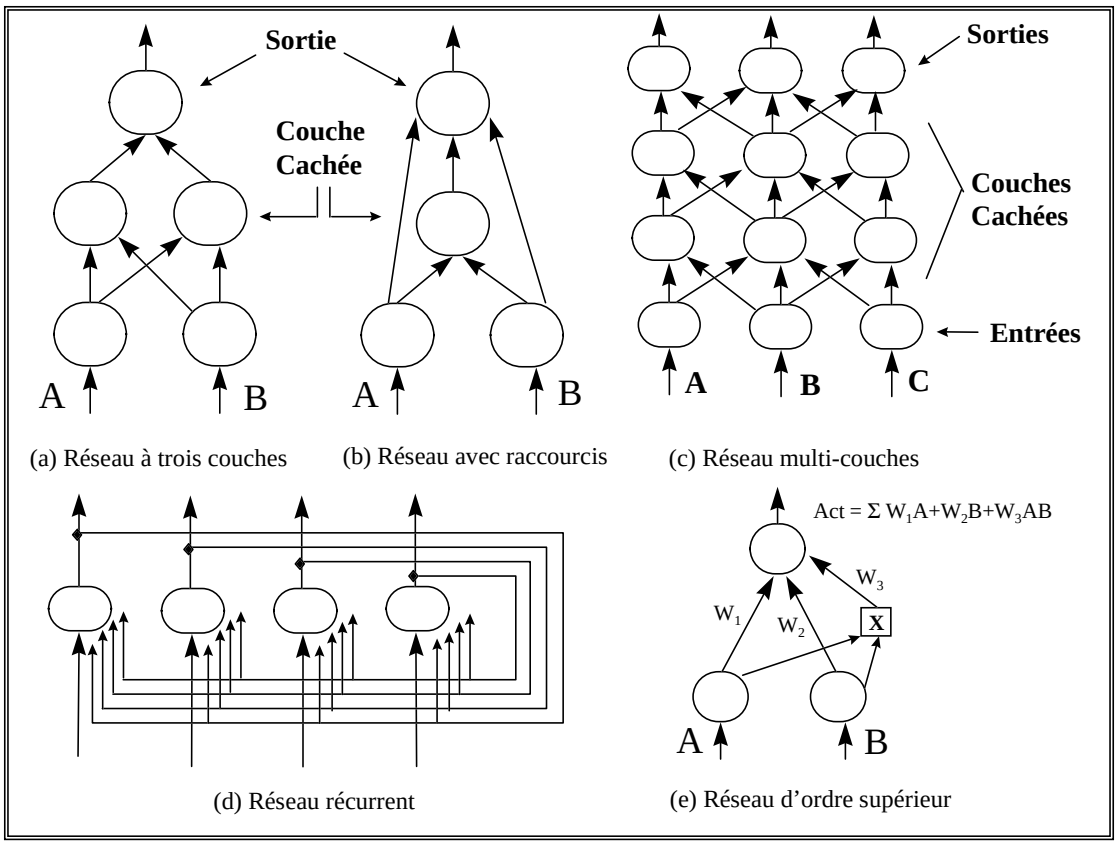


Figure 13 - Architectures d'interconnexion des unités d'un réseau

L'architecture d'un réseau peut être aussi classée selon son évolution au cours de l'apprentissage et de son utilisation. En fonction de ce critère, les réseaux sont divisés dans les groupes suivants :

• *Structure statique* : le réseau a sa structure définie avant l'apprentissage. La quantité de neurones ainsi que sa structure d'interconnexions ne changent pas. Les seuls changements subis par le réseau se situent au niveau des poids synaptiques, modifiés au cours de l'apprentissage. Ce type de modèle pose une difficulté : la détermination du bon nombre de neurones et interconnexions à utiliser. Un réseau avec peu d'unités et d'interconnexions risque de ne pas pouvoir apprendre à résoudre correctement un problème, et un réseau avec trop d'unités peut avoir des problèmes de convergence, de complexité et de généralisation [JOD 94a, FLE 94]. Il n'existe pas de méthode optimale qui permet de spécifier automatiquement le nombre exact

d'unités à employer pour bien résoudre un problème donné. Les réseaux PMC à Rétro-Propagation classiques sont des réseaux statiques.

• *Structure dynamique* : les réseaux ayant une structure dynamique sont des réseaux dont le nombre d'unités et d'interconnexions peut varier. Ils sont aussi appelés réseaux ontogéniques [FIE 94b]. Les changements dans la structure du réseau peuvent être de type *génératif* (*incrémental*) ou de type *destructif* (*élagage*). Le choix entre ces deux méthodes est controversé : faut-il commencer petit et agrandir ou faut-il commencer grand et réduire ensuite ? [ELM 93]. D'un point de vue calculatoire, commencer petit et ajouter des unités et des interconnexions au fur et à mesure de l'apprentissage est plus performant que faire le travail d'apprentissage sur un grand réseau pour en détruire certaines parties ensuite. Une seule chose semble être acceptée par la plupart des chercheurs : les réseaux ontogéniques sont une des meilleures méthodes dont on dispose pour choisir une bonne architecture pour un réseau et pour bien résoudre un problème donné. Les réseaux CasCor, que nous avons utilisé dans cette thèse, sont des réseaux dynamiques du type incrémental.

Le dernier point relevant des architectures concerne la *modularité des réseaux* [AMY 96]. Les réseaux neuronaux peuvent avoir des architectures modulaires : ils peuvent être constitués par des blocs plus ou moins dépendants les uns des autres. On peut avoir des différentes approches pour intégrer et faire coopérer les modules d'un réseau. Une première approche consiste à décomposer le problème et obtenir ainsi des modules spécialisés pour chaque sous-problème. Dans cette approche, si l'on prend une tache de classification en plusieurs classes, on peut considérer que l'identification de chaque classe va pouvoir être traitée par des modules séparés. L'autre approche, plus difficile à mettre en place, est celle où les différents modules vont coopérer pour aboutir à une solution. Dans cette approche, on ne va pas imposer de tâches particulières à des modules pré-spécifiés. La modularité est un problème relatif au choix de l'architecture d'un réseau, mais peut être aussi lié au choix du partitionnement des données. Finalement, la modularité peut devenir un aspect très important à prendre en compte selon la complexité du problème abordé. Un étude plus profonde sur ce sujet a été développée par S. Rouzier [ROU 98].

2.3.1.4 Type d'Application du Réseau

Les réseaux peuvent être appliqués à plusieurs types de tâches, telles que : la reconnaissance de formes (e.g. reconnaissance de visages), la classification, la transformation de données (e.g. compression), la prédiction (e.g. prédiction de séries temporelles), le contrôle de processus et l'approximation de fonctions. Toutes ces tâches peuvent être regroupés en deux groupes principaux selon le type des sorties fournies par le réseau et le comportement qui est recherché. Ces deux groupes principaux d'applications sont :

- *Réseaux pour l'Approximation de Fonctions* : ce type de réseau doit avoir une sortie continue et sera employé pour l'approximation exacte (*interpolation*) ou pour l'approximation approchée d'une fonction représentée par les données d'apprentissage [CHE 97]. Ce type de réseau est capable d'apprendre une fonction de transformation (ou d'association) des valeurs d'entrée vers les valeurs de sortie. Cette fonction acquise par le réseau permet de prédire les sorties étant données les entrées. On appelle ce type de problème, un problème de *régression* [BIS 97]. En général, les fonctions représentées sont des fonctions avec des variables d'entrée et de sortie continues.

- *Réseaux pour la Classification* : ce type de réseau doit attribuer des classes (valeur de sortie non-continue) aux exemples qui lui sont fournis. La classification est un cas particulier de l'approximation de fonctions où la valeur de sortie est discrète et appartient à un ensemble limité de classes. Cet ensemble de classes peut être connu d'avance dans le cas de l'apprentissage supervisé. Un réseau adapté à la classification doit avoir des sorties discrètes ou implementer des méthodes de discrétisation des sorties (e.g. application d'un seuil de discrimination).

Il est un peu délicat de classer précisément tous les différents modèles de réseaux de neurones dans une seule de ces deux classes. La plupart des modèles peuvent être adaptés pour être utilisés dans l'un ou l'autre type d'application. Mais certains modèles sont plus adaptés à une tache de classification que d'approximation et vice versa.

2.3.2 Le modèle du Perceptron Multi-Couches

Nous allons faire ici une brève description des réseaux de type Perceptrons Multi-Couches (PMC) et des processus d'activation et d'apprentissage couramment utilisés avec ce type de modèle. Les réseaux PMC sont constitués par des unités organisées en couches : une couche d'entrée, une ou plusieurs couches cachées, et une couche de sortie (voir Figure 13 (a,b,c)). Les unités de la couche d'entrée reçoivent les signaux provenants de l'environnement. Ces signaux représentent les informations traitées par le réseau, telles que : la description de l'état du système, les valeurs des attributs associés aux variables d'entrée, les mesures obtenues à partir de capteurs externes ou les faits qui composent les prémisses d'un système expert. La couche de sortie rend disponible le résultat de l'activation du réseau à l'environnement extérieur. Enfin, les unités cachées, comme leur nom l'indique, n'ont pas d'interaction directe avec l'environnement.

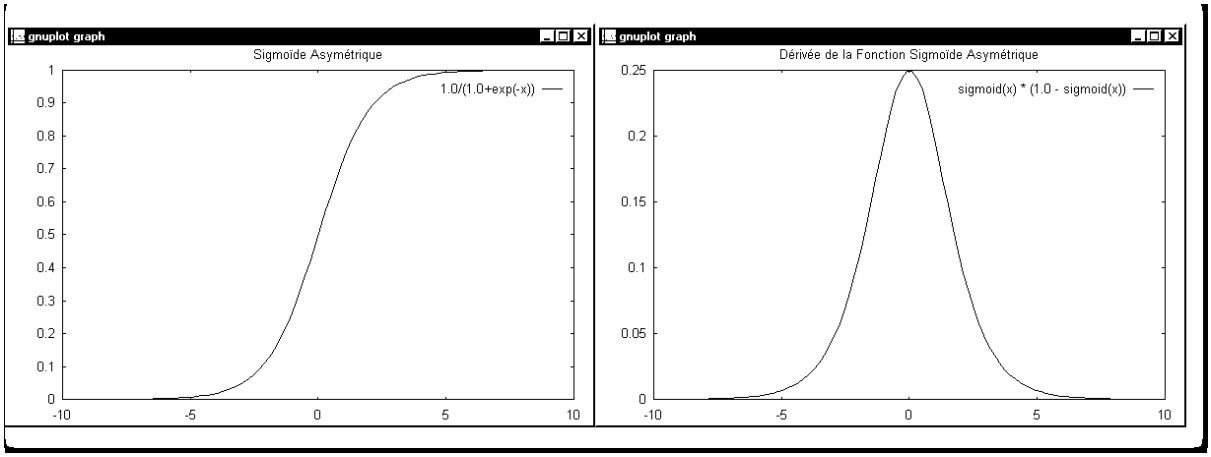


Figure 14- Sigmoïde Asymétrique

Les unités sont connectées selon une architecture sans boucles (connexions unidirectionnelles), mais on peut avoir des connexions inter-couches du type "raccourcis". Les réseaux PMC travaillent sur deux modes opératoires distincts : en mode de rappel (activation du réseau/consultation) et en mode d'apprentissage (adaptation). L'activation des réseaux PMC est réalisée par la propagation des signaux à partir des unités d'entrée vers les sorties. Ces unités reçoivent en entrée des signaux provenant soit des entrées externes soit des sorties des autres unités du réseau auxquelles elles sont connectées. Chaque unité génère une valeur de sortie qui

est envoyée vers les unités auxquelles elle est connecté ou vers la sortie externe. La sortie d'une unité du réseau est obtenue à partir des équations suivantes :

Somme pondérée : $Si = \sum_{j=1}^{N} Wij.Xj + \Theta i$ (Eq. 1)

Fonction d'activation (sigmoïde asymétrique) : $Fa(x) = \frac{1}{1+e^{-x}}$ (Eq. 2)

Fonction d'activation (tangente hyperbolique symétrique) : $Fa(x) = \tanh(x)$ (Eq. 3)

En utilisant la fonction d'activation du type sigmoïde asymétrique on obtient :

Activation de l'unité : $Ai = Fa(Si) = \frac{1}{1+e^{-(\sum Wij.Xj + \Theta i)}}$ (Eq. 4)

avec :

Xj : Valeur de l'entrée 'j' de l'unité en question. Cette valeur peut être originaire d'une entrée du réseau ou de la sortie d'une autre unité.

Wij : Poids associé à l'entrée 'j' de l'unité 'i'.

Θi : Seuil de l'unité 'i'. Il peut être considéré comme un poids synaptique associé à une entrée toujours égale à 1. (biais).

N : Nombre d'entrées de l'unité en question.

Si : Valeur cumulée des entrées pondérées par les poids synaptiques de l'unité 'i'.

Ai : Valeur obtenue à la sortie de l'unité 'i' après activation.

Fa(x) : Fonction d'activation, en général la sigmoïde (voir Figure 14).

Les réseaux PMC utilisent une méthode d'apprentissage de type supervisé. L'apprentissage est réalisé par un algorithme de minimisation sur l'erreur estimé à partir des sorties du réseau. Cet algorithme réalise une descente du gradient de la surface d'erreur (voir Figure 15). L'emploi de

cette méthode permet d'entraîner les unités de sortie du réseau, où l'erreur peut être obtenu directement par simple différence entre la valeur de sortie désirée et la valeur obtenue par le réseau ; mais il permet aussi d'entraîner les unités cachées grâce à une technique de propagation en arrière de l'erreur à travers les couches du réseau, d'où le nom de Rétro-Propagation" (*Back-Propagation Learning*).

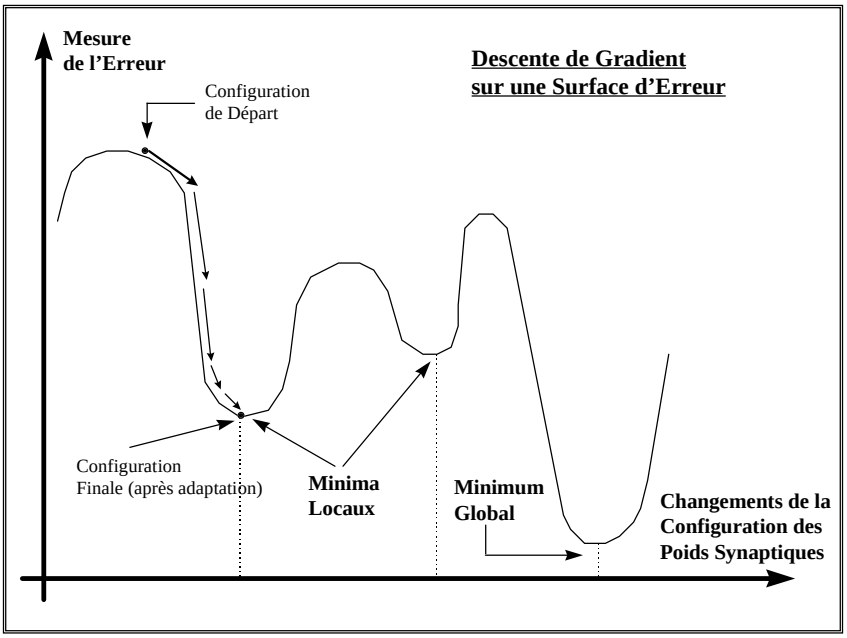


Figure 15 - Descente de gradient sur une surface d'erreur et minima locaux

Les Perceptrons ont été créés par Rosenblatt [ROS 59] il y a longtemps, mais les réseaux de ce type sont restés limités à des architectures avec deux couches seulement. Pourtant, certains problèmes ne pouvaient pas être traités par un simple réseau à deux couches (comme l'apprentissage de la fonction logique "Ou Exclusif" - XOR). C'est seulement à partir des années 80 que des règles d'apprentissage pour les réseaux à plus de deux couches ont été proposées. L'algorithme d'apprentissage de la Rétro-Propagation de l'erreur [RUM 86, PAR 85, LEC 88] est la méthode qui a permis de résoudre le problème de l'attribution du blâme (*credit assignment*) concernant la répartition de l'erreur observée parmi les unités de la couché cachée des réseaux PMC. Cette méthode est en effet assez proche de la règle delta de Widrow et Hoff [WID 90]. Elle sert à minimiser l'erreur quadratique entre la sorties du réseau et la sortie désirée, et adapter les poids des unités cachées du réseau.

L'algorithme d'apprentissage par minimisation de l'erreur quadratique (descente du gradient) est dérivé des formules qui suivent :

Erreur quadratique : $E = \frac{1}{2}\sum_i (Di - Ai)^2$ (Eq. 5)

Adaptation des poids synaptiques par descente du gradient : $\Delta Wij = -\alpha \frac{\partial E}{\partial Wij}$ (Eq. 6)

Calcul du gradient de l'erreur par rapport au poids synaptique :

Unité de sortie avec une sortie linéaire (Ai = Si)

$\frac{\partial E}{\partial Wij} = \frac{\partial E}{\partial Ai}\frac{\partial Ai}{\partial Wij} = \delta i \, \text{Xj}$ (Eq. 7)

$\frac{\partial E}{\partial Ai} = -(\text{Di} - \text{Ai}) = \delta i$ (Eq. 8)

$\frac{\partial Ai}{\partial Wij} = \text{Xj}$ (Eq. 9)

$\Delta Wij = -\alpha . \delta i . Xj = \alpha .(\text{Di} - \text{Ai}).\text{Xj}$ (Eq. 10)

Unité de sortie avec une fonction d'activation non-linéaire (Ai = Fa(Si))

$\frac{\partial E}{\partial Wij} = \frac{\partial E}{\partial Si}\frac{\partial Si}{\partial Wij} = \delta i \, \text{Xj}$ (Eq. 11)

$\frac{\partial E}{\partial Si} = \frac{\partial E}{\partial Ai}\frac{\partial Ai}{\partial Si} = -(\text{Di} - \text{Ai}).\text{Fa}'(\text{Si}) = \delta i$ (Eq. 12)

$\frac{\partial E}{\partial Ai} = -(\text{Di} - \text{Ai})$ (Eq. 13)

$$\frac{\partial Ai}{\partial Si} = \text{Fa'(Si)} \qquad \text{(Eq. 14)}$$

$$\frac{\partial Si}{\partial Wij} = \text{Xj} \qquad \text{(Eq. 15)}$$

$$Fa(x) = \frac{1}{1+e^{-x}} \qquad \therefore \qquad \text{Fa'(x) = Fa(x).(1 - Fa(x))}$$

$$\text{Fa'(Si) = Fa(Si).(1 - Fa(Si)) = Ai.(1 - Ai)} \qquad \text{(Eq. 16)}$$

$$\text{Fa(x) = tanh (x)} \qquad \therefore \qquad \text{Fa'(x) = (1 - Fa(x).Fa(x))}$$

$$\text{Fa'(Si) = (1 - Fa(Si).Fa(Si)) = (1 - Ai.Ai)} \qquad \text{(Eq. 17)}$$

$$\Delta Wij = -\alpha.\delta i.Xj = \alpha.\text{(Di - Ai).Fa'(Si).Xj} \qquad \text{(Eq. 18)}$$

Unité cachée avec une fonction d'activation non-linéaire (Ai = Fa(Si))

$$\frac{\partial E}{\partial Wij} = \frac{\partial E}{\partial Si}\frac{\partial Si}{\partial Wij} = \delta i\,\text{Xj} \qquad \text{(Eq. 19)}$$

$$\frac{\partial E}{\partial Si} = \frac{\partial E}{\partial Ai}\frac{\partial Ai}{\partial Si} = \text{- Fa'(Si).}\sum_k \delta kWki = \delta i \qquad \text{(Eq. 20)}$$

$$\frac{\partial E}{\partial Ai} = \sum_k \frac{\partial E}{\partial Sk}\frac{\partial Sk}{\partial Ai} = \sum_k \frac{\partial E}{\partial Sk}\frac{\partial}{\partial Ai}\sum_h WhkAh = \ldots$$

$$\ldots = \sum_k \frac{\partial E}{\partial Sk}Wki = -\sum_k \delta kWki \qquad \text{(Eq. 21)}$$

$$\frac{\partial Ai}{\partial Si} = \text{Fa'(Si)} \qquad \text{(Eq. 22)}$$

$$\frac{\partial Si}{\partial Wij} = \text{Xj} \qquad \text{(Eq. 23)}$$

$$\Delta Wij = -\alpha.\delta i.Xj = \alpha.\text{Xj.Fa'(Si).}\sum_k \delta kWki \qquad \text{(Eq. 24)}$$

Les notations utilisées sont les suivantes :

Fa'(x) : Dérivée de la fonction d'activation. Nous avons présenté deux exemples de fonctions d'activation Fa(x) et ses dérivées : la sigmoïde asymétrique et la sigmoïde symétrique (tanh).

Di : Sortie désirée de l'unité 'i', provenant de la base d'apprentissage.

Si : Valeur cumulée des entrées pondérées par les poids synaptiques de l'unité 'i'.

Ai : Valeur obtenue à la sortie de l'unité 'i' après activation.

Xj : Valeur de l'entrée 'j' de l'unité en question.

E : Mesure de l'erreur (fonction d'erreur quadratique).

Wij : Poids synaptique associé à l'entrée 'j' de l'unité 'i'

Wki : Poids associé à l'entrée 'i' de l'unité 'k'. L'unité 'k' se trouve connectée à la sortie de l'unité 'i' (couche précédente).

δi : Représente l'erreur de l'unité 'i'.

δk : Représente l'erreur de l'unité 'k'. L'unité 'k' se trouve dans la couche suivante après 'i'.

ΔWij : Valeur du changement effectué sur le poids Wij.

α : Valeur constante qui sert à contrôler la vitesse de convergence de l'apprentissage (pas d'apprentissage - *learning rate*). Cette valeur est comprise dans l'intervalle [0..1].

L'algorithme de la Rétro-Propagation peut être amélioré par l'ajout d'un terme d'inertie dans le calcul du ΔWij :

$$\Delta Wij(t) = -\alpha.\delta i.Xj + \beta.\Delta Wij(t-1) \qquad \text{(Eq. 25)}$$

Où nous avons :

$\Delta Wij(t)$: Valeur du changement effectué sur le poids Wij à l'instant 't'.

$\Delta Wij(t-1)$: Valeur du changement effectué sur le poids Wij à l'instant 't-1'.

β : Valeur constante qui sert à contrôler l'inertie (*momentum*) du changement des poids. Cette valeur est comprise dans l'intervalle [0..1].

On obtient ainsi les équations finales qui représentent l'adaptation des poids d'un réseau PMC par Rétro-Propagation, avec des unités de type sigmoïde asymétrique :

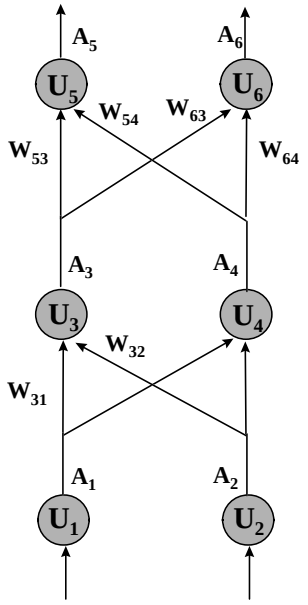
$$Wij = Wij + \Delta Wij(t) \qquad \text{(Eq. 26)}$$

$$\Delta Wij(t) = \alpha.(Di - Ai).(Ai.(1-Ai)).Xj + \beta.\Delta Wij(t-1) \qquad \text{(Unité de sortie)}$$

$$\Delta Wij(t) = \alpha.\sum_{k} \delta kWki.(Ai.(1-Ai)).Xj + \beta.\Delta Wij(t-1) \qquad \text{(Unité de la couche cachée)}$$

En résumé, le processus d'apprentissage consiste à présenter au réseau un exemple avec sa valeur de sortie désirée. L'activation du réseau va produire des valeurs de sortie qui sont ensuite utilisées pour le calcul de l'erreur sur la couche de sortie. Cette erreur des unités de sortie est ensuite propagée aux couches antérieures en utilisant les équations que nous venons de décrire (voir Figure 16). Une fois que toutes les unités ont été traitées et qu'on dispose de toutes les valeurs de $\Delta Wij(t)$, on effectue l'adaptation des poids selon l'Equation 26. Les seuils associés à chacune des unités (Θi) sont traités comme des poids connectés à des entrées qui ont une valeur constante toujours égal à 1. Ainsi, les équations d'adaptation des poids décrites ci-dessus sont appliquées exactement de la même façon pour l'adaptation des seuils des unités du réseaux.

L'adaptation des poids avec l'algorithme de la Rétro-Propagation peut être effectuée après la présentation de chaque exemple, mais on peut aussi faire le cumul des $\Delta Wij(t)$ pendant la présentation de toute la base d'exemples d'apprentissage et réaliser la mise à jour des poids seulement à la fin de cette période. Cette dernière méthode s'appelle apprentissage par paquets ou par époques.



$A_5 = Fa (W_{53}.A_3 + W_{54}.A_4 + \theta_5)$
$A_3 = Fa (W_{31}.A_1 + W_{32}.A_2 + \theta_3)$

$\Delta W_{53} = -\alpha.\delta_5.A_3 = -\alpha.[(D_5-A_5).(A_5.(1-A_5))].A_3$

$\Delta W_{54} = -\alpha.\delta_5.A_4 = -\alpha.[(D_5-A_5).(A_5.(1-A_5))].A_4$

$\Delta W_{63} = -\alpha.\delta_6.A_3 = -\alpha.[(D_6-A_6).(A_6.(1-A_6))].A_3$

$\Delta W_{64} = -\alpha.\delta_6.A_4 = -\alpha.[(D_6-A_6).(A_6.(1-A_6))].A_4$

$\Delta W_{31} = -\alpha.\delta_3.A_1 = -\alpha.[(\delta_5.W_{53} + \delta_6.W_{63}).(A_3 .(1-A_3))].A_1$

δ_i : Erreur de l'unité 'i'
α : Valeur constante ($0.0 =< \alpha =< 1.0$)
D_i : Sortie désirée de l'unité 'i'
A_i : Sortie de l'activation de l'unité 'i'
W_{ij} : Poids de l'entrée 'j' de l'unité 'i'
θ_i : Seuil de l'unité 'i'
Fa(x) : Fonction d'activation (sigmoïde)

Figure 16 - Détail du processus de Rétro-Propagation

2.3.3 Discussion sur les Réseaux Connexionnistes

Avant d'analyser les propriétés des réseaux connexionnistes, nous allons préciser ici un certain nombre de points importants relatifs à notre étude. Notre intérêt pour les réseaux réside essentiellement dans leur utilisation dans des systèmes intelligents tels que les systèmes experts. Pour réaliser ce type de systèmes, nous avons choisi d'utiliser des réseaux avec un apprentissage supervisé et nous nous sommes orientés vers des tâches de classification (plutôt que des tâches d'approximation de fonctions ou de modélisation de comportements dynamiques). Nous avons choisi d'étudier plus en détail les réseaux construits à partir d'unités à prototypes et les réseaux de type Perceptrons (PMC), car la plupart des systèmes experts connexionnistes sont dérivés de l'un de ces deux types de réseaux. Il est utile de rappeller que dans les systèmes experts connexionnistes, l'activation du réseau joue le rôle d'un moteur d'inférence d'un système expert classique, alors que ses connexions et ses poids synaptiques jouent le rôle des connaissances.

Les réseaux connexionnistes, en particulier ceux employés pour la construction de systèmes intelligents, possèdent les avantages suivants :

- *Connaissances Empiriques* : l'apprentissage à partir d'exemples (méthode d'apprentissage empirique) se fait d'une façon assez simple et permet d'obtenir de bons résultats par rapport aux autres techniques d'apprentissage automatique;

- *Dégradation progressive* : les réponses données par les réseaux se dégradent progressivement en présence de bruit dans les entrées. Les réseaux permettent de bien généraliser les connaissances présentes dans la base d'apprentissage et sont moins sensibles aux perturbations que les systèmes symboliques. Le fait de travailler sur une représentation 'numérique' des connaissances rend les réseaux plus adaptés pour manipuler des données quantitatives (valeurs continues). Les réseaux de neurones sont moins vulnérables aux données approximatives et à la présence de données incorrectes dans la base d'apprentissage;

- *Parallélisme massif* : les réseaux sont composés d'un ensemble d'unités de traitement de l'information qui peuvent opérer en parallèle. Bien que la plupart des implémentations des réseaux connexionnistes soient réalisées sur des simulateurs séquentiels, il est possible de faire des implémentations (logicielles ou matérielles) exploitant la possibilité d'activer simultanément les unités. La plupart des implémentations des réseaux de neurones peuvent être facilement converties d'une version séquentielle vers une version parallèle.

Les réseaux connexionnistes présentent aussi un certain nombre d'inconvénients, tels que :

- *Architecture et paramètres* : il n'existe pas de méthode automatique pour choisir la meilleure architecture possible pour un problème donné. Il est assez difficile de trouver la bonne topologie du réseau ainsi que les bons paramètres de réglage de l'algorithme d'apprentissage. L'évolution du processus d'apprentissage est très influencée par ces deux éléments (l'architecture du réseau et les paramètres de réglage) et dépend beaucoup du type de problème traité. Le simple fait de changer la base d'apprentissage utilisée, peut nous obliger à reconfigurer le réseau en entier;

- *Initialisation et codage* : les algorithmes d'apprentissage connexionniste sont en général très dépendants de l'état initial du réseau (initialisation aléatoire des poids) et de la configuration de la base d'apprentissage. Un mauvais choix des poids employés pour initialiser le réseau, de la méthode de codage des données, ou même de l'ordre des données, peut bloquer l'apprentissage ou poser des problèmes pour la convergence du réseau vers une bonne solution;

- *Boîte noire* : les connaissances acquises par le réseau sont codées par l'ensemble des valeurs des poids synaptiques ainsi que par la façon dont les unités sont interconnectées. Il est très difficile pour un être humain de les interpréter directement. Les réseaux connexionnistes sont des boîtes noires, où les connaissances restent enfermés et sont inintelligibles pour l'utilisateur ou pour l'expert. Un réseau ne peut pas expliquer le raisonnement qui l'a amené à une solution spécifique;

- *Connaissances théoriques* : les réseaux classiques ne permettent pas de profiter des connaissances théoriques disponibles sur le domaine du problème traité. Comme les arbres de décision, ils sont dédiés à la manipulation de connaissances empiriques. Une façon simpliste de profiter des connaissances théoriques consiste à convertir des règles en exemples (prototypes). Cependant, cette méthode ne garantit pas que ces exemples vont être bien représentés dans les connaissances du réseau à la fin de l'apprentissage, car nous sommes obligés à passer pour une phase d'apprentissage où se mélangent sans distinction des connaissances empiriques avec des connaissances théoriques codées par des exemples.

Cette liste n'est pas exhaustive (e.g. voir [ORS 95]), mais elle nous permet d'avoir une idée des principaux problèmes des réseaux. En particulier, la méthode d'apprentissage des PMC par Rétro-Propagation présente certains inconvénients, à savoir [FAH 88] :

- *Paralysie de l'apprentissage* : les réseaux PMC qui utilisent la Rétro-Propagation, en raison de la façon dont les poids sont adaptés, ont tendance à ne plus modifier les poids une fois que la sortie de l'unité est assez proche de 0 ou de 1 (pour la sigmoïde asymétrique), ou assez proche de -1 ou de +1 (pour la sigmoïde symétrique). Si l'on observe la fonction de calcul d'erreur, on constate que l'erreur d'une unité est calculée à partir de la multiplication de la dérivée de la fonction d'activation par la différence entre la sortie désirée et la sortie réelle, ou par une valeur proportionnelle à l'erreur obtenue dans la couche suivante. La dérivée de la fonction

d'activation donne des valeurs très faibles (proches de zéro) quand, en entrée de cette fonction, on a des valeurs proches des extrémités de la fonction (très négatives ou très positives, voir Figure 14). La valeur de la dérivée de la sigmoïde étant très proche à zéro, l'erreur calculée est elle aussi proche de zéro et par conséquent les poids ne vont plus évoluer. Ceci permet de donner au réseau une certaine stabilité au cours de l'apprentissage, mais peut paralyser l'adaptation des unités du réseau (*flat-spot*);

- *Instabilité et oubli catastrophique* : inversement au problème de la paralysie, les réseaux souffrent du problème de l'instabilité et de l'oubli catastrophique. Les réseaux minimisent l'erreur d'une façon non-coordonnée : chaque unité essaye de réduire l'erreur le plus possible. Le problème de cette approche est que les unités peuvent modifier excessivement leur poids et perdre les connaissances déjà acquises. Cela peut arriver dans deux circonstances : les unités changent beaucoup "d'avis" pendant l'apprentissage (*moving target problem*) ; le réseau perd des connaissances déjà acquises quand on change la base d'apprentissage (oubli catastrophique). Ainsi, dans certains cas, le réseau n'apprend plus de nouvelles connaissances ou détruit presqu'entièrement les connaissances déjà acquises;

- *Vitesse de convergence et paramètres de réglage de l'algorithme* : dans la plupart des cas, la vitesse de convergence de l'algorithme de la Rétro-Propagation est très faible. Les changements des poids doivent être faits très lentement (avec des pas très petits), pour ne pas risquer de ne pas converger vers une solution. L'algorithme de Rétro-Propagation est doté de deux paramètres α et β, qui contrôlent le pas d'apprentissage (*vitesse*) et l'inertie (*momentum*) respectivement. Ces deux paramètres permettent d'accélérer le processus d'apprentissage, mais il faut les régler précisément pour obtenir de bons résultats. Ils sont essentiels dans le processus d'apprentissage, mais posent du même coup le problème de l'estimation des valeurs à leur donner. De plus, les valeurs de α et β sont très liées à la base d'apprentissage employé et doivent être reconfigurés pour chaque nouvelle application. Une valeur de α ou de β mal choisie peut réduire à néant l'effort d'apprentissage.

Enfin, la Rétro-Propagation n'est pas un algorithme incrémental, ni au niveau de la structure du réseau, ni au niveau de la base de données. L'architecture du réseau est statique et reste difficile à définir par rapport à un problème spécifique donné. Malgré tous ces problèmes, l'algorithme de la Rétro-Propagation reste une des méthodes les plus utilisées pour l'apprentissage

des réseaux connexionnistes. Par ailleurs, conscients de ces points faibles, plusieurs chercheurs ont essayé de résoudre ces différents problèmes [SCH 94, SCH 93]. On peut mettre en évidence certaines techniques améliorées d'apprentissage : la méthode **RPROP** [RIE 93] (paramètres adaptés au cours de l'apprentissage), la méthode **QuickProp** [FAH 88] (une descente du gradient de deuxième ordre), la méthode du **Gradient Conjugué** - *Scaled Conjugated Gradient* [MOL 90] (une autre technique de deuxième ordre), ou enfin la méthode **CasCor** - *Cascade-Correlation* [FAH 90] et les autres techniques ontogéniques d'apprentissage [FIE 94] (évolution des poids et aussi de l'architecture du réseau pendant l'apprentissage).

Une question souvent soulevée est la suivante : "les réseaux de neurones ne sont-ils qu'une simple réalisation de méthodes déjà connues, à savoir les méthodes statistiques d'analyse de données ? ". Une discussion intéressante sur ce sujet est proposée par B. Orsier [ORS 95] qui conclut que les réseaux restent une approche très intéressante, même s'ils ont des points communs avec d'autres approches, car ils peuvent toujours apporter des contributions au domaine de l'analyse de données. Les réseaux possèdent des propriétés qui leur permettent de surpasser certaines méthodes classiques d'analyse de données, dans certaines situations. Avant d'affirmer qu'une méthode surpasse complètement une autre, il faut bien avoir à l'esprit qu'une méthode peut toujours être meilleure ou aussi performante que toutes les autres pour un cas particulier et unique d'application (on trouve toujours *le cas parfait* pour une méthode), mais il semble impossible d'avoir une seule méthode qui soit plus performante par rapport à un grand ensemble d'applications différentes. Une solution possible semble être la combinaison de différentes méthodes afin d'obtenir des solutions composées plus performantes.

2.4 VERS LES SOLUTIONS HYBRIDES

Nous avons présenté, dans les sections précédentes, les différents méthodes d'apprentissage automatique ainsi que leurs propriétés et limitations. Etant donnés les problèmes inhérents à chacune des différentes approches, les chercheurs essayent de trouver de nouvelles solutions plus performantes. Les solutions hybrides se présentent comme une tendance dans les recherches développées sur les systèmes intelligents [KAN 92, MIC 94, ORS 95, LAL 96, HIL 93, HIL 94]. Les systèmes experts de deuxième génération sont le symbole de cette recherche de solutions

hybrides, car on y trouve des systèmes conçus pour permettre l'intégration de différents modèles de raisonnement et d'apprentissage [NIK 97, DAV 93]. Dans cette section, nous allons tout d'abord dresser une brève liste présentant des exemples connus d'hybridation de systèmes puis discuterons sur un type de solution hybride en particulier : l'intégration des approches symbolique et sub-symbolique (connexionniste).

2.4.1 Systèmes et Méthodes Hybrides d'Apprentissage Automatique

Le concept de système hybride ou de méthode hybride est très large. Ce groupe d'applications inclut toute méthode qui intègre deux approches différentes pour la solution d'un problème. Nous allons nous concentrer sur les approches hybrides d'apprentissage automatique, c.-à-d. les différents systèmes et méthodes résultants de la combinaison des méthodes de base présentées dans ce document. Notre idée n'est pas de dresser une liste complète de toutes les possibilités de combinaison d'approches différentes, mais seulement de donner un aperçu de ce qui est fait actuellement en termes de systèmes et méthodes hybrides. Voici quelques exemples d'approches hybrides appliqués à l'apprentissage automatique ou à la conception de systèmes intelligents :

- **Systèmes Symbolique-Flous** : les systèmes symbolique-flous intègrent la logique floue (*fuzzy-logic*) et les systèmes à base de connaissances (*KBS*). Ce type de système est en réalité une extension des systèmes à base de connaissances, dans lesquels on ajoute la possibilité de représenter des règles floues et de les manipuler à travers des mécanismes d'inférence floue. Un exemple de ce type de systèmes est le *Fuzzy-CLIPS* [IIT 96], une version du langage CLIPS adaptée au traitement de la logique floue. Un autre système connu, plus ancien, est le *FLIE* [VES 93].

- **Systèmes Symbolique-Génétiques** : ces systèmes sont normalement composés d'un module génétique responsable de l'acquisition de connaissances à partir des données (apprentissage), et d'un module symbolique responsable du moteur d'inférence symbolique (raisonnement). Des exemples de ce type de systèmes sont les systèmes *DELVAUX* [EIC 96], *COGIN* [GRE 93] et *GABIL* [DEJ 93b].

• **Systèmes Génético-Flous** : une des possibilités d'application des algorithmes génétiques à la logique floue consiste à les utiliser pour l'optimisation des ensembles flous. Un exemple de systèmes génético-flous de ce type est celui décrit par A. Homaifar [HOM 95].

• **Systèmes Neuro-Génétiques** : la plupart des systèmes neuro-génétiques sont conçus afin de contourner le problème relatif au choix de l'architecture du réseau. Les réseaux qui relèvent de ce type d'approche sont aussi appelés réseaux évolutifs (*evolutinary ANN* [POR 97, PAT 97, BAL 95, FLO 94, NOL 94]). L'idée de base repose sur l'utilisation des algorithmes génétiques pour faire évoluer et sélectionner les architectures de RNA plus performantes par rapport à une application donnée. On trouve aussi dans la littérature d'autres types moins répandus d'hybridation neuro-génétique, tels que : adaptation des poids synaptiques réalisée par un algorithme génétique [POT 92], adaptation des paramètres de vitesse de convergence et d'inertie (α et β) par un algorithme génétique [SAL 96].

• **Systèmes Neuro-Symboliques:** ce type de système hybride est sans doute le plus étudié parmi toutes les approches hybrides que nous avons décrites. Les systèmes hybrides neuro-symboliques peuvent être encore divisés en sous-groupe :

♦ **Systèmes Neuro-Flous** : les systèmes hybrides neuro-flous sont une des catégories de systèmes hybrides les plus développées car la logique floue et les réseaux connexionnistes ont beaucoup de points en commun. On trouve dans la littérature un grand nombre de références sur ce type de système [JAN 96, KRZ 97, KAS 95, GON 96, NAU 95b, HAY 94, ALC 93, COX 92, POS 92, ROC 92]. Les systèmes neuro-flous sont principallement de trois types : systèmes qui intègrent des règles floues dans des réseaux (le modèle RBF étant un des plus utilisés) ; les systèmes qui font l'extraction de règles floues à partir des réseaux ; et les systèmes qui implementent des neurones flous. On peut citer comme exemple les systèmes suivantes : *FuzzyARTMAP* [CAR 92], *FUN* [SUL 93], *FuzzyCOPE* [KIL 97, KAS 96], *NEFCON-NEFCLASS-NEFPROX* [NAU 97], *ANFIS* [JAN 93] et *FUNEGEN* [HAL 94].

- **Systèmes Neuro-IDT** : les systèmes hybrides qui combinent des réseaux de neurones avec des arbres de décision (IDT) relèvent de deux approches principales : la construction et initialisation d'un réseau à partir d'un arbre de décision [LEE 95, SAH 95], ou l'extraction d'un arbre de décision à partir d'un réseau après la fin de son apprentissage [BOZ 97b, CRA 96a, CRA 96b, MED 95]. L'insertion d'un arbre de décision dans un réseau sert à simplifier le choix de son architecture et de ses poids initiaux. Cette approche apporte une solution alternative au problème de la construction des réseaux. L'autre approche (extraction d'arbres de décision) adresse le problème du manque d'une représentation symbolique des réseaux. L'extraction d'un arbre de décision va permettre de obtenir des explications sur le raisonnement employé pour la solution d'un problème. Ainsi, l'arbre de décision obtenu à partir d'un réseau permet d'entrer dans la "boîte noire" et de mieux comprendre son fonctionnement. En plus de ces deux approches, on trouve aussi dans la littérature des approches vraiment mixtes, où les noeuds de l'arbre de décision peuvent être remplacés par des unités d'un réseau connexionniste [ALC 93, UTG 88].

- **Systèmes Neuro-CBR** : l'intégration d'un réseau connexionniste et d'un système de raisonnent fondé sur des cas permet d'induire des connaissances de plus haut niveau, tout en gardant certaines des avantages des systèmes CBR. Un exemple de ce type d'approche est le système *ProBis* [MAL 96a, MAL 96b]. Ce système permet d'obtenir un réseau avec des prototypes que représentent les cas appris. Il permet aussi de garder les cas atypiques et frontières dans la mémoire du système CBR. De cette façon, on peut profiter au même temps d'une représentation à base de prototypes (cas généralisés) et d'une représentation à base de cas (cas particuliers).

- **Systèmes Neuro-KBS ou SHNS** : les systèmes qui integrent des réseaux connexionnistes et des systèmes à base de connaissances (KBS) sont couramment appelés systèmes hybrides neuro-symboliques (SHNS). Nous allons utiliser plutôt ce dernier terme pour faire référence à tous les systèmes qui permettent d'intégrer des règles symboliques (connaissances symboliques) avec des réseaux connexionnistes

(connaissances sub-symboliques). Dernièrement, un très grand nombre de publications a été consacré à ce sujet [SUN 97, NIK 97, MEM 97, HIL 96b, AND 96, SUN 95a, GAL 95, HAL 95, GOO 95, HON 95, MED 95, MED 94, HON 94, FU 94, SUN 94, HIL 94b, BOO 93, KAN 92, DIN 92, MIN 90, et, MAH 96, LAL 96, ORS 95, GIA 92, TOW 91].

On trouve actuellement un bon nombre d'exemples de systèmes SHNS, tels que :

- Le système *SYNHESYS* - KBS + RNA à Prototypes [GIA 92, ORS 95];
- Les réseaux *KBANN* - Règles symboliques + RNA type PMC [TOW 91]
- Le système *INSS* - KBS + RNA type PMC [OSO 95a, OSO 97, AMY 97];
- L'algorithme *EBNN* - EBL + RNA [MIT 97];
- Les réseaux *KBCNN* - Règles symboliques + RNA type PMC [FU 94];
- Le système *SCANDAL* - KBS + RNA type PMC [STU 96];
- Le système *RAPTURE* - Règles + RNA type CFBP [MAH 96, MAH 93];
- Le système *MACIE* - Système expert basé sur les RNA à PMC [GAL 93, GAL 88];
- Le réseau *RUBICON* - Base de règles codés dans RNA (sans apprentissage) [SAM 92];
- Le système *CONSYDERR* - Traitement basé sur des règles et sur la similarité [SUN 95a];
- Le système *CORE* - Interaction entre deux modules, KBS et RNA [KAS 90].

Nous avons fait le choix d'orienter les travaux réalisés dans cette thèse sur ce dernier type de systèmes : les systèmes hybrides neuro-symboliques - SHNS. Notre idée est d'utiliser l'intégration des approches symbolique et connexionniste afin de profiter de la complémentarité existante entre ces deux types d'approches. Dans la section suivante, nous allons présenter certaines des caractéristiques que nous avons souhaité obtenir à travers la réalisation de ce type d'intégration.

2.4.2 Vers les Systèmes Hybrides Neuro-Symboliques

Dans ce chapitre, nous avons présenté différentes méthodes d'apprentissage automatique. Nous avons discuté sur ses particularités, ses propriétés et ses limitations. Il est clair que chacune des approches, considérée isolément, reste assez limitée et impose toujours des restrictions au niveau de l'apprentissage. Afin d'exploiter les avantages particulièrs de chaque approche, symbolique et connexionniste, et de surmonter ses limitations, nous avons décidé d'utiliser une approche hybride. Notre but est d'arriver à des systèmes experts basés sur une approche d'acquisition de connaissances qui satisfasse les items suivants :

- Possibilité d'**exploiter toutes les connaissances disponibles** sur le problème traité, c'est-à dire exploiter les **connaissances théoriques** tout aussi bien que les **connaissances empiriques.** Le système d'apprentissage automatique doit être capable de traiter des bases de règles et d'exemples;

- Possibilité de traiter des problèmes où les connaissances ne sont ni complètes ni tout à fait correctes. Les connaissances disponibles sur un problème sont la plupart du temps des **connaissances imparfaites** et des **connaissances qui évoluent** au cours du temps;

- Possibilité de traiter les **données symboliques** tout autant que les **données numériques**. Les connaissances sont représentées par des informations qui contiennent des variables discrètes et des variables continues. Un système intelligent capable de manipuler des données continues, doit aussi être capable de **manipuler des informations approximatives**;

- Possibilité de représenter des connaissances de bas niveau et de haut niveau. Les connaissances doivent être représentées de telle manière qu'on puisse travailler aussi bien sur un **niveau symbolique** que sur un **niveau sub-symbolique**. Le formalisme de **représentation des connaissances** doit être assez **robuste** pour permettre de coder différentes types de relations existant entre les informations.

En conclusion, nous allons nous diriger vers les systèmes hybrides pour que l'on puisse créer des systèmes experts plus robustes. En particulier, les systèmes hybrides basés sur une approche neuro-symbolique se présentent comme des méthodes d'acquisition de connaissances très intéressantes. La complémentarité existante entre ces deux types d'approches (voir section 3.1), nous amène donc à la création de solutions hybrides neuro-symboliques, qui vont nous permettre de résoudre une grande partie des problèmes que nous avons posés jusqu'ici. Dans le chapitre suivant, nous allons présenter plus en détails les SHNS et ses propriétés, ainsi que nous allons donner des exemples de SHNS qui se proposent à résoudre certains des problèmes qui ont été abordés tout au long de ce chapitre.

3. SYSTEMES HYBRIDES NEURO-SYMBOLIQUES

Les Systèmes Hybrides Neuro-Symboliques (SHNS) sont des systèmes qui intègrent des approches de l'I.A. symbolique avec des approches de l'I.A. connexionniste. L'intégration de ces deux types d'approches peut être faite de différentes façons. On peut, par exemple, intégrer des connaissances issues de différentes origines et représentations, faire des traitements coopératifs, faire des transferts entre des modules symboliques et des modules connexionnistes, etc. Compte tenu de l'existence de tous ces différents modes d'intégration neuro-symbolique, nous allons présenter dans ce chapitre (voir section 3.2) une classification des SHNS avec des exemples de systèmes connus. Avant de passer à la classification des SHNS, nous allons tout d'abord présenter un bref résumé des raisons que nous amènent à les utiliser et des avantages liées à l'intégration de ces deux approches.

3.1 GENERALITES

Les deux approches, symbolique et connexionniste, présentent des propriétés particulières à chacun. Ces propriétés constituent un ensemble d'avantages et de désavantages liés au choix de l'utilisation de l'une ou de l'autre approche. Le Tableau 3 dresse, d'une manière assez simple et résumé, une liste des comparaisons des propriétés de ces deux approches [LAL 96, BOZ 95, OSO 95, GIA 92, TOW 91].

I.A. Symbolique	I.A. Connexionniste
☺ - L'insertion de connaissances théoriques sur le problème peut être faite d'une façon simple et directe. Il suffit de les expliciter et de les convertir dans le formalisme de représentation de connaissances utilisé.	☹ - Usuellement, on ne peut pas profiter des connaissances théoriques disponibles sur le problème traité. Il faut toujours des exemples pour pouvoir acquérir des connaissances.
☹ - Le traitement est séquentiel, les temps de réponse lors de la consultation du système sont longs.	☺ - Les réseaux sont composés d'une série d'unités de traitement de l'information qui peuvent opérer en parallèle, avec un temps de réponse très rapide.

☺ - L'insertion de connaissances (e.g. règles) peut être faite très rapidement une fois qu'elles ont été déjà traitées par les experts et/ou ingénieur de connaissances.	☹ - Le processus d'apprentissage peut être assez long puisque les poids (connaissances) sont adaptés petit à petit.
☹ - L'apprentissage n'est pas un processus à la base de ce type de systèmes. L'acquisition de connaissances se fait plutôt par explicitation. Par conséquence, on a le *problème du goulot d'étranglement* de l'acquisition de connaissances dans les systèmes experts.	☺ - L'apprentissage et la généralisation de connaissances à partir d'un ensemble d'exemples sont les points forts des approches connexionnistes. L'apprentissage est le processus de base des approches connexionnistes.
☺ - Les approches symboliques permettent d'obtenir des explications sur les réponses données par le système. On peut justifier les réponses basées sur le processus de raisonnement employé par le système et sur les connaissances codés dans sa base. Les connaissances sont codées dans un langage proche du langage naturel, et donc facilement interprétable.	☹ - Les réseaux sont des "boîtes noires", où les connaissances sont codées dans les poids et les interconnexions. On n'a pas accès à une forme compréhensible de ces connaissances, qui puisse être interprété directement par un être humain afin d'expliquer les réponses obtenues.
☹ - Usuellement, pour que le système puisse bien fonctionner, les connaissances théoriques doivent être à la fois correctes et complètes sur le domaine traité. L'approche symbolique n'est pas adaptée aux traitement d'informations approximatives ou incomplètes. L'utilisation de variables numériques pose des problèmes.	☺ - Les réponses du système se dégradent progressivement en présence d'une entrée bruitée. Les réseaux sont très adaptés au traitement d'informations approximatives et incomplètes. Les variables continues ne posent pas de problèmes particuliers par rapport au traitement connexionniste.
☺ - Les connaissances sont représentées par des règles et par des structures de données. Ce sont des connaissances de niveau supérieur.	☺ - Les connaissances codées dans les réseaux représentent bien les relations existantes entres leurs variables d'entrée.
☹ - Le développement d'un système expert est une tache difficile et assez longue.	☹ - La détermination de l'architecture et des paramètres d'un réseau connexionniste peut être une tache difficile et de longue durée.

Tableau 3 - Propriétés de l'approche symbolique et de l'approche connexionniste

On peut constater que l'approche symbolique et l'approche connexionniste sont tout à fait complémentaires. Les SHNS essayent d'exploiter cette complémentarité afin d'améliorer leurs performances et d'avoir des solutions plus robustes. De plus, l'intégration de ces deux approches permet de diminuer, sinon de résoudre complètement, certaines limitations de l'une ou de l'autre approche [MEM 97]. L'intégration neuro-symbolique va aussi nous permettre de nous adresser à un certain nombre de propriétés souhaitées pour les système d'apprentissage automatique, tels que

les items décrits dans la section 2.4.2. Enfin, le choix du type d'intégration entre les deux approches va être lui aussi très important, puisqu'il va déterminer les principales propriétés du système hybride.

3.2 TYPES D'INTEGRATION ET CLASSIFICATION DES SHNS

Plusieurs classifications sur les modes d'intégration des systèmes hybrides ont été proposées [LAL 96, HIL 96b, ORS 95, MED 92]. Ces classifications sont plus ou moins semblables. Chaque auteur a essayé de mettre en évidence un aspect par rapport aux autres. Nous avons une interprétation plus simple des différentes classifications proposées. Pour nous, plutôt que d'essayer de créer une seule classification hiérarchique et globale des systèmes hybrides, le plus simple est de faire des classifications en ne considérant qu'une seule propriété à la fois. Nous allons donc reprendre les classifications proposées par d'autres auteurs [LAL 96, ORS 95, HIL 94a, MED 92] et nous allons les diviser en plusieurs groupes considérés séparément les uns des autres. Ainsi, comme pour la classification des différents types de réseaux connexionnistes, nous allons pouvoir classer un système SHNS en considérant séparément chacune de ses propriétés. Nous pensons qu'il est très difficile de classer tous les systèmes dans une structure hiérarchique puisque les systèmes partagent certaines propriétés, en raison de l'hybridation de ses composantes. Nous allons classer les SHNS selon les critères suivants :

- le type d'intégration (approche unifiée, semi-hybride, hybride);
- le degré de couplage (couplage faible, couplage moyen, couplage fort);
- le mode d'intégration (traitement chaîné, sous-traitance, méta-traitement, co-traitement);
- les transferts des connaissances ($S \Rightarrow C$, $S \Leftarrow C$, $S \Leftrightarrow C$);
- le type de représentation des connaissances (localiste, distribuée, combinée);
- le type de codage des connaissances (règles d'ordre 0, règles d'ordre 0^+, règles floues, règles probabilistes, prédicats, arbre de décision, réseaux PMC, réseaux à prototypes, etc);
- le type de méthode de raisonnement (chaînage avant, chaînage arrière, résolution, propagation d'activations - connexionniste);

- le mode d'acquisition de connaissances (apprentissage continu, apprentissage non-continu, apprentissage supervisé, apprentissage non-supervisé);

Les différentes classifications présentées ici doivent beaucoup au travaux développés dans le cadre du projet MIX [HIL 93] et particulièrement aux travaux de B. Orsier, Y. Lallement et M. Hilario.

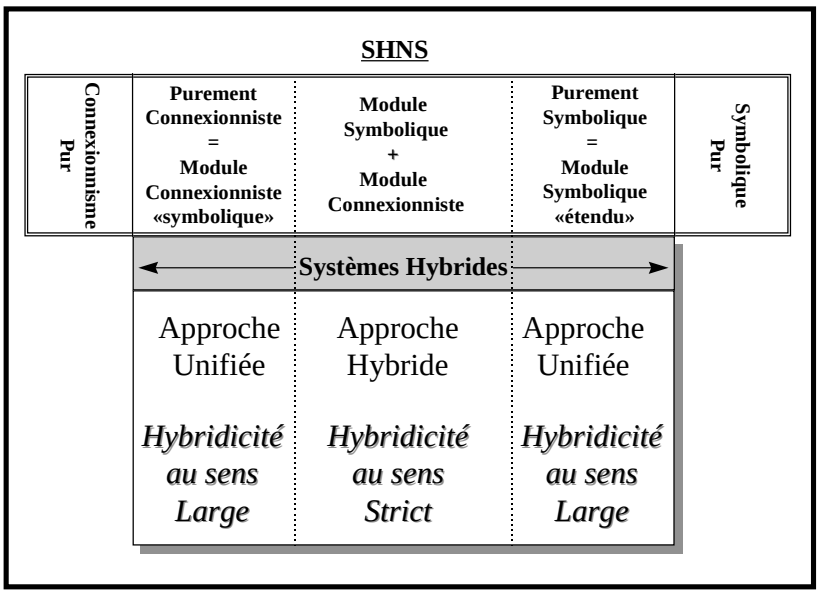


Figure 17 - Intégration Neuro-Symbolique

3.2.1 Types d'Intégration

L'intégration neuro-symbolique peut être classé dans deux grands groupes principaux, selon le "type d'hybridicité" de l'approche employée (voir Figure 17) :

• **Approche Unifiée** : ce type d'approche est aussi connu comme un approche "*pure*" ou comme une *hybridicité au sens large*. L'idée ici est d'intégrer tout dans un seul module, uniquement connexionniste ou uniquement symbolique, les deux types d'approches (symbolique+connexionniste). L'approche unifiée essaye d'integrer les proprietés des systèmes

symboliques et des systèmes connexionnistes dans une solution unique qui les unifie. Cela donne l'origine à des *systèmes purement connexionnistes* (connexionnisme éliminatoire, le tout-connexionniste) et à des *systèmes purement symboliques* (le tout-symbolique). Les systèmes purement connexionnistes essayent de réaliser des processus d'inférence symbolique (e.g. résolution, unification de variables) à travers l'utilisation des réseaux de neurones. Ces systèmes sont aussi appellés CSP - *Connectionist Symbolic Processing*. Les systèmes purement symboliques essayent de réaliser des extensions des processus d'inférence symbolique, afin de leur permettre de raisonner sur l'incertain, sur des valeurs continues et approximatives, sur des probabilités, etc.

- **Approche Semi-Hybride** : ce sont des approches où l'on trouve des travaux relatifs aux traductions. Cela regroupe la *compilation* d'une base de règles dans un *réseau (insertion de connaissances*) et l'*explicitation* de règles à partir d'un réseau (*extraction de connaissances*). Certains systèmes sont dits semi-hybrides parce qu'ils ne réalisent qu'un des deux types de transferts de connaissances entre les modules symbolique et connexionniste. Pour nous, à partir du moment où un système offre la possibilité d'insérer et aussi d'extraire des règles, il devient un système hybride complet (*hybridicité au sens strict*), puisque on peut le coupler facilement avec un moteur d'inférence symbolique. Il est vrai qu'on doit aussi considérer le type des règles qui sont insérées et le type des règles extraites du réseau. Cette information est très importante puisqu'elle va nous permettre de savoir si l'on pourra par la suite utiliser facilement les règles dans un module symbolique. Elle nous permet aussi de savoir si l'on va pouvoir garder le même type de "capacité" de représentation de connaissances au niveau de l'insertion de règles ainsi qu'au niveau de l'extraction de règles d'un réseau.

- **Approche Hybride** : ce type d'approche est basé sur une véritable intégration d'au moins deux modules : un module symbolique et un module connexionniste. On peut aussi envisager une intégration à plusieurs modules symboliques et connexionnistes. Donc, dans ce type d'approche on retrouve au moins un système d'inférence symbolique qui travaille en coopération avec au moins un système d'inférence connexionniste. Cettte approche est connue comme l'*hybridicité au sens strict*.

Dans cette thèse, les références que nous faisons aux Systèmes Hybrides Neuro-Symboliques sont faites dans le contexte d'une *hybridicité au sens strict*, sauf si le contraire est indiqué.

3.2.2 Degré de Couplage

Le degré de couplage définit la force d'interaction entre les deux modules. La classification des différents degrés de couplage pourrait être faite selon une échelle continue pour représenter une progression d'une extrémité (pas de couplage) à l'autre (complètement intégrés). Nous allons présenter ici une classification en trois niveaux :

- **Couplage Faible** : dans ce type d'architecture, les divers modules (deux en général) sont reliés par une simple relation d'entrée/sortie, et les communications sont donc unidirectionnelles (*loosely coupled systems*). L'interaction entre les modules est nettement localisée dans l'espace et dans le temps.

- **Couplage Moyen** : dans cette catégorie, les interactions entre les modules sont plus souples, car elles sont bidirectionnelles; il ne s'agit plus simplement de relations d'entrée/sortie, et chacun des modules peut influencer dans une certaine mesure le fonctionnement de l'autre (*couplage étroit* ou *tightly coupled systems*). Par exemple, un module symbolique peut utiliser un réseau de neurones pour traiter le problème, mais en même temps il peut aussi influencer son apprentissage.

- **Couplage Fort** : dans ces systèmes, les modules partagent des structures de données communes, ou au moins l'un des modules a un accès direct à certaines structures de données de l'autre (*fully coupled*). Les connaissances et les données ne sont pas uniquement transférées, elles sont aussi partagées entre les modules qui utilisent des structures internes communes. Tout cela autorise des communications beaucoup plus souples et plus fines.

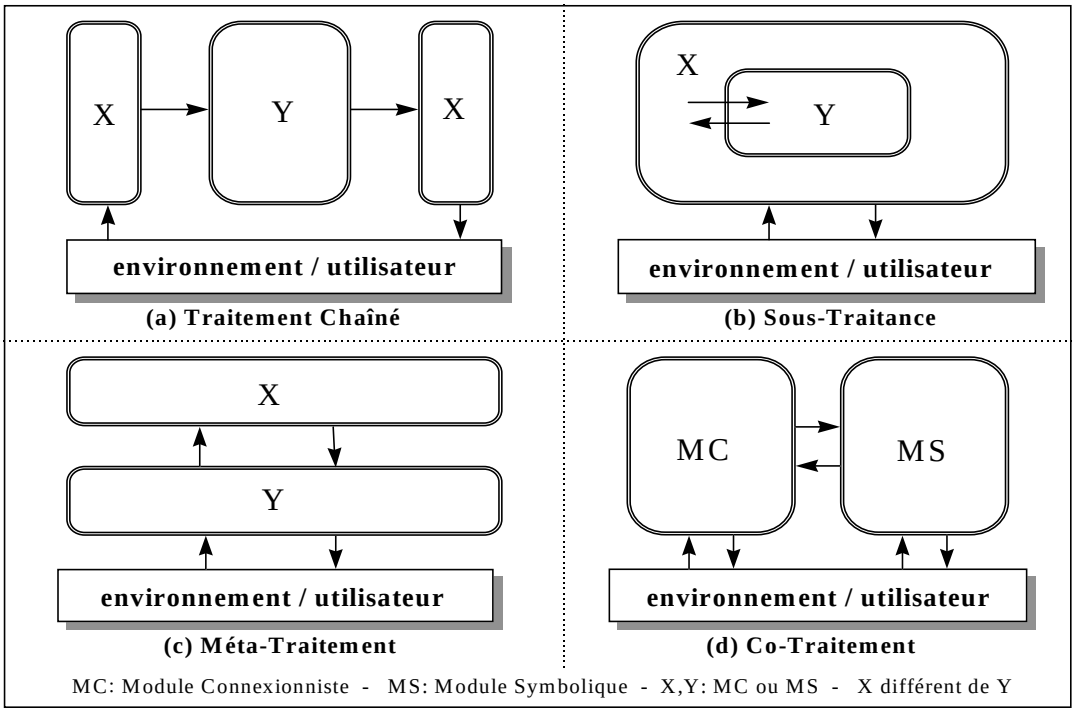


Figure 18 - Modes d'Intégration Neuro-Symbolique

3.2.3 Mode d'Intégration

Les modes d'intégration des systèmes hybrides (au niveau de leur structure) peuvent être classés dans les groupes suivants (voir Figure 18) :

• **Traitement en Chaîne** : le traitement est fait séquentiellement par un module, puis par l'autre (*chainprocessing*). Par exemple, un module connexionniste peut assurer un pré-traitement de données destinées à un module symbolique, ou l'inverse. Ce mode d'intégration est constitué par des interactions du type *pré-traitement* ou *post-traitement*, où les sous-systèmes sont reliés par une simple relation d'entrée/sortie.

• **Sous-Traitance** : le traitement principal est assuré par un des deux modules, qui utilise l'autre comme un prestataire de service pour résoudre des points particuliers (*subprocessing*). Un

module principal symbolique peut par exemple appeler un module connexionniste à une certaine étape du traitement. Dans ce type d'interaction, un des modules est entièrement subordonné à l'autre, qui décide quand l'appeler et comment utiliser ses sorties. Le module de sous-traitance n'a pas de couplage direct avec l'environnement ou avec l'utilisateur.

• **Méta-Traitement** : un des deux modules joue le rôle du résolveur de problèmes de base, et l'autre a un rôle de méta-niveau (tel que la surveillance, le contrôle ou l'amélioration des performances) par rapport au premier (*metaprocessing*). Un module symbolique peut par exemple optimiser le fonctionnement d'un réseau connexionniste.

• **Co-Traitement ou Coopération** : les deux modules sont des partenaires égaux dans le processus et coopèrent pour résoudre le problème (*coprocessing*). Un des deux modules pourra par exemple s'intéresser à un aspect particulier du problème et l'autre module au reste du problème, ou encore les deux pourront proposer des solutions qui seront ensuite évaluées par un module tiers. Chacun peut interagir directement avec l'environnement.

Les différents modes d'intégration présupposent l'existence d'un *Module Symbolique* (MS) et d'un *Module Connexionniste* (MC) bien définis. Chaque module peut être facilement discerné de l'autre en raison de ces propriétés particulières.

3.2.4 Transferts de Connaissances

Les transferts de connaissances entre les modules symbolique et connexionniste (voir item types d'intégration - semi-hybride et hybride) peuvent être classés selon la direction des échanges réalisés :

• **Du Symbolique vers le Connexionniste** : les connaissances symboliques sont transférées du module symbolique et intégrés dans le module connexionniste ($S \Rightarrow C$). A notre avis, les

méthodes de compilation de règles dans un réseau sont les plus intéressantes. Il existe aussi la possibilité de créer des bases d'apprentissage à partir des connaissances symboliques, mais dans cette approche on n'est pas sûr d'avoir à la fin de l'apprentissage un réseau qui intègre parfaitement toutes les connaissances symboliques (on est obligé de passer par une phase d'apprentissage).

• **Du Connexionniste vers le Symbolique** : les connaissances acquises par apprentissage dans les réseaux connexionnistes peuvent être explicités sous la forme de règles symboliques ($S \Leftarrow C$). L'extraction de règles à partir d'un réseau va permettre leur utilisation dans un module symbolique. Les règles obtenues à partir d'un réseau peuvent être classées dans différents groupes selon le type de codage des connaissances utilisé (voir item ci-dessous - types de codage).

• **Les Transferts Bilatéraux** : les connaissances peuvent être transférées dans les deux sens entre les deux modules symbolique et connexionniste ($S \Leftrightarrow C$). Les systèmes hybrides avec transferts de connaissances bilatéraux incluent normalement des mécanismes de compilation et d'extraction de règles à partir des réseaux. Une fois de plus, il faut se rappeler de l'importance du type de règles qui peuvent être insérées dans les réseaux ainsi que du type de règles qui en sont extraites. Un système hybride avec un bon couplage doit permettre l'insertion et l'extraction de règles avec le même pouvoir de représentation de connaissances.

Il faut préciser qu'ici nous avons classé les systèmes hybrides uniquement par rapport aux transferts de connaissances, tandis que dans les autres groupes de classification présentés nous faisons référence plutôt à des transferts entre les modules de toute sorte d'information (e.g.: paramètres, entrées/sorties, connaissances). Des exemples de méthodes d'insertion et d'extraction de connaissances à partir des réseaux connexionnistes vont être présentés plus en détails dans la section 3.3.

La capacité de faire des transferts de connaissances est une des caractéristiques les plus intéressantes des SHNS, puisqu'elle nous permet d'échanger les connaissances disponibles entre les modules symbolique et connexionniste. La compilation de règles nous permet d'insérer des

connaissances initiales dans un réseau, ce qui devrait entraîner une optimisation du processus d'apprentissage. L'extraction de règles nous permet d'expliciter les connaissances d'un réseau qui par conséquent ne sera plus une boîte noire fermée aux yeux de l'utilisateur. Les règles extraites peuvent nous aider à mieux comprendre les réponses données par les réseaux et peuvent nous fournir des informations complémentaires sur le problème traité. Le module symbolique peut aussi profiter des connaissances extraites à partir d'un réseau pour améliorer sa base de connaissances.

Dans la suite de cette thèse, nous avons donnée beaucoup d'importance aux processus de compilation et d'extraction de règles dans les systèmes hybrides, car ces deux processus sont la clef de l'intégration entre les modules symbolique et connexionniste.

3.2.5 Type de Représentation des Connaissances

Les connaissances peuvent être représentées dans les réseaux connexionnistes (approche connexionniste unifiée) de trois façons différentes :

- **Représentation Localiste** : c'est un mode de représentation où chaque noeud du réseau est utilisé pour représenter un concept isolé. Usuellement, il est beaucoup plus facile d'incorporer des connaissances symboliques dans un réseau avec un mode de représentation de connaissances localiste. Chaque unité du réseau va correspondre à un concept. Les réseaux à base de prototypes (e.g. RBF) sont des modèles localistes.

- **Représentation Distribuée** : c'est un mode de représentation où tout un ensemble de noeuds du réseau est utilisé pour représenter chaque concept. La représentation de connaissances est faite d'une façon repartie : un concept est représenté par un ensemble d'unités, chaque unité peut servir à coder plusieurs données. L'apprentissage connexionniste amène plutôt à des représentations distribuées. Tel est le cas des modèles du type PMC utilisant l'apprentissage par rétro-propagation du gradient de l'erreur. L'intégration de connaissances symboliques par une

représentation distribuée dans les réseaux connexionnistes se fait plutôt par apprentissage que par compilation.

- **Représentation Combinée** : certains systèmes peuvent intégrer les deux types de représentation dans un seul réseau. Un exemple de réseau avec une représentation combinée est celui des réseaux obtenus par compilation de règles (représentation localiste) et qui après passent par une phase d'apprentissage (représentation distribuée) afin de raffiner cette base de règles. Les systèmes hybrides qui travaillent d'un côté avec une représentation symbolique (localiste) des connaissances et de l'autre côté avec un réseau connexionniste à représentation de connaissances distribuée sont un autre exemple de l'approche combinée.

Nous parlons ici de la classification des types de représentation de connaissances seulement dans les approches connexionnistes unifiées puisque dans les approches symboliques la représentation de connaissances est essentiellement localiste. Dans les approches symboliques, une classification qui peut être intéressante à étudier est celle qui va identifier les types d'interrelations entre les différents concepts (e.g. concepts isolés, concepts avec des relations contextuelles, concepts fortement liés, concepts contradictoires).

3.2.6 Type de Codage des Connaissances

Les connaissances acquises par les systèmes hybrides peuvent être représentées de manière interne de différentes façons. Les principaux types de représentation sont : par des règles d'ordre 0, par des règles d'ordre 0^+, par des règles floues, par des règles probabilistes, par des prédicats de premier ordre, par des arbres de décision, par des réseaux PMC, par des réseaux à prototypes. Nous avons déjà discuté à propos de ces différentes types de codage (représentation) de connaissances dans le Chapitre 2. Dans la suite de ce chapitre, nous allons faire référence principalement aux groupes cités ci-dessus. Une grande partie des systèmes hybrides neuro-symboliques (hybridicité au sens strict) travaillent avec des règles symboliques d'ordre 0 ou d'ordre 0^+.

3.2.7 Type de Méthodes de Raisonnement

Les systèmes hybrides utilisent différentes méthodes d'inférence afin de résoudre les problèmes. Les approches unifiées sont basées sur une seule méthode d'inférence, tandis que les approches hybrides (hybridicité au sens strict) sont basées sur l'intégration de deux modules différents, ce qui normalement implique l'utilisation de deux types différents de raisonnement. Les principaux types de méthodes de raisonnement à considérer sont : le chaînage avant, le chaînage arrière, la résolution, le parcours d'un graphe (ou d'une arbre) et la propagation d'activations (approche connexionniste).

3.2.8 Mode d'Acquisition de Connaissances

Nous allons mettre en évidence certains modes d'acquisition de connaissances qui permettent de mieux discerner les différences existantes entre les systèmes hybrides que nous avons étudiés. Les deux premiers types d'acquisition de connaissances sont l'apprentissage en mode continu (incrémental), considéré par opposition à l'apprentissage en mode non-continu. Les systèmes avec un mode continu d'apprentissage doivent permettre l'ajout de nouvelles connaissances à des connaissances déjà acquises. Certains systèmes permettent seulement un raffinement des connaissances sans pourtant pouvoir acquérir des connaissances vraiment nouvelles sur le problème traité. Les deux autres types de modes d'acquisition de connaissance à considérer sont l'apprentissage supervisé et l'apprentissage non-supervisé. Ceux derniers sont aussi des traits importants pour l'identification des propriétés des SHNS.

3.2.9 Classification des SHNS

Nous allons présenter ici une brève liste de systèmes hybrides neuro-symboliques décrits selon leurs différentes propriétés. Notre idée n'est pas de dresser ici une liste exhaustive de tous les systèmes existants, mais seulement de revoir certains systèmes hybrides connus et de les classer dans les groupes que nous avons décrits. La plupart des systèmes que nous allons présenter ici sont décrits plus en détails dans les ouvrages de base que nous avons consulté sur les systèmes hybrides [LAL 96, HIL 96b, ORS 95, HIL 94a, GIA 92, MED 92].

Les systèmes que nous avons étudié sont les suivantes :

- **SYNHESYS :** le système hybride SYNHESYS (*SYmbolic and Neural Hybrid Expert SYstem Shell*) a été développé au laboratoire LIFIA, par A. Giacometti [GIA 92, ORS 94, ORS 95]. Ce système est composé de deux modules : un module connexionniste basé sur un réseau à prototypes de type ARN2 et un module symbolique d'inférence par chaînage avant et arrière. Le système SYNHESYS permet de consulter les deux modules en parallèle et de faire des transferts de connaissances entre eux dans les deux sens. Nous allons présenter ce système plus en détails dans la section 3.3.1.

- **ProBis :** le système hybride ProBis (*Prototype-Based Indexing System*) a été développé à l'IMAG à Grenoble dans le cadre d'une collaboration entre les laboratoires LEIBNIZ et TIMC, par M. Malek [MAL 95, MAL 96a, MAL 96b]. Ce système réalise une parfaite intégration entre un réseau à prototypes et un module de raisonnement fondé sur des cas. De cette façon, on peut profiter en même temps d'une représentation à base de prototypes (cas généralisés) et d'une représentation à base de cas (cas particuliers). Il permet d'obtenir un réseau à prototypes qui représentent les cas appris. Il permet aussi de garder les cas atypiques et frontières dans la mémoire du système CBR.

- **NESSY3L :** le système hybride NESSY3L (*NEuroSymbolic SYstem with 3 Levels*) a été développé au Lab. LIFIA/LEIBNIZ à Grenoble, par B. Orsier [ORS 95, ORS 97]. Ce système est composé de trois niveaux (niveau symbolique, niveau neuro-symbolique et niveau neuronal), à la différence de la plupart des systèmes hybrides qui n'ont qu'un module symbolique et un autre

connexionniste. Avec l'utilisation de plusieurs niveaux, on augmente la simplicité des interfaces et la modularité du système. Ainsi, on peut faire une meilleure division des tâches entre les composantes du système hybride. Le niveau supplémentaire est basé sur le codage de règles dans un 'réseau versatile'. Ce système a été applique à des études d'évitement d'obstacle en robotique.

- **INSS :** le système hybride INSS (*Incremental Neuro-Symbolic System*) a été développé au Lab. LEIBNIZ à Grenoble, par F. Osório [OSO 95, DEC 95, DEC 96, OSO 97, AMY 97]. Ce système a été conçu à partir des idées de base des systèmes SYNHESYS et KBANN. Il est composé par deux modules (symbolique et connexionniste) et permet le transfert des connaissances dans les deux sens. Il met en pratique une méthode d'acquisition constructive de connaissances. Nous allons présenter ce système plus en détails dans le Chapitre 4.

- **KBANN :** les réseaux KBANN (*Knowledge Based Artificial Neural Networks*) ont été développés à l'Université de Wisconsin-Madisson aux Etats-Unis, par G. Towell [TOW 91, TOW 93, SHA 95]. Nous allons présenter ce système plus en détails dans la section 3.3.2.

- **SCANDAL** : Le système SCANDAL (Symbolic-Connectionist Architecture for Neural Design and Training) a été développé au CUI à l'Université de Genève par M. Hilario, A. Rida, B. Orsier et M. Stückelberg [HIL 95c, STU 96, AMY 97]. Ce système utilise une approche similaire à KBANN et INSS, puisqu'il permet aussi l'insertion de règles dans un réseau de type PMC. Le réseau peut être entraîné avec des exemples, et il contient des méthodes d'extraction de règles. La particularité de cette approche est due au fait qu'il a été réalisé à l'aide d'une architecture modulaire composée de différents agents spécialisés. Les agents sont capables par exemple : de construire des arbres de décision et d'obtenir des règles par l'utilisation d'algorithmes comme ceux de C4.5 [QUI 93] ; de construire des bases d'exemples artificiels à partir de règles ; de coder des règles symboliques dans un réseau ; de réaliser l'apprentissage des réseaux avec les algorithmes RPROP [RIE 93], Cascade2 [FAH 95], ou même SCG (Scaled Conjugate Gradient) [MOL 90]. Ainsi, le système SCANDAL offre-t-il plusieurs possibilités d'échange de connaissances entre ses composantes avec l'aide d'un agent superviseur.

- **MACIE :** le système MACIE (*Matrix Controlled Inference Engine*) a été développé par S. Gallant [GAL 88, GAL 93, GAL 95, BOZ 95, NIK 97]. Ce système permet de coder des connaissances symboliques dans un réseau, de les raffiner, puis d'en extraire des règles. Le réseau est composé par des unités qui sont identifiées par les labels des concepts qu'elles représentent. L'existence d'une connexion entre deux unités indique une relation de dépendance entre elles.

Pour pouvoir construire un réseau, les dépendances conceptuelles sont prises en compte afin de créer son architecture. Le réseau peut être complètement interconnecté en absence de connaissances suffisantes sur les dépendances. Après la détermination de la structure initiale du réseau, il est entraîné avec des exemples afin d'adapter ses poids (raffiner les connaissances initiales introduites dans le réseau). Une fois que ce type de réseau ne travaille qu'avec des unités à trois états (+1=Vrai, 0=Inconnu, -1=False), l'extraction de connaissances peut être faite facilement d'une manière presque directe.

- **KBCNN :** les réseaux KBCNN (*Knowledge-Based Conceptual Neural Networks*) ont été développés par L. Fu [FU 90, FU 92, FU 93, FU 94]. Ce type de réseau permet l'insertion d'une base de règles et son raffinement par apprentissage, d'une façon similaire aux réseaux KBANN et MACIE. Le réseau KBCNN utilise un algorithme d'apprentissage basé sur la Rétro-Propagation avec une procédure de groupement d'unités (*hidden unit clustering*). Il prévoit l'ajout d'unités au cas où le réseau reste bloqué pendant le processus d'apprentissage, ainsi que la possibilité d'extraire des règles à partir du réseau.

- **SETHEO :** le système SETHEO [LET 92, LAL 96] est un démonstrateur de théorèmes hybride, composé d'un démonstrateur classique (du type ProLog) et d'un module connexionniste servant à optimiser la recherche de preuves. La preuve consiste à trouver un chemin dans un arbre de recherche. Pour éviter l'exploration systématique de tout l'arbre de recherche, un réseau permet de choisir la branche de l'arbre la plus intéressante. Le module connexionniste sert à définir (guider) la stratégie de recherche du moteur d'inférence. Le module connexionniste est composé par un réseau de type PMC entraîné avec la Rétro-Propagation sur une base d'exemple de preuves de théorèmes.

- **WATTS :** le système WATTS (*Wastewater Treatment System*) a été développé par S. Krovvidy et W. Wee [KRO 92, HIL 96b]. Ce système essaye de réduire le temps de résolution d'un problème à travers l'emploi d'un module CBR couplé à un réseau de Hopfield. Le réseau doit déterminer la séquence de traitements à faire subir à des eaux usées afin de les épurer. Le module CBR sert à donner un état initial au réseau qui va permettre d'améliorer la convergence vers la solution du problème.

- **INNATE-QUALMS :** ce système sert à donner le diagnostic de pannes dans une usine de distillation [BEC 91, LAL 96]. Le système est constitué d'un module connexionniste, INNATE, et d'un module symbolique, QUALMS. Le processus principal est contrôlé par le système expert

de diagnostic de pannes. Ce module symbolique appelle un ensemble de réseaux de type PMC afin de obtenir des indications sur les possibles pannes. Une analyse plus fine des réponses des réseaux et de l'état du système permet de confirmer le diagnostic des réseaux ou de proposer une solution alternative.

- **LAM :** le système LAM [MED 94, HIL 96a] est utilisé pour traiter un problème réel, il permet d'aider ses utilisateurs à concevoir divers verres de vitres de gratte-ciel. Le module symbolique (système expert basé sur des règles) aide l'utilisateur dans les tâches de classification des types de verres et du choix du type de verre le plus indiqué étant donné certaines spécifications d'ordre technique et d'ordre pratique. Le système expert appelle un réseau pour réaliser des tâches d'approximation des fonctions. Le réseau est entraîné avec une basé d'exemples de cas pratiques. Il sert à déterminer les paramètres liés aux propriétés sonores et solaires des verres, ce qui était plus difficile à concevoir à travers d'un système basé sur des règles.

- **DDT :** le système DDT (*Device Diagnostic Tool*) a été développé par Gutknecht, Pfeifer et Stolze [GUT 91, LAL 96]. Ce système est destiné à aider au diagnostic de pannes sur un bras de robot. C'est une approche qualifiée de coopérative, car il y a une collaboration entre un système automatique et un système humain. Dans DDT, les deux modules fonctionnent en parallèle pour donner chacun un avis sur le problème à résoudre. Une autre caractéristique intéressante du système DDT est l'amélioration des connaissances avec l'expérience. Les actions et les réponses du système sont stockées dans une base de cas qui est utilisée pour entraîner le réseau lorsque le système n'est pas utilisé.

- **ALVINN :** le système ALVINN (*Autonomous Land Vehicle in a Neural Natwork*) a été développé par D. Pomerleau à l'Université de Carnegie-Mellon - CMU [POM 92, CAU 90]. Une variante de ce système, proposée par Pomerlau [POM 91, HIL 96a], utilise plusieurs réseaux qui ont été entraînés afin de se spécialiser sur certains aspects du contrôle de la conduite autonome (e.g. conduite sur autoroute, conduite sur route secondaire, passage de carrefour, évitement de collision). Le système utilise un module symbolique basé sur des règles pour arbitrer laquelle des réponses des différents sous-modules va être prise en compte. De plus, le module symbolique est aussi responsable pour les tâches de plus haut niveau, tel que la planification de trajectoires.

• **SCRAPS :** le système SCRAPS a été conçu par J. Hendler [HEN 89, LAL 96]. Ce système utilise un raisonnement par analogie. Il est composé par un 'réseau sémantique à propagation de marqueurs', qui est utilisé pour encoder les connaissances dites dures. Il s'utilise d'un module connexionniste de type PMC pour encoder les similarités entre les objets du domaine. Ce réseau associe chacune des ses sorties à un ensemble de traits caractéristiques des objets et ses sorties sont aussi des noeuds du réseau sémantique. De cette façon, le réseau connexionniste va être appelé par le système symbolique afin de mesurer des similarités et ainsi fournir le lien entre l'exemple présenté et les concepts représentés dans le niveau symbolique.

• **RAPTURE :** le système RAPTURE (*Revising Approximate Probabilistic Theories Using Repositories of Examples*) a été développé par J. Mahoney et R. Mooney à l'Université du Texas [MAH 93, MAH 96]. Ce système sert à raffiner des bases de connaissances probabilistes. Il combine des méthodes connexionnistes avec des méthodes symboliques, en utilisant des techniques basées sur les systèmes KBANN et EITHER. La principale caractéristique de ce type de système est l'utilisation des règles et des réseaux avec des indications du degré de certitude (*Certainty Factors*). Le système RAPTURE utilise une version modifiée de la Rétro-Propagation (CFBP) pour raffiner les coefficients de certitude d'une base de règles probabilistes codées dans ce réseau. Il utilise aussi une méthode basée sur le gain d'information (voir la méthode ID3) pour permettre l'ajout de nouvelles règles/unités. Une fois terminé l'apprentissage, les règles peuvent être observées directement dans la structure du réseau.

• **CONSYDERR :** le système CONSYDERR (*CONnectionist SYstem with Dual representation for Evidentiual Robust Reasoning*) a été développé par R. Sun [SUN 91, SUN 95a, SUN 95b, ORS 95]. Ce système est utilisé dans le cadre du raisonnement de sens commun. Pour reproduire cette forme de raisonnement, un système hybride unifié (connexionniste) est réalisé avec deux niveaux : un niveau localiste, qui est une sorte d'implémentation connexionniste d'un système expert ; et un niveau distribué, qui apporte les propriétés purement connexionnistes. Ainsi, le système intègre dans une architecture connexionniste des représentations localistes et distribuées des connaissances, et des processus unifiés de traitement de règles et de similarités.

• **RUBICON :** le système RUBICON a été développé par T. Samad [SAM 92]. Ce système est une implémentation connexionniste d'un système expert basé sur des règles d'ordre 0. C'est un système assez simple qui permet le codage d'une base de règles dans un réseau et son activation. Il possède certaines limitations telles que : le manque d'une méthode pour l'adaptation des poids

(ils sont codés en dur) et l'utilisation uniquement de valeurs discrètes (binaires). Une fois que les connaissances sont codés directement sur le réseau, l'interprétation des règles symboliques peut être faite directement sur les unités.

- **RAAM :** les réseaux RAAM (*Recursive Auto-Associative Memory*) ont été crées par J. Pollack [POL 90, ORS 95]. Ce type de réseau est une implémentation d'un approche hybride unifiée. Il permet d'obtenir des représentations de structures réduites de données complexes sur lesquelles on peut faire des inférences. L'idée derrière ce type d'approche est l'étude de la représentation de structures de données récursives et de taille variable (e.g. tableaux, listes, arbres) par des réseaux connexionnistes.

- **BoltzCONS :** les réseaux BoltzCONS ont été développés par D. Touretzky [TOU 89]. Ils permettent de créer et manipuler dynamiquement dans un réseau des structures comme des piles, des arbres ou plus généralement des graphes orientés. Le but ici est le même des RAAM, c.-à-d. étudier les possibilités d'un traitement symbolique par des réseaux connexionnistes.

- **SHRUTI** : les réseaux SHRUTI ont été développés par L. Shastri et V. Ajjanagadde [AJJ 91, SHA 93, HAY 96]. Ce type d'approche permet de créer des réseaux sémantiques connexionnistes dans lesquels sont codés des larges bases de connaissances. Les réseaux SHRUTI permet l'implémentation d'un mécanisme de *'dynamic variable binding'*.

- **EBNN :** l'algorithme EBNN (*Explanation-Based Neural Network Learning*) a été introduit par T. Mitchell et S. Thrun [MIT 93, MIT 97]. Cet algorithme intègre une approche basée sur les réseaux connexionnistes avec une approche EBL. De la même façon que les méthodes EBL, l'algorithme EBNN utilise la théorie sur un domaine pour guider l'apprentissage d'un concept. Cela est fait à travers des explications et des analyses obtenues à partir des exemples d'apprentissage. La méthode d'apprentissage est similaire à la Rétro-Propagation, où l'on utilise un algorithme qui s'appelle TangentProp.

Le Tableau 4 présente la liste des systèmes hybrides neuro-symboliques, ainsi que leurs propriétés particulières.

3.2.10 Discussion sur les SHNS

Après avoir étudié les différents systèmes SHNS et leurs propriétés, nous nous sommes retrouvés face aux questions suivantes :

- Quel est le meilleur type d'approche à utiliser : l'approche unifiée, l'approche semi-hybride ou l'approche hybride ?
- La représentation connexionniste localiste est-elle plus intéressante que la représentation distribué, ou vice versa ?
- Parmi les systèmes étudiés, en existe-t-il un qui regroupe les propriétés les plus importantes ?
- Comment doit-on coupler les modules d'un système hybride ?

Toutes ces questions sont encore très ouvertes à la discussion, puisque il n'y a pas de consensus sur le type 'idéal' de système hybride. Puisque nous n'avons pas les moyens de répondre précisément aux questions présentées ci-dessus, nous avons donc fait un certain nombre de choix vis-à-vis des propriétés que l'on aimerait avoir dans notre système. Les choix qui ont été faits portent principalement sur les aspects présentés dans la section 2.4.2. Nous essayons aussi de mieux exploiter les complémentarités entre les systèmes symboliques et connexionnistes, tel que nous les avons présenté dans le Tableau 3.

Etant donné les aspects considérés, nous avons choisi de ne pas utiliser de méthodes portant sur des approches du type unifiée. Nous nous sommes plus intéressés aux approches hybrides au sens strict, composées par deux modules (symbolique et connexionniste) et offrant la possibilité de faire des transferts dans les deux sens. Ce type d'approche inclut d'une certaine façon les approches semi-hybrides. Nous sommes de l'avis qu'avec une approche hybride au sens strict on doit pouvoir mieux exploiter la complémentarité entre ces modules, et ainsi pouvoir travailler auusi bien à un niveau symbolique qu'à un niveau sub-symbolique. Il est vrai que l'un des points-clefs de la discussion entre approches unifiées et hybrides se situe autour des capacités de représentation de connaissances et d'inférence sur les données. Nous essayons de garder le meilleur de l'approche symbolique ainsi que meilleur de l'approche connexionniste, sans pour

autant essayer de les unifier, car cela peut amener à l'imposition de restrictions sur l'une ou l'autre approche.

Plusieurs systèmes sont à la source de nos inspirations : SYNHESYS, KBANN, MACIE, KBCNN, DDT et RAPTURE. Les points communs entre ces systèmes sont la possibilité de faire des transferts de connaissances entre un module connexionniste et un module symbolique, ainsi que le principe de fonctionnement orienté vers une acquisition incrémentale de connaissances. Ces systèmes permettent de réaliser un raffinement d'une théorie sur un domaine (*connectionist theory refinement*).

Nous avons aussi fait un choix sur le type de réseau à utiliser, les réseaux de type PMC. Notre intention étant d'utiliser des modèles assez connus (et reconnus) par la communauté des chercheurs qui travaillent dans le domaine des systèmes connexionnistes, nous avons délibérément évité d'utiliser des réseaux plus particuliers tels que les réseaux à coefficients de certitudes, ou des modèles plus spécifiques tels que les modèles employés dans des systèmes comme CONSYDERR, SHRUTI, RAAM ou EBNN. Notre choix des réseaux PMC peut être justifié par les point suivants :

- Notre but principal était d'arriver à la réalisation d'un système hybride fonctionnel et surtout simple à utiliser. Nos sources d'inspiration sont surtout les SHNS qui possèdent ces deux caractéristiques;

- Les réseaux à coefficients de certitude restent un cas à étudier plus en détails. L'utilisation de ce type de réseau a été critique pendant longtemps [MAH 96] et nous pensons que ce type de formalisme doit être encore mieux analysé avant de proposer des SHNS. Nous avons choisi de ne pas nous lancer sur ce type d'approche qui reste ouvert à l'étude dans des recherches futures;

- Certaines réalisations d'approches connexionnistes unifiées semblent trop se rapprocher des représentations de connaissances au niveau symbolique et pour nous il n'est pas sûr de pouvoir garder toujours les avantages et particularités de l'approche connexionniste (e.g.: la capacité de manipulation de variables numériques et le traitement de données incertaines et approximatives). Une approche comme celle des réseaux CONSYDERR semble être assez intéressante, mais peut-être un peu trop complexe à mettre en place. Cette approche est aussi une des voies de recherche qui sont intéressantes à suivre dans des recherches futures;

- Les réseaux récurrents sont sans aucun doute un domaine à exploiter. Malheureusement, à l'heure actuelle il nous semble que l'utilisation de ce type de réseaux dans des systèmes SHNS n'est pas assez développée et il reste encore beaucoup de problèmes à résoudre. Les réseaux récurrents sont très instables et leur apprentissage est un processus difficile à mettre en place. La plupart des résultats pratiques obtenus avec ce type de réseau concernent des problèmes très simples.

De plus, nous avons été amenés au choix des modèles basés sur les PMC afin d'aller vers une nouvelle voie de recherche dans notre équipe, puisque d'autres recherches sur un système hybride à base de prototypes (SYNHESYS) étaient en cours de développement par d'autres chercheurs de notre groupe. Les résultats obtenus dans ces recherches (qui d'ailleurs ont montré un certain nombre de faiblesses de ce type de SHNS à base de prototypes) nous ont dirigés vers cette nouvelle solution. Une de nos idées était justement d'étudier comment porter les idées de base du système SYNHESYS sur un système hybride basé sur des réseaux PMC.

Un autre point que nous avons considéré est lié à la possibilité de faire une acquisition constructive de connaissances, c.-à-d. que le système doit avoir la capacité de faire évoluer ses connaissances. Dans le cas des réseaux connexionnistes, cela veut dire que le réseau doit être aussi capable de faire évoluer sa structure. Ce type de caractéristique est retrouvé dans les systèmes comme SYNHESYS, KBCNN et RAPTURE. On trouve dans la littérature sur les systèmes hybrides plusieurs références qui indiquent la nécessité de développer des nouvelles recherches sur des méthodes qui puissent permettre de faire évoluer la structure des réseaux connexionnistes employés dans les systèmes hybrides [SHA 95, ORS 95, HIL 94b].

De plus, il est intéressant de préciser que notre système, le système INSS, a été conçu à partir des années 1993/1994, et qu'à cette époque les deux systèmes les plus connus et les plus accessibles à l'étude étaient le système SYNHESYS et le système KBANN. Donc, nous avons été influencés principalement par ces deux systèmes. Nous allons les décrire plus en détails dans les sections suivantes.

Nom du Système	*Type de Intégration*	*Degré de Couplage*	*Mode de Intégration*	*Transfert de Connaissances*	*Type de Représentation*	*Codage des Connaissances*	*Méthode de Raisonnement*	*Méthode de Apprentissage*
SYNHESYS [GIA 92]	Hybride	Forte	Co-Trait.	S ⇒ C : $R0^+$ S ⇐ C : $R0^+$	Localiste	MS : $R0^+$ MC: Prototype	Avant/Arrière Propagation	Continue Supervisée
ProBis [MAL 96]	Hybride	Moyen	Co-Trait.	S ⇒ C : Cas S ⇐ C : Cas	Localiste	MS : CBR MC: Prototype	K-PPV Propagation	Continue Supervisée
NESSY3L [ORS 95]	Hybride	Moyen/Forte(1)	Chaîné/Méta(1)	S ⇒ C: $R0^+$ (1)	Combiné	MS: RFloues(1) MC:PMC/Prot(1)	Ch.Avant Propagation	Non-Continue Supervisée
INSS [OSO 95]	Hybride	Moyen	Co-Trait.	S ⇒ C : $R0^+$ S ⇐ C: $R0^+$ (1)	Combiné	MS : Système Expert (CLIPS) MC : PMC (3)	Avant/Arrière(1) Propagation	Continue Supervisée
KBANN [TOW 91]	Semi-Hybride	Moyen	[pas de MS]	S ⇒ C : R0 S ⇐ C : R0 (2)	Localiste	MS : --- MC : PMC	Propagation	Non-Continue Supervisée
SCANDAL [STU 96]	Hybride	Moyen	Co-Trait.	S ⇒ C : $R0^+$ (1) S ⇐ C : $R0^+$ (1)	Combiné	MS : Règles, arbres de décision, et autres. MC : PMC	Inférence Symbolique et Propagation	Continue (1) Supervisée
MACIE [GAL 88]	Semi-Hybride	Moyen	[pas de MS]	S⇒C: Dépendances S ⇐ C : R0	Localiste	MS : --- MC : "Pocket Algorithm"	Propagation	Non-Continue Supervisée
KBCNN [FU 93]	Semi-Hybride	Moyen	[pas de MS]	S ⇒ C : R0 S ⇐ C : R0	Localiste	MS : --- MC:PMC (3)	Propagation	Continue Supervisée
SETHEO [SUT 90]	Hybride	Faible	Sous-Trait.	S ⇐ C : Paramètres pour l'optimisation de la preuve	Combiné	MS:Démonstrateur Automatique MC : PMC	Preuve Autom. Propagation	Non-Continue Supervisée
WATTS [KRO 92]	Hybride	Faible	Trait. Chaîné	S ⇒ C : Cas pré-analysés	Combiné	MS : CBR MC : Réseaux de Hopfield	Propagation	Non-Continue Supervisée
INNATE-QUALMS [BEC 91]	Hybride	Faible	Sous-Trait.	S ⇐ C : Diagnostic	Combiné	MS : Système Expert MC : PMC	Inférence Symbolique et Propagation	Non-Continue Supervisée
LAM [MED 92]	Hybride	Faible	Sous-Trait.	S⇒C:Cas à traiter S⇐C: Prediction	Combiné	MS : R0 MC : PMC	Ch. Avant Propagation	Non-Continue Supervisée
DDT [GUT 90]	Hybride	Faible	Co-Trait.	Pas d'intéraction	Combiné	MS : Système Expert MC : PMC	Inférence Symbolique et Propagation	Continue (+/-) Supervisée
ALVINN [POM 91]	Hybride	Faible	Méta-Trait.	S⇒C: Problème S⇐C: Réponse	Combiné	MS:Planification et Contrôle MC : PMC	Inférence Symbolique et Propagation	Non-Continue Supervisée
SCRAPS [HEN 89]	Hybride	Fort	Sous-Trait.	Partage de données	Combiné	MS : Réseau Sémantique MC : PMC	Inférence Symbolique et Propagation	Non-Continue Supervisée

RAPTURE [MAH 93]	Hybride	Faible	[pas de MS]	S ⇒ C : R0 avec CF	Localiste	MC : CFBP (ajout d'unités basé sur ID3)	Propagation	Continue Supervisée
CONSYDERR [SUN 91]	Unifié	---	[pas de MS]	---	Combiné	MC: Approche Spécifique	Propagation	Continue Supervisée
RUBICON [SAM 92]	Unifié	---	[pas de MS]	---	Combiné	MC: Approche Spécifique	Propagation	Sans-Apprent. (codé directement)
RAAM [POL 90]	Unifié	---	[pas de MS]	---	Distribué	MC : Réseau RAAM	Propagation (unification)	Supervisée
BoltzCONS [TOU 90]	Unifié	---	[pas de MS]	---	Distribué	MC: Approche Spécifique	Propagation (manip. de structures symb. complexes)	Supervisée
SHRUTI [SHA 93]	Unifié	---	[pas de MS]	---	Localiste	MC: Approche Spécifique	Propagation	Codage
EBNN [MIT 93]	Semi-Hybride	Faible	[pas de MS]	---	Distribué	MC: Approche Spécifique (TangentProp)	Propagation (Clauses de Horn représentés par RNAs)	Supervisée

(1) = Améliorations et extensions possibles prevues; (2) = Règles d'ordre 0 et règles du type M_Parmi_N (Subset: M_of_N); (3) = PMC avec Rétro-Prop. modifié
R0 = Règles d'ordre 0; $R0^+$ = Règles d'ordre 0^+; MS = Module Symbolique; MC = Module Connexionniste; CF = Degré de certitude (*Certainty Factor*)

Tableau 4 - Classification de plusieurs SHNS connus

3.3 REALISATIONS DE SYSTEMES HYBRIDES

Nous allons décrire dans cette section les deux systèmes qui ont le plus contribué à la conception du système INSS : le système SYNHESYS et les réseaux KBANN.

3.3.1 Système SYNHESYS

Le système SYNHESYS a été développé au laboratoire LEIBNIZ (ancien lab. LIFIA) par A. Giacometti [GIA 92]. Il est en fait un noyau d'un système expert hybride, qui a été utilisé dans la conception de plusieurs applications, notamment SHADE (aide au diagnostic en électromyographie) et SATAN-hybride (aide à la manoeuvre navale). Ce système a été développé dans le cadre plus général d'une étude psycho-mimétique de l'expertise. Son but était d'étudier et mettre en pratique une approche basée sur la démarche experte humaine afin d'obtenir de meilleurs systèmes experts.

Giacometti défend l'hypothèse suivante : " un expert n'est pas uniquement un sujet qui raisonne à partir de règles ou un sujet qui a mémorisé et est capable de discriminer des milliers de cas particuliers. C'est un être hybride dans le sens où il utilise des connaissances, des modes de pensée et d'apprentissage de natures différentes. Sa compétence ne peut pas être réduite à un mode de pensée et/ou d'apprentissage particulier. Elle résulte en partie de l'interaction entre des modes de pensée et d'apprentissage différents, mais pouvant être complémentaires. ". C'est en se basant sur cette hypothèse que le système SYNHESYS a été conçu.

Nous présentons ici une description très simplifiée du fonctionnement de ce système. Notre but est de donner un aperçu général du système de façon à mettre en valeur ses propriétés principales qui nous ont influencés dans nos recherches et travaux. Le système SYNHESYS se compose d'une architecture à deux modules principaux : le module symbolique et le module connexionniste. Ces deux modules peuvent être utilisés en parallèle dans un mode de coopération

ou bien on peut réaliser des transferts de connaissances entre les deux. La Figure 19 montre l'architecture du système, en mettant également en évidence le fait que les deux modules ont exactement les mêmes entrées, provenant d'une situation (ensemble de variables avec leurs attributs respectifs), et qu'ils produisent chacun une décision (réponse du système) qui peut être ensuite analysée par l'expert. Dans les sections suivantes, nous allons d'abord décrire les modules symbolique et connexionniste, pour pouvoir ensuite présenter les méthodes de transferts de connaissances existantes et les types de processus de coopération réalisés entre les modules.

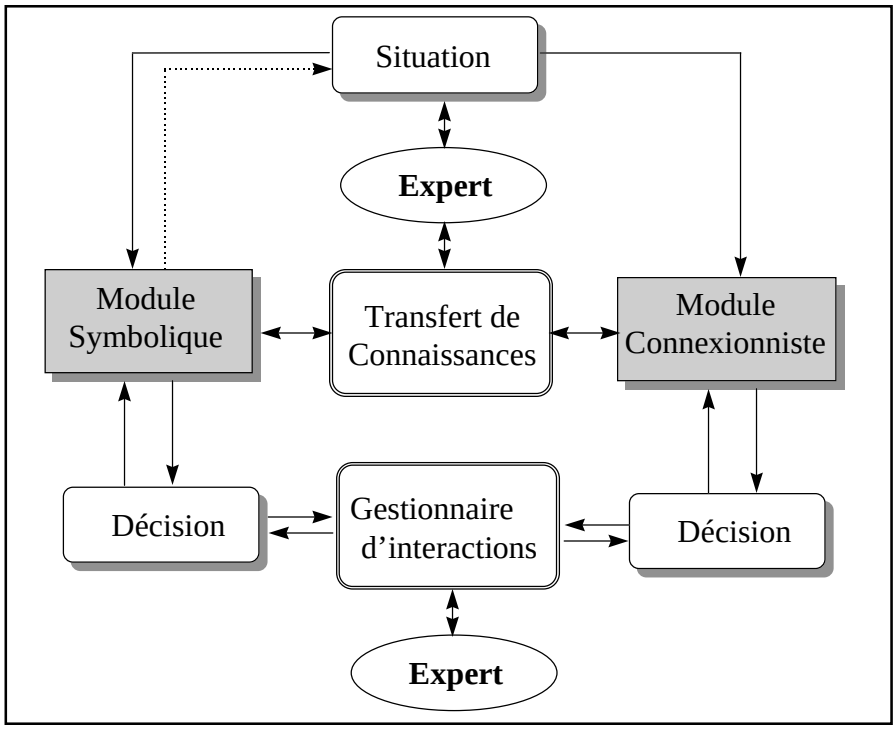


Figure 19 - Architecture du système SYNHESYS

3.3.1.1 Module Symbolique du Système SYNHESYS

Le module symbolique du système SYNHESYS travaille sur des règles propositionnelles d'ordre 0^+. Ces règles représentent les connaissances théoriques du système sur le problème en question. La Figure 20 montre un exemple d'une base de règles utilisé par le système SYNHESYS.

```
BASE_REGLES : exemple
Auteur : Osorio
Date   : 22/01/97

/*-----------------------------------------------------------*/

ENTREES
y [0,20]
x [0,30]

ACTIONS
act1 act2 act3

REGLE r1   Si y dans  [0,10] alors y_faible        FINREGLE r1
REGLE r2   Si y dans [10,20] alors y_fort          FINREGLE r2
REGLE r3   Si x dans  [0,10] alors x_faible        FINREGLE r3
REGLE r4   Si x dans [10,20] alors x_moyen         FINREGLE r4
REGLE r5   Si x dans [20,30] alors x_fort          FINREGLE r5
REGLE r6   Si y_faible alors act1                  FINREGLE r6
REGLE r7   Si y dans [14,20] et x_moyen alors act2 FINREGLE r7
REGLE r8   Si y_fort et x_faible alors act3        FINREGLE r8
REGLE r9   Si y_fort et x_fort   alors act3        FINREGLE r9

/*-----------------------------------------------------------*/

FIN
```

Figure 20 - Exemple typique d'une base de règles symboliques de SYNHESYS

Le module symbolique possède un moteur d'inférence avec un chaînage avant et aussi un chaînage arrière. Le chaînage avant permet d'établir les décisions (actions/réponses) à partir des situations (cas particuliers) présentées au système. Inversement, le chaînage arrière permet de savoir quelles sont les prémisses nécessaires au déclenchement d'une action spécifique. Le chaînage arrière prend une grande importance par rapport au processus de coopération entre les deux modules. Il permet de valider les réponses obtenues par le module connexionniste (voir section 3.3.1.4).

3.3.1.2 Module Connexionniste du Système SYNHESYS

Le module connexionniste réalisé utilise des réseaux à base de prototypes, appelés ARN2 [GIA 92, ORS 95, MAL 96b]. Ce module permet d'exploiter deux types distincts de réseaux à prototypes : des réseaux à *hyper-sphères* et des réseaux à *hyper-rectangles*. Les réseaux à hyper-sphères sont illustrés par l'exemple de la Figure 11 (voir section 2.3.1.2), pour le cas d'un réseau à deux entrées seulement. La Figure 21 montre un exemple d'un réseau à hyper-rectangles, qui

représente lui aussi le cas d'un réseau à deux entrées seulement. L'utilisateur du système peut choisir le type de réseau (un seul à la fois) qui va être employé dans le module connexionniste. Il faut savoir que seuls les réseaux à hyper-rectangles permettent l'extraction de règles symboliques.

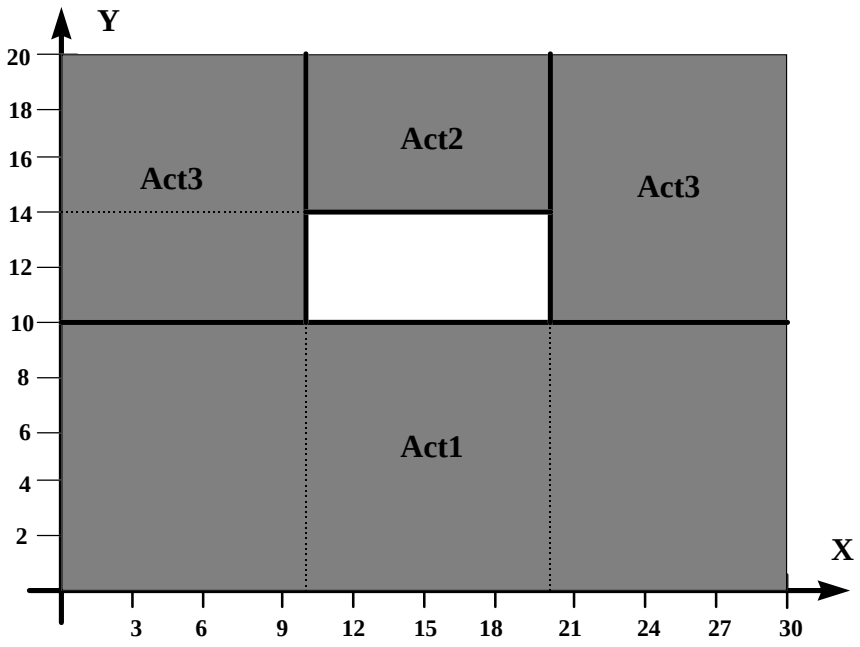


Figure 21 - Prototypes dans un réseaux à hyper-rectangles avec deux entrées

Le MC est composé par un réseau à trois couches : une couche d'entrée, une couche cachée de prototypes et une couche de sortie regroupant les décisions (classes). La dynamique du processus d'activation est déclenchée par une situation qui est présenté à l'entrée du réseau. L'activation de la couche d'entrée va entraîner le calcul des similitudes entre le vecteur d'entrée et les prototypes stockés. Un processus de compétition (WTA - *Winner Take All*) est ainsi déclenché et permet de sélectionner le prototype le plus 'proche' du vecteur d'entrée présenté au réseau. Le prototype gagnant étant associé à une des unités de sortie fait que cette unité est activé à son tour. Dans ce type de réseau, un ou plusieurs prototypes sont associés à une des unités de sortie, et chacune de ces unités représente une décision. Ce modèle est du type incrémental puisqu'on peut ajouter des nouveaux prototypes dans la couche cachée pendant le processus d'apprentissage. La Figure 22 présente un schéma de l'architecture de ce type de réseau.

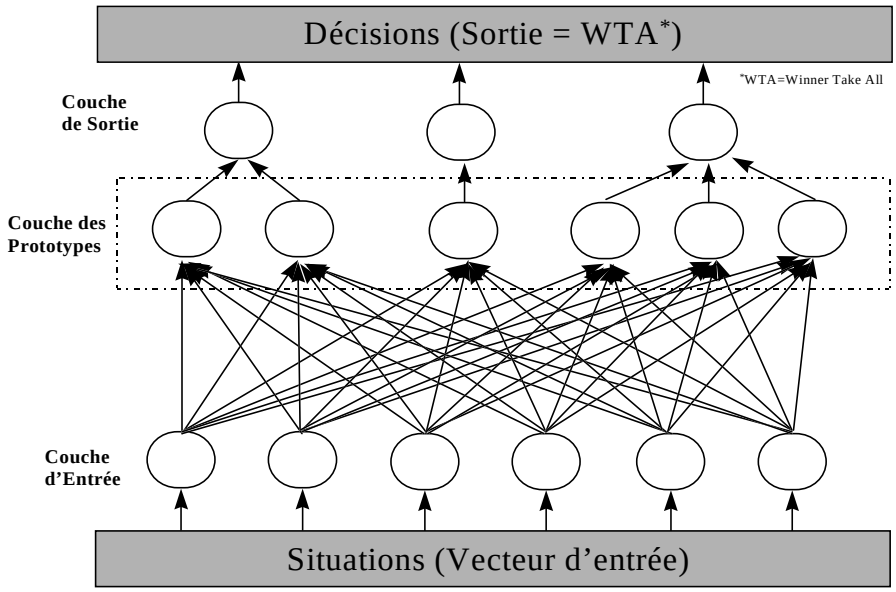


Figure 22 - Architecture multicouches du réseau à prototypes utilisé dans SYNHESYS

L'apprentissage dans un réseau à prototype consiste à réaliser une des tâches suivantes :

- S'il n'y a pas de prototype assez proche de l'exemple présenté, il va falloir créer un nouveau prototype qui va représenter cet exemple, ou il va falloir modifier un des prototype existants afin de l'approcher de l'exemple. Ce cas de figure se produit quand l'exemple se trouve en dehors de la région d'influence de tous les prototypes du réseau, et par conséquence aucune des sorties du réseau n'est activée. En particulier, les réseaux de type ARN2 font la création d'un nouveau prototype dans un cas comme celui-ci;

- Si l'exemple tombe dans la région d'influence d'un prototype qui n'appartient pas à la même classe que lui (la réponse du réseau n'est pas la bonne), il va falloir passer par une phase d'adaptation. Dans ce cas, le prototype activé doit être adapté afin de changer sa région d'influence. L'adaptation peut être obtenue par la réduction de la région d'influence du prototype (on appelle cette approche, qui est la technique employée par les réseaux ARN2, *différentiation*). Une autre possibilité d'adaptation est le déplacement du prototype dans une direction opposé à celle où se trouve l'exemple;

- Si l'exemple tombe dans la région d'influence d'un prototype qui représente la même classe que lui (la réponse du réseau est la bonne), on peut alors réaliser une *accommodation*

(déplacement) de la région d'influence du prototype, de façon à mieux prendre en compte l'exemple présenté. L'idée étant de déplacer les prototypes ver le 'centre de gravité' des exemples qu'il représente. Les réseaux ARN2 utilisent ce type d'approche.

La base d'exemples est présentée au réseau jusqu'à ce que le réseau ait assez bien appris le problème, c.-à-d. que l'erreur obtenue dans ses réponses soit arrivée à un niveau satisfaisant. Puisque le réseau est du type incrémental, il est toujours possible d'ajouter des nouvelles unités afin d'améliorer sa performance jusqu'au niveau envisagé. Par contre, il faut faire très attention dans ce type de modèle à bien régler certains paramètres d'apprentissage afin de ne pas ajouter trop d'unités, ce qui implique usuellement une perte de la capacité de généralisation du système. Le cas extrême est celui où le MC va représenter chaque exemple de la base d'apprentissage par un prototype.

Avec ce type méthode, il faut aussi faire attention aux cas contradictoires (deux cas/situations identiques associés à décisions/actions différentes) qui peuvent gêner le processus d'apprentissage. Un autre problème à traiter est l'oubli des cas, puisque tout changement subi par un prototype (différentiation ou accommodation) peut entraîner l'exclusion indésirable (ou même l'inclusion) d'exemples dans le rayon d'influence de celui-ci. La solution dans ce cas est la réalisation de plusieurs présentations de la base d'exemples afin de bien établir les rayons d'influence des prototypes. Ceci étant dit, on s'aperçoit que le module connexionniste de SYNHESYS est très sensible à certains paramètres, tels que : le rayon initial d'influence des prototypes, l'ordre de présentation des exemples, les pas de déplacement et de réduction du rayon d'influence des prototypes, et à la quantité de 'bruit' (cas mal classés) présent dans la base d'apprentissage.

3.3.1.3 Transfert de Connaissances dans le Système SYNHESYS

Le système SYNHESYS met en oeuvre deux types de transferts de connaissances entre les modules symbolique et connexionniste :

• MS ⇒ MC : transfert par la compilation d'un ensemble de règles dans un réseau. Ce type de méthode ne peut être appliqué que dans le cas des réseaux à hyper-rectangles. Il est aussi possible de faire le transfert par génération aléatoire d'un ensemble d'exemples (classés avec l'aide de la base de règles) et de réaliser son apprentissage postérieurement par un réseau à base d'hyper-sphères ou d'hyper-rectangles.

• MC ⇒ MS : transfert par l'extraction de règles d'un réseau. Ce type de méthode de transfert de connaissances n'est disponible que pour les réseaux à base d'hyper-rectangles.

Les algorithmes de compilation et d'extraction de règles ont été présentés de manière formalisée par Giacometti [GIA 92]. Nous allons nous contenter ici d'en décrire ces processus de manière intuitive et très simplifiée. Le fait de nous concentrer sur ces deux types d'algorithmes de transfert de connaissances implique que nous allons décrire les processus qui interviennent exclusivement sur les réseaux à base de hyper-rectangles.

Prenons comme exemples les Figure 20 et Figure 21, qui représentent respectivement une base de règles et les prototypes d'un réseaux à hyper-rectangles. Les règles 'r1' et 'r6' de la Figure 20 peuvent être compilés dans une représentation connexionniste à prototypes, tel que l'on montre par la région indiqué comme 'Act1' dans la Figure 21. Inversement, on peut facilement extraire des règles d'ordre 0^+ à partir des prototypes à hyper-rectangles, comme par exemple, la région indiqué par 'Act2' dans la Figure 21 qui correspond à la règle suivante : *Si* X dans [10,20] et Y dans [14,20] *Alors* Act2. On peut lire cette règle autrement : Si x appartient à l'intervalle [10,20] et y appartient à l'intervalle [14,20] alors l'action Act2 est effectuée, ou bien, on considère que Act2 est actif/vrai. Les exemples à deux variables sont plus simples à interpréter, mais rien ne nous empêche de passer à plus de deux dimensions, avec des prototypes basé sur des hyper-rectangles en N-dimensions. Les processus de compilation et d'extraction marchent aussi bien avec deux variables qu'avec plus.

Dans le cas du transfert de connaissances du module symbolique vers le module connexionniste par génération d'exemples, nous sommes de l'avis que ce type de méthode a

quand même un certain nombre d'inconvénients. Premièrement, le fait de créer des exemples aléatoirement ne permet pas de garantir que toutes les règles vont être bien représentés dans la base d'exemples. Deuxièmement, le fait de passer par une phase d'apprentissage à partir des exemples implique un coût computationel additionnel et en plus il ne nous assure pas que le réseau final ait bien appris tous les connaissances codées dans cette base. En conclusion, cette méthode n'est pas capable d'assurer un transfert complet et correct des connaissances entre les deux modules.

3.3.1.4 Coopération entre modules dans le Système SYNHESYS

En plus des méthodes de transfert de connaissances, les modules symbolique et connexionniste peuvent avoir une coopération par interaction à travers du module gestionnaire d'interactions. On peut avoir différents types d'interactions entre les modules selon l'ordre utilisé pour les consulter, la façon d'interpréter ses résultats, et les différentes priorités données aux réponses des deux modules. Il est aussi possible d'avoir des interactions entre l'expert et les modules MS et MC. En particulier, nous nous sommes plus intéressés à l'un de ces modes d'interactions lié au processus de validation des réponses et à la possibilité d'améliorer les connaissances acquises par le système. Nous allons décrire ci-dessous ce mode particulier de fonctionnement de SYNHESYS.

Une fois que les deux modules peuvent être utilisés en parallèle, on peut avoir des réponses contradictoires à la sortie des deux modules. C'est justement à travers la comparaison des réponses des modules MS et MC qu'on va pouvoir mettre en pratique une méthodologie d'analyse des réponses, d'interaction avec l'expert et de raffinement des connaissances acquises par le système. Dans ce mode d'opération, le système SYNHESYS va être considéré comme un *outil d'acquisition constructive et interactive de connaissances.*

L'acquisition interactive de connaissances est faite à travers la coopération entre les différents composantes du système et aussi avec l'aide d'un expert. Ce processus est brièvement décrit ci-dessous :

1. *Activation du MC* : le module connexionniste est activé en premier. Ce module va fournir une décision par sa sortie Dc (Décision Connexionniste). Selon le type de situation proposée à l'entrée du MC, il peut arriver qu'il ne donne aucune réponse. Cela se produit si la situation présentée constitue un cas qui n'est pas suffisamment proche des prototypes connus et ainsi le système ne pourra pas prendre une décision. Dans ce cas, on va être obligé de consulter le MS (on passe à l'étape 4).

2. *Validation de la Dc par le MS* : après l'activation du MC, on va prendre sa sortie Dc et faire l'activation du module symbolique par chaînage arrière. Le chaînage arrière sert à vérifier si la réponse donnée par le MC correspond à une des réponses possibles obtenue à partir des règles du MS. Si l'activation du MS est en accord avec le MC alors la réponse finale du système est communique à l'expert (on passe à l'étape 5). Si l'activation du MS est en contradiction avec le résultat obtenu par le MC alors on doit résoudre le conflit entre les deux modules.

3. *Conflit entre les deux modules* : les réponses du MC et du MS sont passés à l'expert afin qu'il puisse juger laquelle des deux est la réponse correcte. Si la sortie Dc est considérée correcte, alors il va falloir changer la base de règles. Si la sortie Ds est considérée correcte, alors il va falloir adapter le MC (adaptation ou élimination du prototype responsable de la mauvaise réponse). Après cet étape, le système est prêt pour une nouvelle consultation.

4. *MC n'est pas arrivé à une décision* : le MS est activé par chaînage avant et obtient ainsi une réponse Ds (Décision Symbolique). La réponse Ds donnée par le MS doit être validé par l'expert. Le MS est capable de produire une explication de son raisonnement et si l'expert trouve cela correct, il va falloir adapter le MC afin de prendre en compte cet exemple. Dans ce cas, le réseau va être modifié de façon à donner la réponse Ds pour cette situation proposée, ce qui pourra donner l'origine à un nouveau prototype.

5. *MC et MS sont en accord* : si les deux modules ont données des réponses qui sont en accord, alors le système présente à l'expert la solution finale. L'expert va juger cette solution, et

s'il trouve qu'elle est convenable le système reste inaltéré. Si la réponse donnée n'est pas correcte, il va falloir réviser les connaissances des deux modules.

La Figure 23 montre un schéma simplifié du processus d'acquisition interactive de connaissances décrit ci-dessus et qui est réalisé par le système SYNHESYS. Nous trouvons ce mode de coopération entre les deux modules et l'expert très intéressant puisqu'il permet d'améliorer les connaissances symboliques et connexionnistes. Les interactions entre modules permettent de mettre en relation les connaissances théoriques représentées dans le MS avec les connaissances empiriques représentées dans le MC. Cette approche a certains points en commun avec le système DDT [GUT 90], mais le système SYNHESYS va encore plus loin et nous offre la possibilité de faire des transferts de connaissances entre les deux modules.

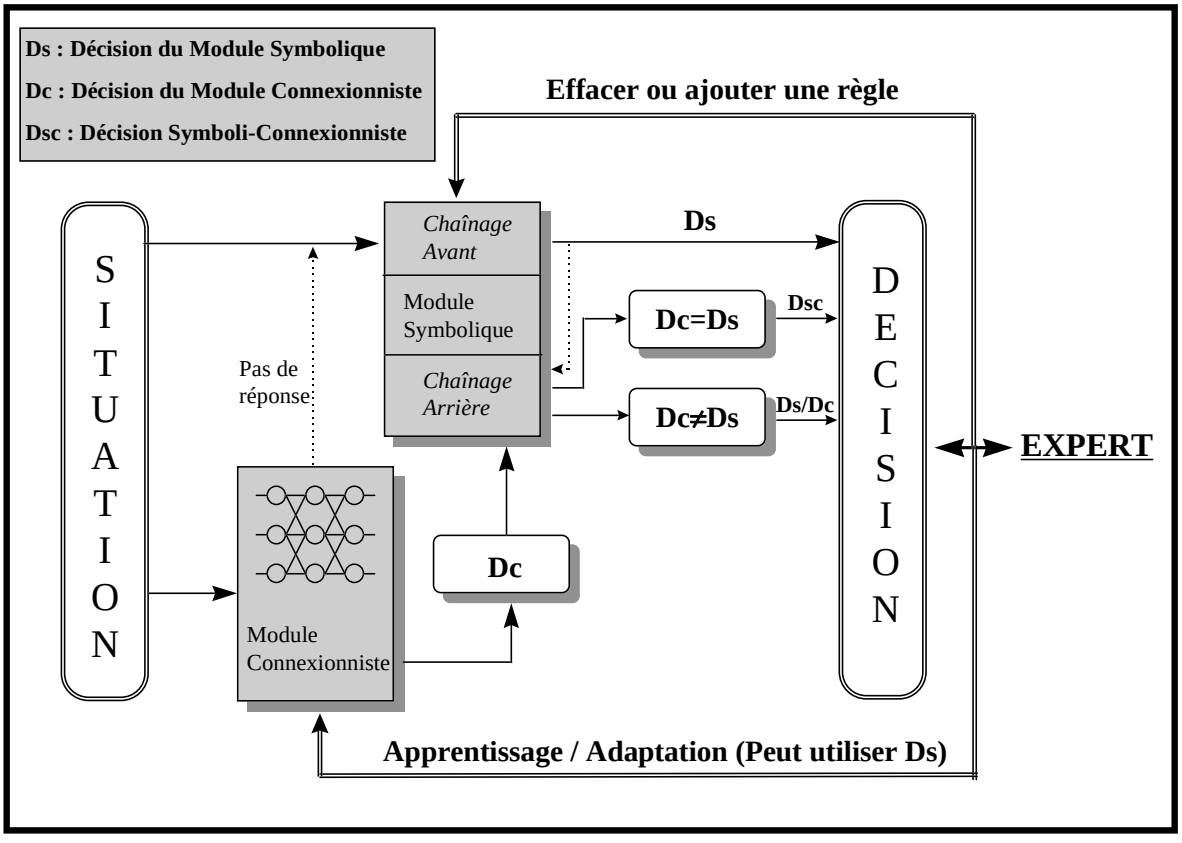


Figure 23 - L'acquisition interactive de connaissances fait par SYNHESYS

3.3.1.5 Discussion sur le modèle de SHNS proposé par SYNHESYS

Le système SYNHESYS est un exemple très réussi d'implémentation d'un noyau de système hybride neuro-symbolique. Ce système exploite bien les méthodes d'intégration entre MS et MC, et permet le transfert de connaissances dans les deux sens entre ses modules. De plus, il montre bien l'importance de l'interaction entre MS, MC et l'expert afin d'améliorer ses connaissances. C'est un système qui possède des caractéristiques que l'on aimerait retrouver dans d'autres systèmes SHNS :

- Le couplage à la fois moyen (avec des transferts dans les deux sens) mais aussi fort (utilisation de liens internes entre les deux modules) [ORS 95, ORS 94]. Le couplage fort permet d'optimiser le processus d'inférence dans le système;

- Les processus d'interaction et de coopération entre ses modules lui donnent un mode d'intégration par co-traitement;

- La possibilité d'augmenter le réseau afin d'acquérir des nouvelles connaissances (modification structurale);

- La possibilité de traiter directement des valeurs continues (variables quantitatives) sans restrictions et sans avoir besoin d'utiliser des techniques de pre-traitement des données (discrétisation). Le système travaille avec des règles d'ordre 0^+ qui peuvent être compilées dans son module connexionniste.

Cependant, le système SYNHESYS présente aussi un certain nombre de limitations. Bien que le module connexionniste avec ses réseaux à prototypes soit un modèle assez intéressant, il possède des propriétés qui peuvent être assez gênantes selon le type de problème traité. La plupart des problèmes (et pas mal d'avantages) du système SYNHESYS sont attribués au choix de son modèle de réseau connexionniste à prototypes, à savoir :

- Les prototypes à hyper-rectangles ne sont pas bien adaptés à la représentation des cas qui constituent des frontières 'diagonales' (non-parallèles aux axes de l'espace d'entrée) ou trop irrégulières [GIA 92] (c'est aussi un problème commun aux arbres de décision[QUI 93]);

- Les connaissances sont représentées par des unités localistes que ne prennent pas en considération des relations existantes *entre* les différentes prototypes. Ce type de représentation connexionniste pose des limitations par rapport à sa capacité générale de représentation de connaissances et complexifie l'apprentissage de règles telles que : Si A=B Alors ActionX ou Si A>B Alors ActionY;

- Les réseaux à hyper-rectangles utilisés par SYNHESYS peuvent présenter des problèmes de non-convergence [ORS 95] et ils manquent d'une preuve plus formelle de leurs propriétés de convergence vers une solution du problème;

- Le processus d'apprentissage passe par des phases de création, accommodation et différentiation des prototypes. Ces processus sont à la fois très sensibles aux paramètres de configuration de l'algorithme d'apprentissage, mais aussi responsables de problèmes comme l'oubli de cas ou la création excessive de prototypes;

- Les réseaux à prototypes sont basés sur une mesure de similarité, qui n'est pas toujours adaptée aux différents problèmes qui sont posés. Le système SYNHESYS utilise un type de réseau et une mesure de similarité qui sont bien adaptés aux variables continues (données quantitatives) mais qui montre des limitations assez fortes par rapport au traitement de variables discrètes (données qualitatives) [ORS 95]. Compte tenu du fait que la majorité des applications des systèmes experts utilisent beaucoup de données qualitatives, le système SYNHESYS a un fort handicap en ce qui concerne le traitement de ce type de problèmes;

- Les réseaux à seulement trois niveaux imposent des restrictions pour la représentation de concepts intermédiaires. Une base de règles symboliques peut en réalité être constituée par plusieurs niveaux intermédiaires. Il faut alors créer des structures internes parfois un peu complexes de connexion entre le module symbolique et le module connexionniste afin de pouvoir préserver les liens existants entre les concepts intermédiaires définis dans une base de règles et les structures connexionnistes de représentation de connaissances;

- Les règles extraites à partir des réseaux sont parfois trop nombreuses. On peut aussi se poser des questions sur *la qualité* et sur *l'utilité* de ces règles après leur extraction, à partir du moment où elles sont fournies à l'expert afin d'être analysées et utilisées dans la pratique.

On trouve une discussion plus détaillée à propos des propriétés et des limitations du système SYNHESYS dans les études publiés par B. Orsier [ORS 95, ORS 94]. Nous n'avons pas l'intention de revenir sur tous ces détails, car notre intérêt était plutôt de mettre en évidence certains facteurs qui nous ont influencés dans le développement de notre système hybride INSS. Dans notre système, on a retenu les points positifs de SYNHESYS et nous les avons combinés avec une autre approche hybride, les réseaux KBANN, afin d'obtenir un SHNS plus performant.

3.3.2 Réseaux KBANN

Les réseaux KBANN (*Knowledge-Based Artificial Neural Networks*) ont été développés dans le MLRG-UWM (*Machine Learning Research Group - University of Wisconsin-Madison*) par G. Towell et J. Shavlik [TOW 91, TOW 93, TOW 94, SHA 94, SHA 95]. Les réseaux KBANN sont une approche semi-hybride qui permet la compilation, le raffinement et l'extraction de règles symboliques. Cette approche permet une intégration de connaissances théoriques et empiriques d'une façon simple et avec des résultats pratiques très bons.

Les réseaux KBANN sont le résultat de l'implémentation de trois algorithmes : le premier algorithme est responsable de la traduction de règles symboliques approximativement correctes vers un RNA de type PMC; le deuxième algorithme fait le raffinement des connaissances du réseau par apprentissage d'une base d'exemples à travers l'utilisation de la Rétro-Propagation; le troisième algorithme va réaliser l'extraction des règles raffinées à partir du réseau qui a été entraîné. De cette façon, ces trois modules permettent de faire des transferts de connaissances entre un module symbolique et le module connexionniste. On peut éventuellement réintroduire les règles raffinées dans le réseau. La Figure 24 présente un schéma des trois modules principaux qui constituent ce type d'approche.

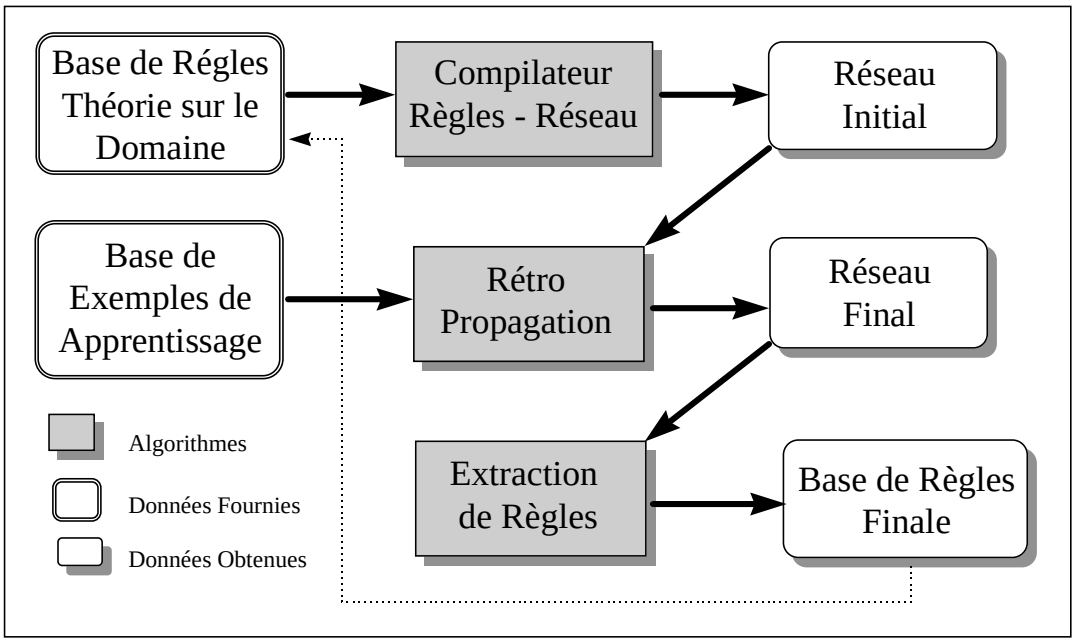


Figure 24 - Schéma de l'intégration neuro-symbolique utilisé dans les réseaux KBANN

Selon Towell [TOW 91], les réseaux KBANN peuvent soit éliminer complètement soit réduire significativement une grande partie des problèmes inhérents aux réseaux de neurones classiques et aux méthodes d'apprentissage empiriques. Il a montré que les réseaux KBANN dépassent nettement les performances des approches symboliques pures utilisés par des systèmes comme EITHER, ID3, COBWEB ou Labyrinth-K, et aussi des approches connexionnistes pures comme c'est le cas des réseaux classiques de type PMC à Rétro-Propagation. Cette approche a été testée sur différentes applications, tel que des problèmes réels du domaine de la biologie moléculaire et dans la modélisation du développement du raisonnement géométrique chez l'enfant. Les résultats obtenus avec ce système sont très encourageants.

Nous allons présenter dans la section suivante la compilation de règles en réseau de neurones, puis la méthode de raffinement de connaissances par apprentissage qui est utilisée dans les réseaux KBANN. Après, nous allons présenter les méthodes d'extraction de règles, pour finir par une discussion sur les propriétés et problèmes des réseaux KBANN.

3.3.2.1 Compilation de Règles dans KBANN

Le transfert de connaissances théoriques (règles symboliques) vers le module connexionniste est réalisé dans les réseaux KBANN par un algorithme de compilation de règles approximativement correctes. Le Tableau 5 montre les relations entre une base de connaissances et les réseaux de type KBANN. La compilation de règles nous permet d'obtenir un réseau qui donne exactement les mêmes réponses que nous pouvons obtenir à partir des règles qui ont servi à le créer. Cette façon de créer les réseaux permet de simplifier beaucoup le processus de spécification et d'apprentissage d'un réseau. L'architecture initiale du réseau est obtenue de façon automatique et nous n'avons plus de soucis par rapport à la détermination du nombre de couches, de neurones et d'interconnexions du réseau. De plus, les règles donnent au réseau des connaissances de départ. Ces connaissances sont codées par des liens qui représentent des relations de dépendance entre les différents attributs d'entrée et les réponses qu'on doit obtenir à la sortie. Cette configuration initiale introduite dans le réseau est probablement plus proche de la configuration finale qu'on veut obtenir qu'une autre configuration aléatoire quelconque des poids. Ainsi, le problème du choix aléatoire des poids initiaux d'un réseau et de ses conséquences (souvent négatives) sur l'apprentissage peut être contrôlé et minimisé. Towell a montré dans ses recherches que ce type d'approche permet de réduire le temps d'apprentissage et le nombre d'exemples nécessaires à un réseau pour bien apprendre à résoudre un problème.

Base de Connaissances	**Réseau de Neurones**
Conclusion Finales	Unités de Sortie
Conclusions intermédiaires	Unités de la couche cachée
Faits (prémisses de base)	Unités d'entrée
Dépendances	Poids des connexions

Tableau 5 - Correspondances entre les bases de connaissances et les RNAs

Les règles utilisées par KBANN sont exprimés par des clauses de Horn avec une notation que nous décrivons dans la Figure 25. Deux restrictions sont imposées sur l'ensemble de règles. Premièrement, les règles doivent exprimer des propositions et ne doivent pas contenir des variables. Deuxièmement, les règles doivent être acycliques, c.-à-d. qu'une règle ne doit pas avoir

un conséquent qui intervient dans un de ses antécédents (directement ou indirectement). Donc, les règles utilisées sont des règles d'ordre 0 codées de façon hiérarchique. Les réseaux KBANN travaillent plutôt avec des valeurs discrètes, et les valeurs continues impliquent la réalisation d'un prétraitements des données (une sorte de discrétisation). Les travaux développés par Towell restent imprécis en ce que concerne le traitement de valeurs continues et son application dans la pratique.

<Conséquent> :- <Antécédent> , <Antécédent> , ... <Antécédent> .

<Conséquent> = C'est la partie qui représente la conclusion de la règle.

<Antécédent> = C'est la partie qui représente les prémisses. Un antécédent peut être composé par un **<opérateur>** suivi d'une formule atomique.

<Opérateur> = Il s'applique sur une formule atomique. L'opérateur le plus utilisé est le 'Not', mais les réseaux KBANN acceptent aussi des opérateurs 'Greater-Than', 'Less-Than' et 'N-True'.

Exemple:

Les règles symboliques - Si (A et B) ou (A et Non(D)) Alors C

Si A et N-parmi-M(2,B,C,D,E) Alors F

sont décrites par - C :- A , B.

C :- A , Not D.

F :- A, N-True(2,B,C,D,E).

Figure 25 - Syntaxe et exemple de règle symbolique utilisés dans KBANN

La Figure 25 montre un exemple d'une base de règles compatible avec les réseaux KBANN. Les règles expriment des conjonctions (opérateur 'et logique') entre les termes qui sont séparés par des virgules dans la liste des antécédents d'une règle. Les disjonctions (opérateur 'ou logique')

sont représentées par un ensemble composé de plus d'une règle dont les conséquents sont les mêmes. L'opérateur 'Not' permet d'inverser la valeur logique des prémisses, et les opérateurs 'Greater-Than' et 'Less-Than' réalisent certaines opérations de comparaison entre deux variables (valeurs discrètes ordonnées - variables nominales). L'opérateur 'N-True' introduit un concept intéressant du type N-parmi-M (*NofM*), c'est-à-dire que la prémisse est vraie si N des formules atomiques parmi les M listées sont vraies, sinon elle est fausse. Nous allons revenir sur ces opérateurs dans les sections suivantes.

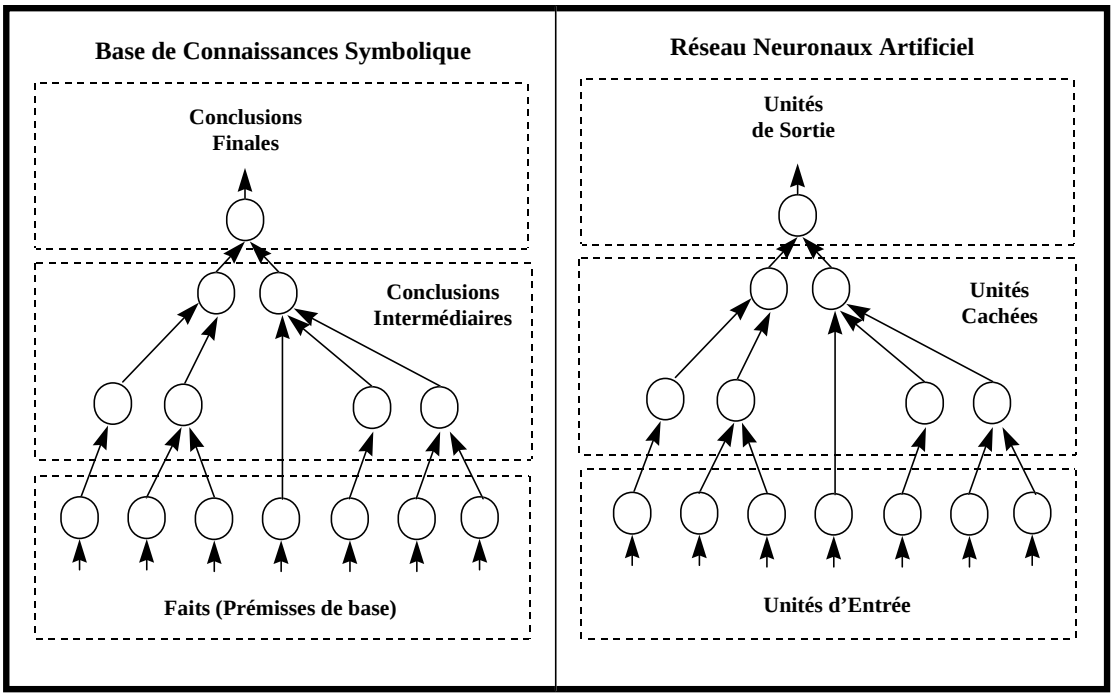


Figure 26 - Correspondance entre la structure d'une base de règles et d'un réseau de neurones

Le processus de compilation de règles dans un réseau est réalisé par un algorithme divisé en quatre étapes principales, à savoir :

1. *Réécriture des disjonctions* : les disjonctions sont réécrites de façon qu'une règle n'ait qu'une seule prémisse (formule atomique, sans négation) comme antécédent. Cette opération est nécessaire pour permettre de réaliser la compilation des règles dans un réseau. La réécriture des

règles ne change pas les connaissances ni le résultat de son application, elle va changer seulement la représentation écrite des règles. Voici un exemple de réécriture des disjonctions :

Avant réécriture	⇒	Après réécriture
A :- B, C, D.	⇒	A :- A'.
		A' :- B, C, D.
A :- D, E, F, G.	⇒	A :- A".
		A" :- D, E, F, G.

2. *Construction du réseau* : cette étape permet d'établir les correspondances entre un ensemble de règles et un réseau de neurones. La Figure 26 montre les correspondances entre ces deux types de représentations. Ainsi, la structure du réseau peut être créée à partir des règles symboliques disponibles. Une fois que toutes les unités sont bien disposées dans les couches du réseau selon ces correspondances, on passe à la définition des poids des connexions ainsi que du seuil de chacune des unités (*biais*). Les réseaux KBANN codent les poids et seuils selon le type de règle et les opérateurs qui y sont représentés (voir Figure 27) :

- Les poids des connexions originaires d'une prémisse positive reçoivent des valeurs égales à : **Poids = +W**;
- Les poids des connexions originaires d'une prémisse niée reçoivent des valeurs égales à : **Poids = -W**;
- Les seuils des unités qui représentent les conclusions obtenues à partir d'une conjonction reçoivent des valeurs égales à : **Seuil = (-P + 0.5) * W**;
- Les seuils des unités qui représentent les conclusions obtenues à partir d'une disjonction reçoivent des valeurs égales à : **Seuil = -0.5 * W**;
- La valeur W est une valeur constante spécifiée par l'utilisateur. Plus grand est W, plus grand sera l'écart entre une conclusion positive et une conclusion négative. KBANN utilise une valeur W=4, qui a été déterminé empiriquement et semble donner des bons résultats avec ce type de réseau;

- La valeur P indique le nombre de prémisses positives (sans négation) présentes dans la liste d'antécédents d'une unité qui représente une conclusion.

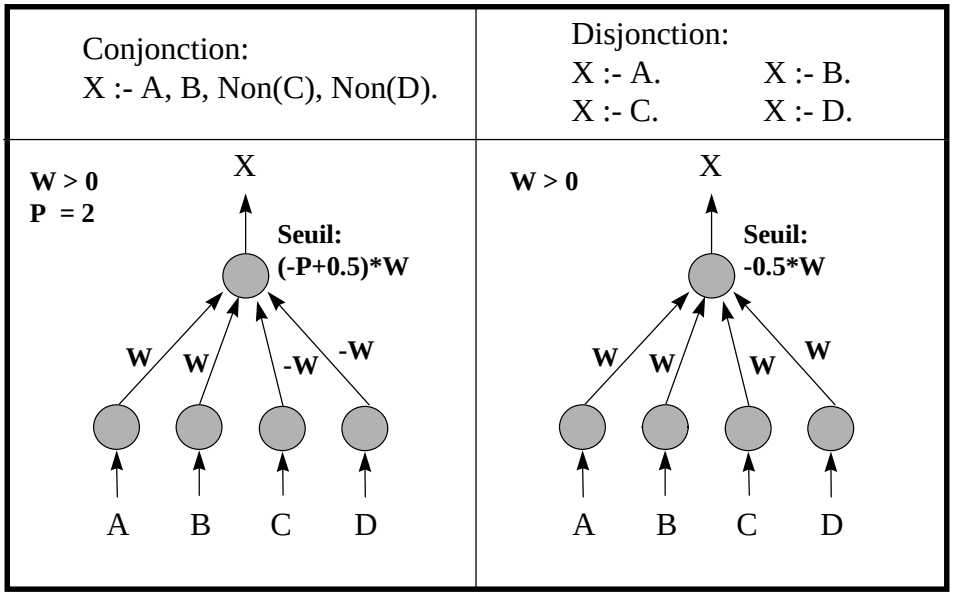


Figure 27 - Obtention des poids et des seuils des unités d'un réseau KBANN

Une preuve mathématique formelle et détaillé de l'exactitude des méthodes de transformation des règles vers des réseaux KBANN est présenté par Towell [TOW 91]. Donc, les réseaux KBANN obtenus par cette méthode codent parfaitement l'ensemble de règles. Des procédures similaires à celle-ci ont été proposées par d'autres chercheurs avant Towell [MCC 43, KLE 56].

3. *Ajout d'unités et de connexions* : une fois que la structure de base du réseau est construite, des unités peuvent y être ajoutés. Cette étape est optionnelle dans la construction des réseaux KBANN. On peut ajouter dans la couche d'entrée des unités qui représentent des attributs d'entrée non référencés dans la base de règles. On peut aussi ajouter des unités dans la couche cachée en cas d'indication explicite de l'expert à propos du placement de ces unités (procédure 'manuelle' d'insertion). Après l'ajout d'unités, on passe à l'étape d'ajout de connexions. Les connexions peuvent être ajoutés de façon à : lier toutes les unités appartenant à deux couches adjacentes; lier tous les unités d'une couche avec les unités des couches supérieures; lier chacune des entrées avec

chacune des unités cachées ou de sortie. Les connexions ajoutées reçoivent des poids et seuils égaux à zéro, et par conséquence, ne changent pas le comportement du réseau par rapport aux règles introduites.

4. Perturbation des poids : l'étape finale de la transformation de règles en réseau est réalisée par une perturbation de tous les poids du réseau. Cette perturbation consiste à additionner une valeur aléatoire très petite à chacun des poids des connexions. La perturbation introduite dans les poids est si faible qu'elle n'a aucun effet sur le comportement des réseaux KBANN avant l'apprentissage. Cependant, cette méthode permet d'éviter les problèmes d'apprentissage résultant d'une symétrie du réseau.

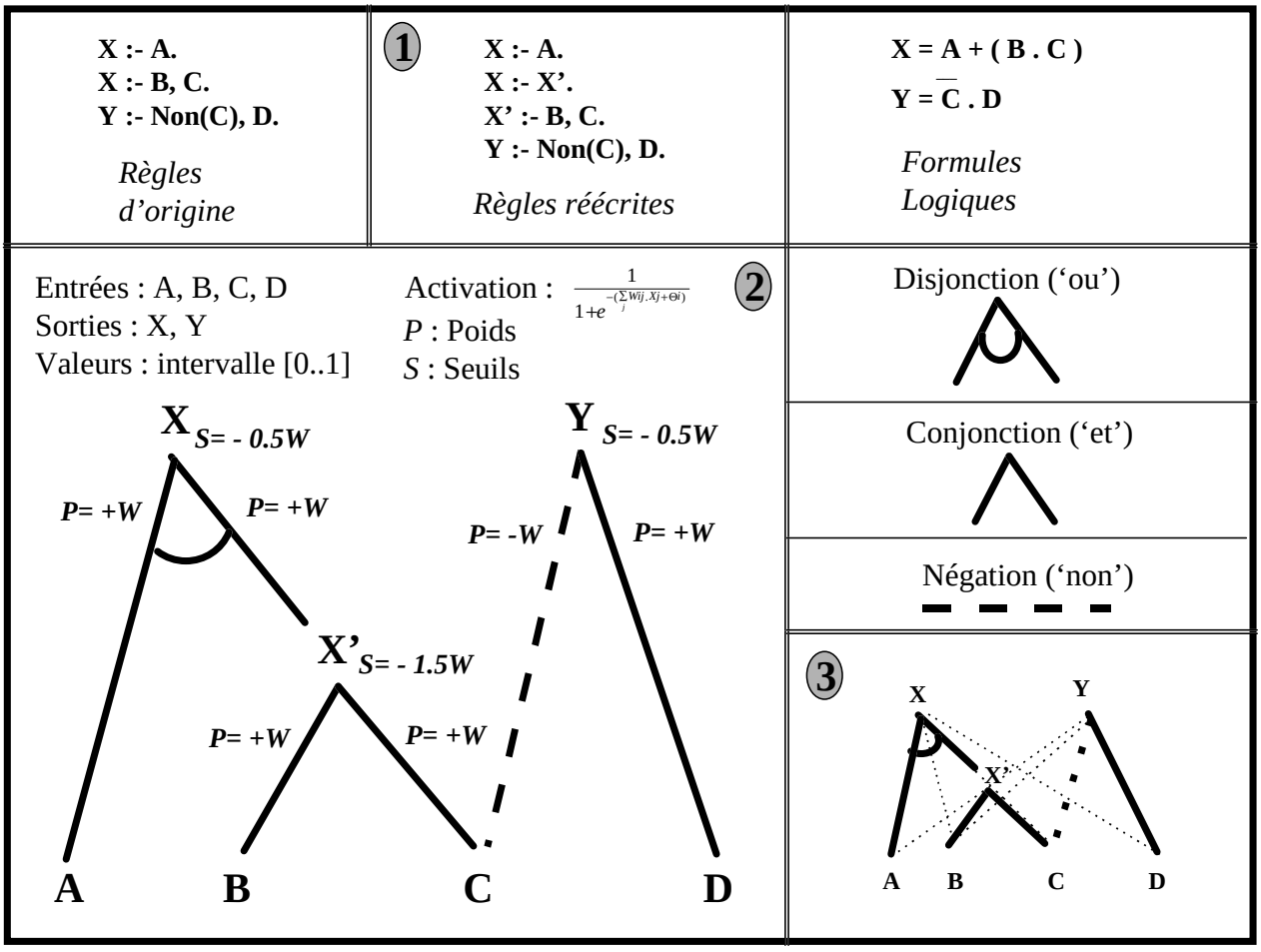


Figure 28 - Processus de transformation de règles dans un réseau KBANN

La Figure 28 montre un schéma du processus de transformation de règles en réseau. Nous précisons que les réseaux KBANN ainsi obtenus sont des réseaux de type PMC, où les valeurs d'entrée de chaque unité sont usuellement comprises dans l'intervalle [0,1] et les sorties de toutes les unités sont obligatoirement comprises dans l'intervalle [0,1]. Les réseaux KBANN utilisent une fonction de transfert du type sigmoïde asymétrique. Dans ces réseaux, une valeur d'entrée ou de sortie égale à 0.0 symbolise une prémisse/conclusion fausse et une valeur égale à 1.0 symbolise une prémisse/conclusion vraie. Les variables avec des attributs manquantes sont codées par des valeurs égales à 0.5. Cette valeur indique ainsi une 'indécision' dans la sortie d'une unité. Les réseaux KBANN n'utilisent pas de variables ayant des valeurs négatives. Les variables avec des attributs quantitatifs (valeurs continues) sont récodées d'une façon discrétisée par des sous-intervalles, où chaque sous intervalle correspond à des valeurs d'activation dans l'intervalle [0..1] (voir Figure 29).

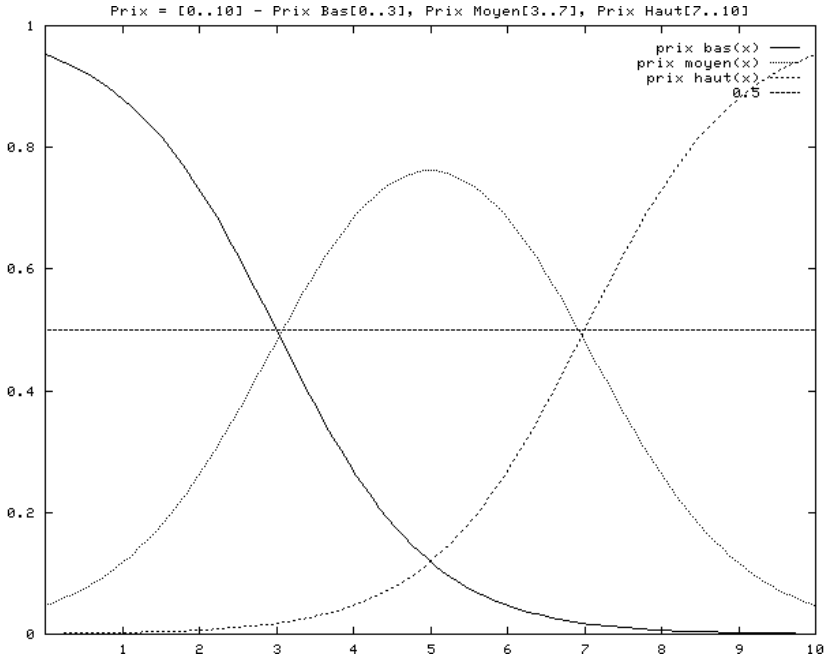


Figure 29 - Exemple de codage d'attributs numériques dans KBANN

3.3.2.2 L'Apprentissage dans les Réseaux KBANN

Les réseaux obtenus à partir des règles symboliques peuvent ensuite être entraînés avec une base d'exemples. Cette procédure permet de raffiner les connaissances introduites précédemment dans le réseau. Ce raffinement se fait en utilisant l'algorithme d'apprentissage de la Rétro-Propagation de l'erreur (voir section 2.3.2)[RUM 96].

Malheureusement, les réseaux KBANN posent des problèmes par rapport à l'algorithme classique de Rétro-Propagation, car ils commencent l'apprentissage avec un comportement fortement influencé par les connaissances compilées sous forme de réseau. Les réseaux compilés donnent des réponses sûres, c.-à-d. très proches de 1.0 (vrai) ou de 0.0 (faux), ce qui va rendre beaucoup plus difficile l'apprentissage. En raison de la formulation mathématique de la Rétro-Propagation, les unités qui ont une sortie proche de zéro ou proche de un ne vont pas avoir une adaptation significative de leurs poids synaptiques indépendamment de l'erreur constatée dans leur sortie. Puisque la dérivée de la fonction d'activation est très faible quand la sortie de l'unité s'approche de 1.0 ou 0.0 (voir Figure 14), cela va bloquer le processus d'apprentissage (voir Section 2.3.2 - Eq.18 et 24). Ce phénomène de paralysie de l'apprentissage est connu comme le problème du *'flat spot'* de la Rétro-Propagation [FAH 88, TOW 91].

Parmi les solutions possibles qui peuvent être utilisées pour résoudre ce problème, Towell a choisi de modifier la fonction d'erreur (erreur quadratique) utilisée par l'algorithme de la Rétro-Propagation. Donc, la seule différence entre la méthode d'apprentissage des réseaux KBANN et la méthode de la Rétro-Propagation se trouve dans la fonction d'erreur utilisé. Dans les réseaux KBANN, cette fonction est la fonction d'erreur d'entropie-croisée (*cross-entropy error function*) [HIN 89]. Nous présentons ci-dessous les différences entre les équations utilisées dans KBANN et dans la Rétro-Propagation.

- Rétro-Propagation - Algorithme classique :

  Erreur quadratique :

$$E = \frac{1}{2}\sum_i (Di - Ai)^2$$

  Equation utilisée dans le calcul de l'erreur d'une unité (voir Eq. 12, Section 2.3.2) :

$$\frac{\partial E}{\partial Si} = \frac{\partial E}{\partial Ai}\frac{\partial Ai}{\partial Si} = -(Di - Ai).Fa'(Si) = \delta i$$

- Rétro-Propagation - Algorithme utilisé dans KBANN :

  Erreur basée sur la fonction d'entropie-croisée :

$$CE = -\sum_i [(1 - Di) * Log_2(1 - Ai) + Di * Log_2(Ai)]$$

  Equation utilisée dans le calcul de l'erreur d'une unité :

$$\frac{\partial E}{\partial Si} = \frac{\partial E}{\partial Ai}\frac{\partial Ai}{\partial Si} = Log(2) * (\frac{Di}{Ai} - \frac{1 - Di}{1 - Ai}) * Ai * (1 - Ai) = \delta i$$

  Dans le cas particulier où Di va assumer seulement deux valeurs (Di=1.0 ou Di=0.0) :

  δi = (1 - Ai) pour Di=1.0

  δi = -Ai pour Di=0.0

La caractéristique de cette fonction est qu'elle a une dérivée toujours très forte si, pour une sortie particulière, l'écart entre la valeur désirée Di et la valeur réelle Ai est grand. Donc, cela augmente la correction effectuée sur les poids des unités dont les valeurs de sortie sont proches de zéro alors qu'elles devraient être à un, et vice-versa. Ainsi, on augmente la vitesse d'apprentissage et on évite le problème de la paralysie de l'adaptation du réseau. En conclusion, la fonction d'entropie-croisée est plus adaptée aux réseaux KBANN et au type de sortie données après la transformation de règles en réseau. D'après Towell [TOW 91], les résultats pratiques des comparaisons entre les réseaux KBANN avec une fonction d'erreur classique et les réseaux KBANN avec une fonction d'erreur d'entropie-croisée, montrent une supériorité (performance et généralisation) de ce dernier type de fonction d'erreur.

3.3.2.3 Extraction de Règles des Réseaux KBANN

Deux méthodes d'extraction de règles à partir des réseaux ont été développées pour être utilisées avec les réseaux KBANN. La première méthode, qui s'appelle SUBSET, est similaire aux approches proposées par Saito et Nakano [SAI 88], et par Fu [FU 91]. La deuxième méthode a été crée par Towell et Shavlik [TOW 91] et s'appelle algorithme NofM. Tout d'abord, nous allons présenter les propriétés et les caractéristiques qui sont communes aux deux types de méthodes, puis nous allons détailler chacune de ces méthodes dans les section suivantes.

Deux hypothèses sont faites par rapport aux réseaux obtenus par compilation et raffinés par apprentissage :

1. Les unités des réseaux KBANN peuvent être assimilées à des unités à seuil : lorsque leurs valeurs d'entrée (somme pondérée) sont en dessous du seuil qui leur correspond, leur activation est proche de 0.0, sinon elle est proche de 1.0. La fonction d'activation utilisée est toujours la sigmoïde asymétrique, mais selon la spécification de certains paramètres (e.g. la valeur de 'W'), cette fonction peut être assimilée à la fonction seuil [JOD 94a, TOW 93].

2. L'apprentissage ne modifie pas d'une façon significative le rôle qu'une unité avait avant l'apprentissage. Ainsi, nous pouvons garder les identificateurs (étiquettes) des unités qui leur ont été attribués en correspondance aux règles symboliques compilées. Les règles extraites à partir d'un réseau sont plus facilement interprétables, car elles conservent les noms attribués aux prémisses et conclusions des règles symboliques.

Les deux méthodes s'appuient sur l'idée que l'on doit trouver les combinaisons des valeurs d'entrée d'une unité qui font que son activation de sortie soit proche de 1.0. Cette combinaison d'entrées nécessaire à l'activation d'une unité va être exprimé sous la forme d'une règle symbolique. Par conséquence, les méthodes d'extraction cherchent à étudier les relations établies

entre l'ensemble des entrées et la valeur du seuil d'activation d'une unité. Une valeur résultant de la somme pondérée des entrées d'une unité supérieure au seuil implique que cette unité soit activée (valeur d'activation proche de 1.0), et par conséquence on va exprimer cette relation entre les entrées et l'activation de la sortie par une règle (e.g. Si entrée1 et entrée2 et non(entrée3) Alors unité-conclusion est active). Une valeur résultante de la somme pondérée des entrées d'une unité inférieur au seuil implique que cette unité ne va pas être activée (valeur d'activation proche de 0.0), et par conséquence on ne va pas exprimer par une règle la relation entre les entrées et la sortie de cette unité.

Puisque les unités des réseaux KBANN ont des entrées toujours très proches de 0.0 ou de 1.0, cela va beaucoup simplifier la recherche des relations valables entre les entrées et la sortie. Nous considérons comme valables les relations dont la sortie de l'unité est activée, c.-à-d. quand on a 'Ai > 0.5'. Ce sont les 'relations valables' qui donnent l'origine aux règles extraites. Dans le cas des réseaux KBANN, l'espace de recherche est réduit à cause des caractéristiques particulières de ces réseaux que l'on vient de décrire. Ainsi, on peut résumer le problème de l'extraction de règles comme suit :

♦ Entrée = 0.0	⇒ Poids = +Wij	⇒ Wij.Xj = 0.0	N'ont pas d'effet sur
♦ Entrée = 0.0 (niée)	⇒ Poids = -Wij	⇒ Wij.Xj = 0.0	la somme pondérée

♦ Entrée = 1.0	⇒ Poids = +Wij	⇒ Wij.Xj= +Wij	Préservent le signe et la
♦ Entrée = 1.0 (niée)	⇒ Poids = -Wij	⇒ Wij.Xj = -Wij	valeur des poids

Sachant que : $Ai = Fa\ (\Sigma_j\ Wij.Xj + \Theta i)$

$Fa(Si) \approx 1.0$ quand $Si > 0.0$

$Fa(Si) \approx 0.0$ quand $Si < 0.0$

Alors on cherche à savoir si : $\Sigma_j\ Wij.Xj + \Theta i > 0.0$

Comme nous savons que $\Theta i < 0.0$ et $Xj \approx 0.0$ ou 1.0, ce qu'il faut trouver ce sont toutes les combinaisons des poids qui dépassent la valeur du seuil : $\Sigma_j Wij > \Theta i$

Finalement, il est important de souligner que cette façon simplifiée de traiter les sorties des activations et de faire la recherche des 'relations valables', a pour résultat l'obtention d'un ensemble de règles qui n'est pas forcement tout à fait égal au réseau qu'il est sensé représenter. Une base de règles obtenue par extraction à partir d'un réseau peut contenir des erreurs d'omission ou de généralisation excessive par rapport à ce réseau. Ces erreurs sont généralement le résultat d'un cumul d'effets des poids faibles qui ne sont pas représentés dans les règles extraites. Un des principaux problèmes de l'extraction de règles est la minimisation de ce type d'erreurs tout en évitant de simplement réécrire le contenu du réseau dans un langage de plus haut niveau.

3.3.2.3.a Algorithme SUBSET d'extraction de règles

L'algorithme SUBSET est une méthode similaire aux approches décrits par Saito et Nakano, ainsi que par L. Fu [SAI 88, FU 91]. Il a été nommé SUBSET car on cherche les sous-ensembles des poids d'entrée d'une unité qui cumulés permettent de depasser son seuil d'activation. Donc, le principe de l'algorithme SUBSET va être de trouver des combinaisons d'unités entrantes (connectés à l'entrée de l'unité en question) qui doivent être activées ou inactivées pour que l'unité correspondante soit activée. Pour chaque combinaison obéissant à cette définition, on crée et on ajoute à la base de règles la règle ayant en prémisses les noms des unités devant être activées et les noms des unités devant être inactivées, et en conclusion, le nom de l'unité étudiée.

On essaie, pour cela, de trouver des sous-ensembles d'unités reliées par des poids positifs dont la somme dépasse le seuil de l'unité étudiée. Pour chaque groupe de poids positifs ainsi exhibés, on regarde parmi les poids négatifs, s'il en existe dont la somme des poids annule les poids précedents plus le seuil. S'il existe de tels poids, alors, les entrées respectives doivent être niées, afin que le groupe de poids positifs extrait active l'unité, sinon, l'activation de ce groupe de poids suffit pour produire l'activation de la sortie de l'unité en question. A cet effet, l'algorithme suivant présenté dans le Tableau 6 a été développé.

```
Pour chaque sous-ensemble Bp de poids positifs reliés à l'unité Ui
        Si  Somme(Bp) + Θi > 0.0
        Alors ajouter Bp à Gp

Pour chaque ensemble P ∈ Gp
        Pour chaque sous-ensemble Bn de poids négatifs reliés à Ui
                Soit Z un nouveau prédicat non utilisé jusqu'à présent
                Si  Somme(Bp) + Somme(Bn) + Θi < 0.0
                Alors CréerRègle("Z :- Bn")
        Si Z a des antécédents
        Alors CréerRègle ("Ui :- P, non(Z)")
        Sinon Ui :- P

Elimination des règles dupliquées

Ui : Unité 'i' de la couche cachée ou de la couche de sortie
Θi : Seuil d'activation de l'unité 'i'
```

Tableau 6 - Description de l'algorithme SUBSET

La Figure 30 présente un exemple de l'application de l'algorithme SUBSET à une unité d'un réseau KBANN. Les régles qui y sont représentées, sont le résultat obtenu par l'application de cet algorithme.

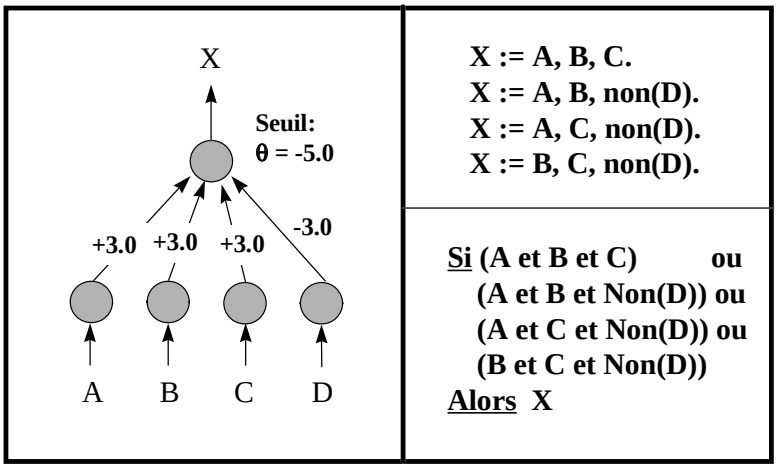


Figure 30 - Exemple d'application de l'algorithme SUBSET

L'avantage de cette technique est que les règles qui en sont extraites sont compréhensibles et faciles à exploiter. Par contre, la complexité de cet algorithme impose un temps d'exécution qui peut augmenter significativement en fonction du nombre d'entrées de l'unité analysée, car il exécute une recherche de toutes les possibilités de combinaisons des entrées/poids qui mènent à l'activation de la sortie. De plus, cet algoritme doit être appliqué à chaque unité de la couche cachée et de la couche de sortie du réseau. En conséquence, le nombre de règles exhibées augmente très rapidement en fonction du nombre d'entrées et d'unités du réseau, ce qui limite parfois la capacité d'interprétation des règles par un expert et par conséquence son utilisation pratique.

3.3.2.3.b Algorithme NofM d'extraction de règles

Towell, ainsi que les autres chercheurs qui étudient des techniques d'extraction de règles à partir des RNA, ont cherché à perfectionner ces méthodes et à améliorer la qualité des règles obtenues par leurs algorithmes d'extraction. Les critères qu'on utilise pour mesurer la qualité des règles et des algorithmes d'extraction sont, par exemple [AND 95, BOZ 95] :

- L'*exactitude* des règles obtenues : sert à indiquer si les règles sont capables de bien classer les exemples appris et si elles ont une bonne capacité de généralisation;

- La *fidélité* des règles : vérifie si les règles sont capables de bien reproduire le comportement du réseau à partir duquel elles ont été obtenues;
- La *consistance* des règles : sert à déterminer si les résultats obtenus avec la base de règles sont assez proches de ceux obtenus à partir de l'utilisation du réseau, même dans des conditions d'utilisation qui soient différentes de celles rencontrés pendant l'apprentissage;
- L'*intelligibilité* des règles : mesure la difficulté de compréhension des connaissances codées dans la base de règles. Cette mesure est basée principalement sur deux paramètres : le nombre total de règles extraites (moins de règles ⇒ plus d'intelligibilité, plus de règles ⇒ moins d'intelligibilité), et le nombre d'antécédents moyen de chaque règle;
- La *complexité* de l'algorithme d'extraction.

L'algorithme NofM (appelé aussi de MofN) créé par Towell [TOW 91, TOW 93] dans le cadre du système KBANN, a été développé pour pallier les inconvénients de l'algorithme SUBSET et pour essayer d'augmenter la qualité des règles et de l'algorithme d'extraction, selon les critéres que nous venons de décrire. Le principe de fonctionnement de l'algorithme NofM est très proche de celui de l'algorithme SUBSET : on analyse chaque unité de façon séparée en essayant de trouver des relations entre celle-ci et ses antécédents. La première différence se situe au niveau de la forme des règles extraites, qui sont exprimées en fonction d'une primitive de base : NofM. Cette primitive prend en entrée un entier et une base de faits et rend en sortie un booléen. La formule *NofM(i, P_1, P_2, ..., P_n)* rend vraie (1.0) sa sortie ssi au moins i propositions parmi P_1, ..., P_n sont vraies, sinon elle donne la valeur fausse (0.0). Cette forme a l'avantage d'être plus réduite et nécessite moins de règles pour exprimer les connaissance. Par exemples, les règles extraites de l'exemple présenté dans le paragraphe précédent peuvent s'exprimer par la seule règle : X <- NofM (3, A, B, C, non (D)).

Cet algorithme fonctionne selon le principe suivant, en six étapes :

1. **Regroupement (*clustering*)** : on partitionne, pour chaque unité, l'ensemble des liaisons d'entrée en classes d'équivalence, les liaisons d'une même classe d'équivalence portant des poids dont les valeurs sont proches les unes des autres. La *'proximité'* des valeurs des

poids est établie par un paramètre fourni à l'algorithme d'extraction. KBANN utilise une valeur constante de distance inférieur à 0.25 pour regrouper les valeurs des poids des connexions.

2. **Moyenne (*averaging*)** : ensuite, on fixe la valeur des poids des liaisons d'une même classe à la moyenne des poids de cette classe.

3. **Elagage (*eliminating*)** : on détermine, de façon heuristique, les classes non significatives pour l'unité étudiée, c'est-à-dire les classes dont la valeur d'activation est indifférente pour l'activation de l'unité dont on extrait les règles. Pour cela, on procède sur une base d'exemples: pour chaque exemple, on calcule l'activation du réseau, et on regarde si, en annulant toutes les activations des poids d'une classe particulière, l'activation de l'unité étudiée est modifiée. Si aucun changement notable n'est observé, alors la classe n'est pas significative pour l'exemple courant. Une fois que toute la base a été examinée, on enlève les classes qui ne sont significatives pour aucun exemple de la base.

4. **Optimisation (*optimizing*)** : des modifications sur le seuil de l'unité étudiée peuvent être nécessaires, du fait de l'erreur que peut apporter la partition en classes d'équivalence de ses poids d'entrées, ainsi que la suppression des classes non significatives. Pour cela, on recommence un apprentissage, en gelant tous les poids portés par les liaisons, de manière à ce que seuls les seuils des unités du réseau soient modifiés.

5. **Extraction (*extracting*)** : on exprime ensuite, pour chaque unité du réseau, les relations entre l'unité étudiée et ses antécédents sous la forme d'une inéquation. Cette inéquation est exprimée de la manière suivante :

   - Soit Θ le seuil de l'unité étudiée;
   - Soit C_1, C_2, ..., C_n les identificateurs des liaisons entrantes, portant les poids W_1, W_2, ..., W_n;

- L'inéquation associée à chaque unité est la suivante :

W_1*NTrue(C_1)+W_2*NTrue(C_2)+ ... + W_n*NTrue(C_n) + Θ >0.

NTrue est une primitive prenant en entrée une liste de faits, et rendant en sortie un entier qui représente le nombre de faits justes (vrais) parmi ceux de la liste d'entrée.

6. **Simplification (*simplifying*)** : on réécrit enfin les inéquations sous forme de règles de type NofM, en recherchant, dans chaque groupe, le nombre de règles permettant à l'unité responsable de la conclusion d'être activée. Par exemple :

L'unité X possède des liaisons d'entrée à partir des antécédents identifiés par A, B, C, D, réliées par des connexions de poids égaux à 5.1, et des antécédents identifiés par M, N, O, réliées par des connexions de poids égaux à 3.5. L'unité X a un seuil Θ = –10.0. Cette unité est représentée par l'équation suivante :

X :- 5.1 * NTrue (A, B, C, D) + 3.5 * NTrue (M, N, O) > 10.0

Après simplification cet équation va être représentée par les règles suivantes :

X :- NofM(2, A, B, C, D).

X :- NofM(1, A, B, C, D), NofM(2, M, N, O).

X :- M, N, O.

La Figure 31 montre un exemple du processus complet d'extraction de règles par la méthode NofM. Cet algorithme fourni des ensembles de règles assez compacts qui sont usuellement beaucoup plus petits que les ensembles de règles obtenus par l'autre algorithme, SUBSET. Le fait d'exprimer les règles sous une forme plus compacte peut nous permettre d'en réduire le nombre, et ainsi d'en augmenter la compréhension. Par contre, les règles du type NofM sont moins naturelles pour un utilisateur humain, et, même si elles sont plus compactes, elles n'augmentent pas forcément l'intelligibilité de l'ensemble.

En ce qui concerne la complexité des algorithmes d'extraction de KBANN, Towell a déterminé la complexité de l'algorithme SUBSET comme étant de l'ordre exponentielle et la complexité de l'algorithme NofM, dans la pratique, comme étant de l'ordre cubique. En effet, la complexité de NofM est assez difficile à analyser avec précision en raison de la phase d'apprentissage pour l'optimisation des seuils des unités. Towell estime la complexité de l'apprentissage à $O(n^3)$ et les autres étapes de l'algorithmes ont une complexité de O(n), sauf pour l'étape de regroupement des poids qui est de O(n*log(n)). En bref, cet algorithme peut être efficace pour extraire des règles d'unités ayant beaucoup de liaisons, mais semble mal adapté dans le cas où peu de liaisons sont en jeu et la phase d'apprentissage finit par être plus importante.

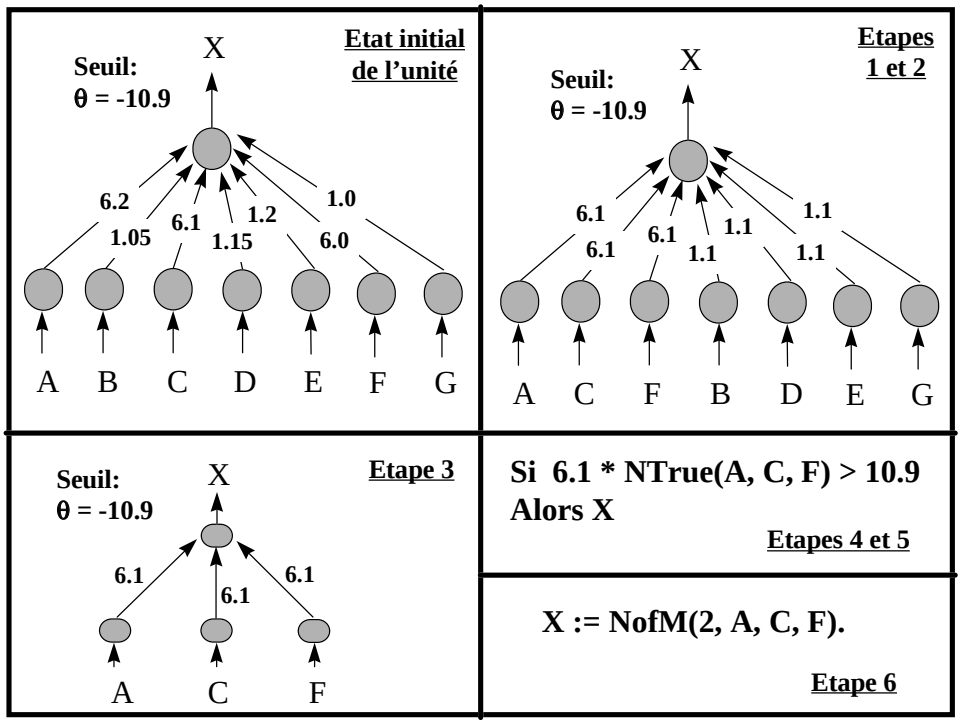


Figure 31 - Exemple d'application de l'algorithme NofM

3.3.2.4 Discussion sur les Réseaux KBANN

Ce système est très intéressant, car il combine de manière très forte une optique symbolique et une optique connexionniste. Cette méthode d'intégration nous permet de tirer pleinement profit de la complémentarité entre ces deux types de modélisation. Par contre, son inconvénient majeur est que, lors de l'apprentissage, il ne modifie pas la structure du réseau, mais seulement les valeurs des poids entre les cellules. Nous allons dresser ici une liste des points forts de KBANN suive d'une autre liste des limitations dont nous pensons qu'elles constituent les points faibles les plus importants de ce type de réseaux.

Les réseaux KBANN présentent des points forts tels que :

- Le transfert bidirectionnel des connaissances entre les modes de représentation connexionniste et symbolique est réalise de manière simple et directe. Les réseaux KBANN nous permettent de profiter en même temps des connaissances théoriques disponibles (règles symboliques) et des connaissances pratiques disponibles (exemples). Ce type de réseau essaie d'utiliser et d'intégrer toutes les connaissances disponibles sur le problème;
- L'insertion de connaissances (règles) dans les réseaux permet d'accélérer significativement le processus d'apprentissage et de résoudre le problème de la spécification d'une topologie initiale et des poids initiaux des réseaux;
- La structure du réseau obtenue par compilation des règles garde une correspondance directe avec la base de règles d'origine. Les concepts intermédiaires y sont préservés;
- Les réseaux KBANN permettent de réaliser un raffinement d'une base de règles approximativement correcte par apprentissage d'exemples;
- L'apprentissage des réseaux KBANN peut être fait à partir d'une base d'exemples plus petite étant donné que le réseau initial peut coder des connaissances a priori sur le problème;
- Les réseaux KBANN sont relativement robustes pour le traitement de problèmes représentés par des bases de règles et d'exemples qui contiennent des données incomplètes ou incorrectes;

- Les méthodes d'extraction de règles utilisées permettent de bien retrouver les connaissances compilées dans le réseau et raffinées par la suite. Ces méthodes profitent du fait que le réseau préserve dans sa structure des liens directs avec la base de règles d'origine;

- Deux méthodes d'extraction de règles à partir des réseaux sont proposées dans KBANN : la méthode SUBSET et la méthode NofM. En particulier, la méthode NofM exploite l'idée de l'extraction d'un type particulier de règles, avec un algorithme plus performant que SUBSET (sous certaines conditions). Ces règles du type NofM ont un plus grand pouvoir d'expression des connaissances et elles permettent de coder les connaissances d'une façon plus compacte.

Cependant, les réseaux KBANN possèdent un certain nombre de points faibles et limitations, à savoir :

- Les règles symboliques sont des propositions d'ordre 0. Les réseaux KBANN ne sont pas très bien adaptés pour traiter des entrées avec des valeurs continues, sauf si celles-ci sont préalablement traitées (e.g. discrétisation des valeurs continues). Les valeurs continues sont pour KBANN des entrées entre 0.0 et 1.0 qui représentent la 'certitude' ou la 'proximité' des valeurs par rapport aux concepts qu'elles représentent (le valeur 0.5 représentant justement l'indécision);

- L'algorithme d'apprentissage représente un des plus gros défauts des réseaux KBANN. Cet algorithme, une version très proche de la Rétro-Propagation du gradient, présente un bon nombre d'inconvénients :

  - ♦ L'apprentissage est fait sur des réseaux statiques. Les changements subis par les réseaux pendant la phase d'apprentissage ne permettent d'exploiter que l'espace des poids, et la structure des réseaux reste toujours inaltérée. Towell propose l'inclusion d'unités dans les réseaux KBANN, mais cela est fait de façon 'manuelle' et avant apprentissage;

  - ♦ L'algorithme de la Rétro-propagation ne marche pas très bien dans les réseaux composés d'un très grand nombre de couches. La méthode de la rétro-propagation de l'erreur vers les couches précédentes ne permet pas d'estimer l'erreur précisément. Ainsi, les ajustements des unités sont de moins en moins précis. Malheureusement, le processus de création de réseaux par compilation de règles

obtient la plupart du temps des réseaux avec une quantité assez importante de couches, car elle préserve la hiérarchie présente dans la base de règles symboliques;

- Towell a choisi d'employer un algorithme classique de rétro-propagation et n'a pas pris en compte le fait qu'on dispose maintenant de nombreuses techniques beaucoup plus performantes (e.g. Quick-Prop, R-Prop, Gradient Conjugué). L'algorithme de la Rétro-Propagation présente des inconvénients tels que : la faible vitesse de convergence, la difficulté de réglage des paramètres (α et β), la paralysie de l'apprentissage et l'oubli catastrophique.
- Une analyse et des critiques de la méthode d'apprentissage des réseaux PMC par Rétro-Propagation a été présenté dans la Section 2.3.3.

- Les réseaux KBANN ne marchent pas très bien dans les cas où la base de règles a des problèmes significatifs d'incomplétude ou d'incorrection. Un réseaux KBANN est capable d'ajouter/effacer des antécédents dans les règles, mais il est incapable de créer des nouvelles règles (unités) [OPI 95a]. Les changements nécessaires dans la base de règles ne doivent pas être très importantes pour que l'algorithme d'apprentissage de KBANN puisse fonctionner;

- Les réseaux KBANN doivent préserver les significations d'origine associés aux unités à partir du processus de compilation de règles. L'étiquette associée à une unité n'est valable que si, après l'apprentissage, cette unité conserve son rôle d'origine. Le problème est qu'on ne peut pas être sûr que l'apprentissage ne va pas réaliser un déplacement de la signification des unités (*meaning shift problem*);

- L'extraction de règles dans KBANN implique l'analyse de toutes les unités du réseau. Les méthodes utilisées dans ce type de réseaux ne permettent pas de réaliser une extraction orientée seulement vers les nouvelles connaissances acquises, car l'algorithme d'apprentissage réalise des adaptations sur toutes les unités du réseau. Les règles d'origine qui ont été compilées sous la forme d'un réseau peuvent être modifiées et ainsi les connaissances de départ sont mélangées avec les nouvelles connaissances acquises par apprentissage. Par conséquence, l'algorithme d'extraction doit analyser toutes les unités du réseau afin d'extraire les connaissances qui y sont représentées;

- L'algorithme d'extraction de règles utilisé dans les réseaux KBANN est orienté spécifiquement vers ce type de réseaux, c'est-à-dire qu'il lui faut un réseau avec des sorties 'binaires' (les sorties sont toujours très proches de 0.0 ou de 1.0). Les algorithmes utilisés dans KBANN ne permettent pas de manipuler des valeurs de sortie continues, d'extraire des règles floues, et principalement d'utiliser des valeurs d'entrée 'vraiment' continues;

- Le type de règles symboliques utilisées dans les réseaux KBANN (insertion et extraction) ont une capacité limitée de représentation de connaissances. Nous pensons qu'il faut étudier la possibilité d'utiliser des règles de plus haut niveau pour arriver à une meilleure représentation des connaissances sur certains problèmes (voir Section 4.2.1 et Section 5.3). La capacité de généralisation fournie par un système est aussi limitée par sa capacité de représenter les connaissances.

Nous avons essayé de retenir les points positifs de KBANN, mais aussi de corriger ses points faibles et ses limitations, afin d'obtenir un SHNS plus performant - le système INSS. Pour cela, nous avons choisi de travailler sur un modèle de réseau incrémental avec un apprentissage plus performant, ce qui nous a permis de résoudre une bonne partie des problèmes que nous venons de citer.

3.3.3 Systèmes Connexionnistes de Raffinement de Connaissances

Nous allons discuter brièvement ici les approches connexionnistes de raffinement de connaissances (*connectionist theory refinement*). Les systèmes KBANN et SYNHESYS étant deux exemples typiques de ce type d'approche. En particulier, nous nous sommes intéressés au système KBANN puisqu'il est basé sur des réseaux de type PMC. Ce système présente une forte limitation, car il ne permet pas de faire évoluer la structure de ces réseaux. Par contre, cela ne pose pas de problèmes aux réseaux à prototypes, comme ceux utilisés par SYNHESYS. Cependant, nous trouvons dans la littérature d'autres systèmes connexionnistes de raffinement de connaissances qui sont très proches ou qui sont dérivés de KBANN, dont certains permettent même de faire évoluer la structure du réseau.

Le premier exemple des méthodes dérivées à partir de KBANN sont les algorithmes de construction de réseaux proposés par D. Opitz : TopGen, Regent et Addemup [OPI 95a, OPI 95b, OPI 97]. Ces trois méthodes utilisent les algorithmes de base de KBANN pour la construction d'un réseau initial suivi d'un raffinement des connaissances par apprentissage en utilisant la Rétro-Propagation. La différence entre ces algorithmes et KBANN vient du fait qu'ils permettent de faire évoluer l'architecture des réseaux à travers la génération de plusieurs topologies de réseaux et de l'application d'algorithmes génétiques. Plusieurs réseaux possédants des structures légèrement différentes sont soumis à un processus d'apprentissage avec le même algorithme employé dans KBANN. Ensuite, sont appliquées des techniques de sélection du meilleur réseau (sauf dans le cas de la méthode Addemup, où l'on cherche une architecture modulaire issue d'une combinaison de plusieurs réseaux). Ces trois techniques sont assez intéressantes, mais on peut les classer comme des méthodes du type "force brute", étant donné qu'ils sont très lourds du point de vue computationnel. De plus, les recherches développées aboutissent à des méthodes où l'on ne se pose jamais de questions à propos de la qualité des données d'apprentissage. Les "règles approximativement correctes" compilées dans un réseau sont raffinées à partir d'une base d'apprentissage, mais on n'étudie pas beaucoup les effets résultant de l'utilisation de données "approximativement correctes" dans cette base. Le seul but de ces méthodes est d'obtenir la meilleure généralisation possible à partir des informations disponibles, sans chercher à interpréter les connaissances acquises, comme par exemple, à travers l'identification des règles incorrectes et des nouvelles règles acquises. Les méthodes d'extraction de règles, même s'il n'y a pas de raison apparente qui nous empêche de les utiliser sur les réseaux obtenus par TopGen, Regent ou Addemup, n'ont pas été appliquées sur ces réseaux.

Un autre exemple de réseau connexionniste de raffinement de connaissances, permettant de faire évoluer sa structure, a été développé par J. Fletcher [FLE 93, FLE 94]. Il propose d'utiliser un réseau obtenu par compilation de règles comme une des unités cachées d'un réseau à trois couches (reliée aux unités d'entrées et aux unités de sortie). Les autres unités cachées du réseau sont ajoutées par un algorithme constructif appelé HDE [FLE 94]. Cette approche est intéressante puisqu'on préserve toutes les connaissances introduites par compilation, mais il présente des inconvénients tels que:

- Les réseaux ont une structure restreinte à seulement trois couches;

- Nous ne pouvons pas profiter des connaissances intermédiaires contenues dans les unités obtenues par compilation. Le réseau de départ reste une boîte noire incrustée à côté des unités du réseau constructif;

- Les unités ajoutées dans la couche cachée du réseau sont des unités qui ne semblent pas être compatibles avec les algorithmes d'extraction de règles utilisées dans les réseaux KBANN. La sortie d'une unité n'est pas forcement toujours proche de 0.0 ou de 1.0;

- Les travaux faits par Fletcher sont concentrés surtout sur la construction de réseaux par ajout d'unités. Dans sa thèse, il n'a pas développé d'études plus approfondies sur l'insertion de règles dans un réseau, ni sur la possibilité d'extraire des règles à partir d'un réseau raffiné.

Notre intérêt pour les travaux développés par Fletcher se concentre uniquement sur l'idée de garder le réseau d'origine obtenu par compilation de règles et d'employer un algorithme constructive pour faire évoluer l'architecture du réseau. Les études menées par Opitz [OPI 95a] ont démontré que cette approche est assez limitée par rapport à sa capacité de raffiner et de corriger les connaissances de départ.

Les travaux développés par J. Mahoney dans le cadre de son système RAPTURE [MAH 93, MAH 96], ont abouti à la création d'un système de raffinement de connaissances exprimées par des règles probabilistes (utilise des coefficients de certitude associés aux règles). Le système RAPTURE utilise une méthode de construction de réseaux, basée sur l'algorithme Upstart [FRE 90], qui fait l'ajout d'unités dans la couche cachée connectées directement aux entrées et à une unité de sortie. La méthode employée permet d'obtenir des réseaux assez semblables à ceux obtenus par Fletcher, sauf que dans ce cas Mahoney utilise des réseaux obtenus à partir de règles avec des coefficients de certitude. A notre avis, les principales limitations du système RAPTURE sont :

- Il fait l'apprentissage jusqu'à arriver à 100% de réponses correctes sur la base d'apprentissage, ce qui peut mener à un surapprentissage;

- Il n'est pas capable de traiter des problèmes avec des contradictions (base avec des exemples incorrects);

- Il travaille de la même façon que KBANN par rapport aux entrées continues, et donc il possède les mêmes limitations que KBANN par rapport au traitement de ce type de données;

- Les unités ajoutées dans les réseaux vont constituer une structure toujours composée par seulement trois couches (cela n'est pas le cas des règles compilées sous forme de réseau).

De plus, l'algorithme d'apprentissage employé possède des paramètres difficiles à régler tel l'algorithme classique de la Rétro-Propagation du gradient. Dans sa thèse[MAH 96], Mahoney propose comme futurs travaux possibles, l'étude d'autres techniques connexionnistes pour améliorer les performances de son système. Il indique aussi que l'on pourra interagir avec les experts afin de mieux comprendre les erreurs présentes dans la base de connaissance théorique et étudier les nouvelles règles acquises par le système par apprentissage d'exemples.

Le dernier type de système de raffinement de connaissances que nous allons aborder ici, sont les systèmes basés sur des réseaux récurrents. Dans cette catégorie de systèmes, les approches qui permettent la construction de réseaux à partir de FSAs (*Finite State Automata* - Automates à états finis), ainsi que l'extraction de FSAs à partir des réseaux, font partie d'un des principaux axes de recherche de ce domaine [OML 94, OML 93, OML 96a, FRA 95, FRA 96a, FRA 96b, FRA 92, FRA 91]. Plus récemment, nous trouvons aussi des recherches qui ont étendu ces systèmes aux automates flous (*fuzzy-finite state automata*) [OML 96b]. Cependant, comme nous avons déjà dit antérieurement, nous avons choisi de ne pas travailler sur les réseaux récurrents et par conséquence nous n'avons pas approfondi nos recherches dans cette direction. Toutefois, cette voie de recherche semble être très intéressante. Dans un futur proche, nous comptons pouvoir étudier les possibilités de faire évoluer notre système INSS afin de pouvoir exploiter certaines des techniques utilisées dans cette approche.

3.4 CONCLUSION SUR L'ETUDE DES SHNS

Dans les sections antérieures, nous avons orienté le sujet de discussion de cette thèse vers l'étude des systèmes experts intelligents pour la classification. Notre intérêt s'est porté principalement sur un type particulier de systèmes, les SHNS (Systèmes Hybrides Neuro-Symboliques) avec un mode d'intégration par co-traitement. Dans ce cadre, nous avons présenté dans ce chapitre un aperçu des SHNS, de leur classification et de leurs propriétés. Nous avons aussi mis en valeur les processus de transfert de connaissances entre les modules symbolique et connexionniste, et nous avons fini en présentant d'une façon plus détaillée, les deux systèmes hybrides qui nous ont le plus influencé dans nos recherches : le système SYNHESYS et le système KBANN. Ces deux systèmes présentent des propriétés très intéressantes que nous avons envisagé d'incorporer dans notre système, tout en améliorant leurs points faibles et leurs limitations.

Nous avons essayé de présenter un état de l'art des SHNS dans ce dernier chapitre. Malheureusement, il n'est pas possible d'aborder ici un thème si étendu d'une façon exhaustive et complète. Nous avons choisi de diriger nos études sur certains aspects particuliers des SHNS et de faire une analyse plus complète des systèmes SYNHESYS et KBANN. Ainsi, nous laissons au lecteur la possibilité de consulter d'autres ouvrages plus généraux sur certains thèmes plus vastes non abordés complètement par nous. Le premier exemple, est celui des publications de O. Boz [BOZ 95], de R. Andrews [AND 95] et de L. Decloedt [DEC 96] qui abordent le sujet des transferts de connaissances d'une façon plus complète. En ce qui concerne l'extraction de règles à partir des réseaux, Andrews décrit plusieurs méthodes qui sont classées en deux groupes : *méthodes decompositionnelles* (extraction de règles à partir de l'analyse de chaque unité et de ses composants - poids de connexions et seuil) et *méthodes pédagogiques* (extraction de règles à partir de l'étude du comportement global du réseau sans considérer ses composants). Dans les sections antérieures, nous ne décrivons que des méthodes d'extraction de règles du type decompositionnel. L'autre exemple, est constitué par les études sur l'intégration de systèmes hybrides, qui forment un thème assez complexe et qui est encore un vaste domaine de recherche, tel que l'a bien montré le projet MIX et les chercheurs qui ont participé de ce projet [HIL 94a, ORS 95, LAL 96].

4. SYSTEME INSS

Dans le chapitre précédent, nous avons présenté différents systèmes hybrides neuro-symboliques, suivis d'une discussion à propos des fonctionnalités proposées dans deux systèmes en particulier : SYNHESYS et KBANN. Nous avons essayé de tirer profit des aspects intéressants présentés par ces deux systèmes, de façon à créer un nouveau SHNS plus performant. Ce nouveau système que nous avons développé a été nommé ***INSS*** pour ***Incremental Neuro-Symbolic System***.

Le système INSS est un système hybride neuro-symbolique qui utilise un mode d'intégration par co-traitement et qui possède des modules capables de réaliser le transfert de connaissances entre les deux modules symbolique et connexionniste (voir Figure 32). Nous l'avons conçu sous l'environnement UNIX (Sun Solaris), et plus récemment il a été aussi porté sur des machines du type IBM-PC. Il a été développé initialement au sein du laboratoire LIFIA (Laboratoire d'Informatique Fondamentale et d'Intelligence Artificielle de l'IMAG/INPG, qui a été démembré en 1995 donnant l'origine à d'autres laboratoires) et actuellement il reste toujours en développement au sein de l'équipe Réseaux d'Automates du laboratoire LEIBNIZ. Le système INSS est un outil pour la construction de systèmes experts. Ses principales applications sont dans le domaine des problèmes de classification, de l'aide au diagnostic, ou de l'aide à la décision. Cependant, il a été aussi employé dans des tâches diverses, telles que le contrôle de robots autonomes et l'étude de modèles cognitifs du comportement humain (voir Chapitre 5).

MS $\Rightarrow$ transfert $\Rightarrow$ MC (Transformation de Règles en Réseau)
MS $\Leftarrow$ transfert $\Leftarrow$ MC (Explicitation de Règles du Réseau)

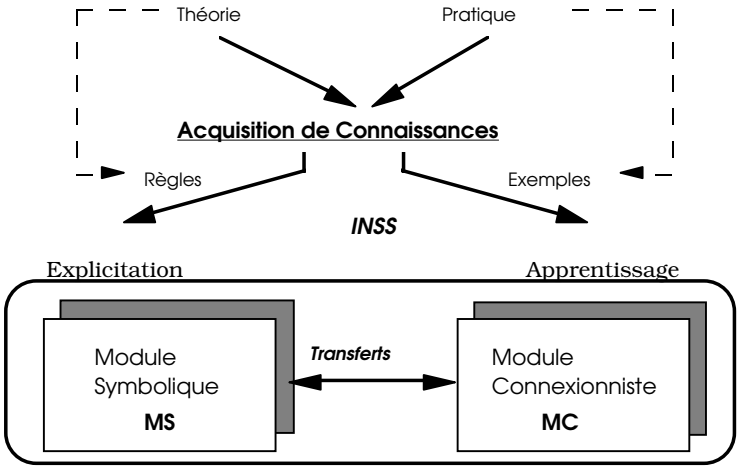


Figure 32 - Schéma Global du Système Hybride INSS

Le système INSS a été inspiré par le modèle développé par G. Towell pour les systèmes KBANN [TOW 91]. Il essaie d'améliorer leurs points faibles en ajoutant de nouvelles propriétés. Au contraire du système KBANN qui utilise l'algorithme de la Rétro-Propagation [RUM 86b], fondé sur des réseaux statiques, INSS utilise la méthode d'apprentissage Cascade-Correlation (CasCor) [FAH 90]. Cet autre algorithme permet l'ajout de nouvelles unités (neurones) au réseau pendant l'apprentissage - c'est une méthode constructive. Notre approche nous a permis d'obtenir un réseau capable de faire évoluer sa structure, ainsi que ses connaissances, tout en restant parfaitement intégré dans le cadre d'un système hybride neuro-symbolique. Par conséquent, le système INSS a l'avantage de permettre un apprentissage incrémental et aussi une extraction incrémentale de règles. Les principales améliorations de notre système par rapport à ses prédécesseurs, sont : l'utilisation de l'algorithme Cascade-Correlation ; l'extraction incrémentale de connaissances ; et la validation des nouvelles connaissances acquises par le système. Tout cela nous a permis d'implémenter un *système hybride neuro-symbolique d'acquisition constructive de connaissances.* Le système INSS constitue donc une nouvelle approche de système d'apprentissage automatique.

Le système INSS est un système hybride qui permet le transfert de *"connaissances théoriques"*, représentées par un ensemble de règles symboliques, d'un module symbolique (MS) à un module connexionniste (MC). Pour réaliser ce transfert on utilise un convertisseur capable

de transformer des règles symboliques en réseau de neurones (ensemble d'unités liées par des connexions pondérées). Le réseau obtenu de cette façon pourra subir un apprentissage supervisé à partir d'un ensemble d'exemples, qui constitue ce qu'on appelle les *"connaissances pratiques"*. Ainsi, on va pouvoir réaliser un raffinement des connaissances théoriques antérieurement acquises. Après l'amélioration des connaissances du réseau par apprentissage de la base de connaissances pratiques, on peut appliquer à ce réseau un algorithme d'explicitation de règles. Cet algorithme va extraire les règles qui représentent les nouvelles connaissances acquises par le réseau. Ces règles peuvent être réinsérées dans le module symbolique, bouclant ainsi la boucle.

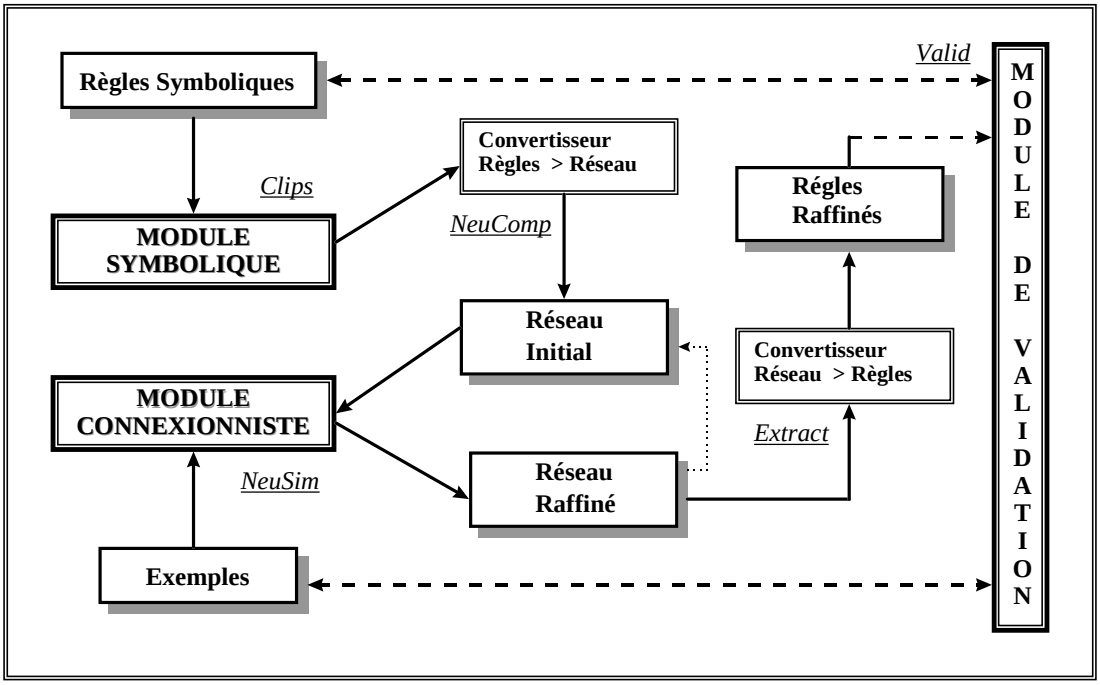


Figure 33 - L'Acquisition Constructive de Connaissances dans le Système INSS

Le système INSS est composé de deux modules principaux, le Module Symbolique et le Module Connexionniste, et en plus des modules de transfert et de validation de connaissances. La Figure 33 et la Figure 34 présentent un schéma des composants du système INSS. Le Module Symbolique est le seul module d'INSS qui n'a pas été développé par nous (voir Section 4.1). Toutes les autres composantes du système ont été réalisées par nous, à savoir :

- *NeuComp* : il réalise l'insertion (traduction / compilation) de règles dans un réseau connexionniste;

• *NeuSim* : c'est le Module Connexionniste proprement dit du système INSS. Il est responsable de la simulation et de l'apprentissage des réseaux;

• *Extract* : il permet d'extraire de règles à partir des réseaux connexionnistes,

• *Valid* : il fait la validation des nouvelles connaissances acquises par rapport aux anciennes. Il utilise tout l'ensemble de connaissances disponible sur le problème (règles ⇔ exemples).

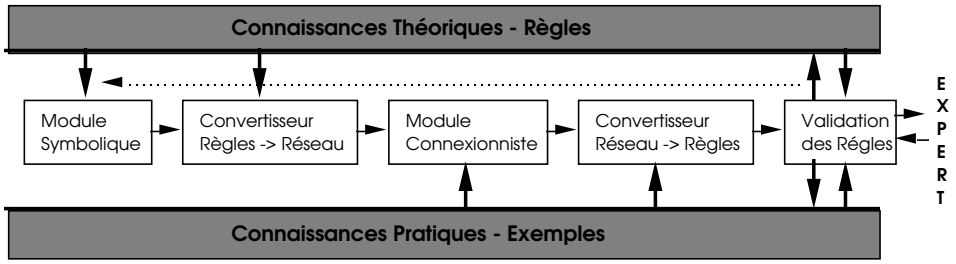


Figure 34 - Les Connaissances dans le Système INSS

Il est important de bien voir que le système comporte deux boucles principales de fonctionnement. Une première, la "petite boucle", correspond à l'entraînement du réseau connexionniste. Comme nous le verrons par la suite, dans cette boucle les nouvelles connaissances présentées au réseau sont gelées au fur et à mesure qu'elles sont apprises. La deuxième boucle, la "grande boucle", correspond, elle, au parcours complet de la chaîne de traitement des connaissances dans le système : compilation, apprentissage, extraction, validation. Dans cette deuxième boucle, les connaissances ne sont pas forcément gelées. Au terme de la validation, on peut arriver à la conclusion que de nouvelles règles doivent remplacer des règles initiales qui s'avèrent non valides. Dans ce cas il faut de nouveau compiler la base de règle modifiée en conséquence. On obtient alors un nouveau réseau qu'il faut à son tour soumettre à l'apprentissage.

Chacun des composants du système INSS va être présenté plus en détails dans les sections suivantes de ce chapitre.

4.1 MODULE SYMBOLIQUE

Le moteur d'inférence symbolique utilisé dans INSS est basé sur le système CLIPS (*C Language Integrated Production System*) [GIA 93,CUL 93]. CLIPS est un langage outil pour la création de systèmes experts qui a été développé au *Lyndon B. Jonhson Space Center* de la *NASA*, au sein de la STB (*Software Technology Branch*, *ex- Artificial Intelligence Section*), depuis 1984. Nous avons utilisé la version 6.0 de CLIPS disponible pour l'environnement Unix, mais qui est aussi disponible en versions adaptées aux machines du type IBM-PC ou Macintosh.

Comme son nom indique, CLIPS a été écrit dans le 'Langage C', ceci entraînant une grande portabilité et une bonne vitesse d'exécution. Ses qualités et fonctionnalités font qu'il est utilisé dans plusieurs sites de la NASA, dans de nombreux bureaux militaires et fédéraux aux Etats-Unis, ainsi que dans certaines universités et entreprises. Ce système est distribué librement aux universités et aux centres de recherche, et on peut même avoir accès au code source de tout le logiciel. Le système CLIPS est à l'origine d'autres produits dérivés, à savoir : le *Fuzzy-CLIPS* [IIT 96] développé au IIT (Institute for Information Technology) - CNRC, Canada ; le *JESS* (Java Expert System Shell) [SAN 97,WAT 97] développé au Sandia National Laboratories de Livermore aux Etats-Unis ; et l'*ECLIPSE* [HAL 97] développé par The Halley Enterprise aux Etats-Unis.

Le système CLIPS présente l'avantage de combiner trois méthodologies différentes de représentation de connaissances et de manipulation des informations. Originellement, il permettait seulement l'utilisation de règles symboliques avec un moteur d'inférence par chaînage avant basé sur l'algorithme Rete [FOR 82]. Il a ensuite acquis dans ses versions ultérieures les possibilités de programmation procédurale (comme en C ou en Lisp). Puis finalement la représentation et la manipulation de connaissances utilisant le concept d'objets ont été intégrées au système. Donc CLIPS est un outil très puissant qui permet la représentation de connaissances par des règles et par des objets, ainsi que la manipulation de ces informations à travers l'implémentation d'un mécanisme d'inférence par chaînage avant ou à travers de la programmation procédurale.

```
(clear)
(reset)
(set-strategy depth)

(assert (cas 1 FALSE FALSE Out_False))
(assert (cas 2 TRUE  FALSE Out_True))
(assert (cas 3 FALSE TRUE  Out_True))
(assert (cas 4 TRUE  TRUE  Out_False))

(defrule xor
   ?nfact <- (cas ?x ?x1 ?x2 ?Out)
   (test (and (or ?x1 ?x2) (not (and ?x1 ?x2))) )
  =>
   (printout t "Cas: " ?x " # Clips: Out_True # Exemple: " ?Out crlf)
   (retract ?nfact) )

(defrule not-xor
   ?nfact <- (cas ?x ?x1 ?x2 ?Out)
   (test (not (and (or ?x1 ?x2) (not (and ?x1 ?x2)))) )
  =>
   (printout t "Cas: " ?x " # Clips: Out_False # Exemple: " ?Out crlf)
   (retract ?nfact) )

(facts)
(run)
```

Figure 35 - Exemple de programme en CLIPS (Fonction Xor)

Dans le cadre du système INSS, nous n'avons utilisé qu'une partie des potentialités de CLIPS. Ce système a servi uniquement comme moteur d'inférence de notre SHNS et dans la réalisation du processus de validation de connaissances. La syntaxe utilisée par CLIPS pour décrire les règles et les faits (les exemples) est un peu différente de la syntaxe que nous avons utilisée dans les outils que nous avons développés nous-mêmes. La conversion d'un format à l'autre de description des règles et des exemples peut être faite facilement à travers l'utilisation de convertisseurs des fichiers utilisés par CLIPS et par INSS. La Figure 35 montre un exemple de fichier CLIPS décrivant la base de connaissances associée au problème du XOR (Ou Exclusif). Dans l'Annexe A, nous présentons des exemples complets des fichiers CLIPS et INSS appliqués à ce même problème.

Nous allons centrer nos discussions plus sur les autres modules d'INSS (NeuComp, NeuSim, Extract et Valid) que sur le Module Symbolique, car ce dernier module est un système bien développé et étudié, et l'on peut trouver une vaste documentation sur le sujet. De plus, le module NeuSim du système INSS peut être utilisé à la place de CLIPS, car une fois que les règles sont insérées dans le Module Connexionniste, celui-ci peut être utilisé comme un moteur d'inférence symbolique.

4.2 INSERTION DE CONNAISSANCES A PRIORI

Le module *NeuComp* (Neural Compiler) permet d'obtenir un RNA à partir d'un ensemble de règles symboliques. La transformation de règles en réseau suit les mêmes principes que dans le système KBANN (voir Section 3.3.2.1). On obtient un réseau où sont spécifiés les connexions et ses poids respectifs. Ce réseau est construit de façon à obtenir par son activation un comportement qui correspond exactement aux résultats qu'on peut obtenir à partir de l'utilisation de la base de règles d'origine. Nous avons utilisé une syntaxe très proche de celle employée dans la représentation des règles symboliques de KBANN, comme on peut le constater dans l'exemple présenté ci-dessous.

```
% NeuComp File: Xor.Symb

$Features:
 X1:binary;
 X2:binary.
$End_Feat.

$Rules:
 X1_OR_X2   <- X1;
 X1_OR_X2   <- X2;
 X1_AND_X2  <- X1,X2;
 XOR        <- X1_OR_X2,Not(X1_AND_X2).
$End_rules.

$End.
```

Figure 36 - Règles symboliques utilisées par NeuComp

L'exemple ci-dessus montre des propositions d'ordre 0 décrites selon la syntaxe acceptée par NeuComp. Les règles qui ont les mêmes conséquents représentent des disjonctions ('ou' logique) entre ses antécédents, et les conjonctions ('et' logique) sont représentées par des antécédents séparés par des virgules. Donc, les règles représentées dans l'exemple présenté ci-dessus peuvent être interprétées de la façon suivante :

Si (X1 ou X2) et non (X1 et X2) Alors XOR est Vrai.

Nous avons étendu le type de règles symboliques accepté par NeuComp de façon à pouvoir travailler avec des règles de "plus haut niveau", c'est-à-dire, avec des règles d'ordre 0^+ (inclusion de la notion d'intervalle et utilisation de prémisses formées par des valeurs continues). Ainsi, Neucomp accepte aussi des règles de production avec des comparaisons du type suivant :

<Attribut> <opérateur> <Valeur> ou <Attribut> <opérateur> <Attribut>

(en utilisant des opérateurs de comparaison tels que : Plus-Grand-Que, Plus-Petit-Que, Egal)

Nous allons décrire les processus de conversion de ce type de règles vers des réseaux de type PMC dans la Section 4.2.1. Afin de donner une meilleure description des règles symboliques que nous utilisons avec NeuComp, la Figure 37 montre un résumé de la syntaxe et des différents type de règles acceptées tandis que la Figure 38 montre un exemple pratique plus complet d'un fichier de règles symbolique d'INSS.

```
Conséquent <- Antécédent [ , Antécédent, ... ] ;
Conséquent <= Antécédent [ , Antécédent, ... ] ;
Conséquent <- Fonction ( Antécédent ), Fonction ( Antécédent, Liste-de-Paramètres );
Conséquent <- Antécédent (True), Antécédent (False);

où,

'<-' : Représente une règle adaptable (peut être modifiée au cours de l'apprentissage)
'<=' : Représente une règle statique (n'est pas modifiée au cours de l'apprentissage)
Fonction : Choisie parmi un des opérateurs suivants
            Equal, In, In_Range, Ntrue, NofM, MofN, Not
            LT, Less_Than, LTOE, Less_Or_Equal,
           GT, Greater_Than, GTOE, Greater_Or_Equal
```

Figure 37 - Aperçu de la syntaxe utilisée par Neucomp

La Figure 38 montre aussi que NeuComp accepte plusieurs types de données différents comme entrées des réseaux : type 'binary' (valeurs booléennes), type 'nominal' (valeurs discrètes), type 'range' (valeurs continues), type 'continuous' (série ordonnée de valeurs continues) et type 'ordered' (série ordonnée de valeurs discrètes). Les variables du type 'binary' et 'range' sont codées

dans une seule unité d'entrée, tandis que les variables du type 'nominal' nécessitent une unité d'entrée pour coder chaque valeur nominale (e.g. une variable d'entrée couleur, pouvant assumer trois valeurs, rouge, verte ou bleue, va être codée par trois unités d'entrée : couleur#rouge, couleur#verte, couleur#bleue ; seulement une des ces trois entrées va pouvoir être activée à la fois). Les variables du type 'ordered' et 'continuous' n'ont pas été beaucoup étudiées par nous jusqu'à présent, car elles peuvent aussi être représentées en utilisant respectivement les types 'nominal' et 'range' (ce dernier avec l'utilisation de la fonction In_Range).

```
% NeuComp File: Example.Symb

#define MINVALUE -40.0
#define MAXVALUE  40.0

$Features:
  respiration : binary;
  heart       : nominal    # 2 : beat, stopped;
  temperature : range          : [0,45];
  extern_temp : range          : [MINVALUE, MAXVALUE];
  pulse       : continuous # 3 : low[0,30], medium[30,60], high[60,90];
  day         : ordered    # 7 : mon, tue, wed, thu, fri, sat, sun.
$End_Feat.

$Rules:
  dead  <= NOT (heart(beat));
  alive <= heart(beat), respiration(yes);
  dead  <- respiration(false);
  dead  <- EQUAL(temperature, extern_temp);
  alive <- heart(beat), NOT(dead);
  alive <- IN_RANGE(temperature, 35, 40), pulse(medium);
  alive <- NTRUE(2, heart(beat), pulse(medium), respiration(yes));
  alive <- GTOE(temperature, 35);
  cold  <- Less_Than(extern_temp, 0.0);
  hot   <- GT(extern_temp, 25.0);
  weekend <- day(sat);
  weekend <- day(sun);
  have_a_good_time <- weekend, hot, alive.

#include "other-rules.symb"

$End_rules.
$End.
```

Figure 38 - Exemple de fichier de règles symboliques (structures syntaxiques)

Le module NeuComp utilise plusieurs fichiers : des fichiers d'entrée (*ccfg* : spécifie les paramètres de la compilation ; *symb* : base de règles symboliques) et des fichiers de sortie (*wts* : poids des connexions des unités du réseau compilé ; *top* : topologie du réseau et symboles associés à chaque unité ; *rep* : rapport d'accompagnement du processus de compilation ; *gnup* : fichier graphique compatible avec 'gnuplot' [LIA 94, KON 96], il affiche la topologie du réseau obtenu ; *tmp* : fichier temporaire crée par le pré-processeur). Le fichier des paramètres de

compilation nous permet de spécifier les options de compilation, selon l'exemple présenté ci-dessous :

```
%
% >> Neucomp : Options de Compilation <<
% File: Exemple.ccfg
%
$Flags [DGPO]
$WF    4.0
%
% Flags:
%    D - Afficher les tables de symboles créées pendant la compilation
%    G - Créer un fichier graphique avec la topologie du réseau (compatible avec gnuplot)
%    P - Utiliser le pré-processeur du langage C (cpp - directives: #include et #define)
%    C - Connecter toutes les entrées et/ou les unités du réseau entre deux couches
%    I  - Créer seulement la topologie du réseau et initialiser les poids aléatoirement
%    O - Créer les entrées dans le même ordre utilisé pour les déclarer dans 'features'
%
% WF:
%    Valeur du "Weight_Factor" (valeur constante 'W' utilisée dans KBANN)
%
```

Figure 39 - Fichier de configuration du compilateur NeuComp

Les fichiers décrivant des poids (*.wts) et la topologie du réseau (*.top) vont ensuite pouvoir être lus par NeuSim. Ils établissent ainsi une configuration initiale du réseau avant de commencer l'apprentissage à partir de la base d'exemples. Dans l'Annexe B, nous présentons des exemples de chacun de ces fichiers.

4.2.1 Compilation de Règles d'Ordre 0^+

Une des principales contributions de nos recherches par rapport au modèle proposé par Towell [TOW 91], est l'extension de la capacité de traitement des bases de règles utilisées par KBANN. Nous avons développé une méthode qui permet d'insérer des règles d'ordre 0^+ dans nos réseaux, le tout d'une façon compatible avec le mode de fonctionnement des réseaux du type KBANN. Nous proposons des nouvelles méthodes de compilation de règles en réseaux, afin d'ajouter les fonctions suivantes à nos bases de règles :

1. **Greater_Than (Trait, ValCte, [Paramètres])** : l'unité qui représente cette fonction va s'activer si la valeur de la variable d'entrée *'Trait'* est supérieur à la valeur constante *'ValCte'* spécifiée comme paramètre de cette fonction. Une variante de cette fonction est la fonction

Greater_Or_Equal, dans laquelle la sortie de l'unité est activée seulement si la variable d'entrée *'Trait'* est égale ou supérieur à la valeur constante *'ValCte'*. Les deux fonctions décrites ci-dessus possèdent des noms abrégés, reconnus par le compilateur : GT et GTOE. Cette unité du type Greater_Than possède une seule entrée et une sortie, et la valeur de son poids ainsi que la valeur de son seuil sont obtenues de la façon suivante :

Poids_GT(T,V) = Sensibilité

Seuil_GT(T,V) = -1.0 * (ValCte - Confiance) * Sensibilité

Sortie de l'unité: S = Fct_Sigmoïde (Trait * Poids_GT + Seuil_GT)

Les valeurs de la *'sensibilité'* et de la *'confiance'* sont des paramètres optionnels définis dans la fonction en question. Nous allons décrire ces paramètres et son influence par rapport au comportement des fonctions, à la fin de cette section.

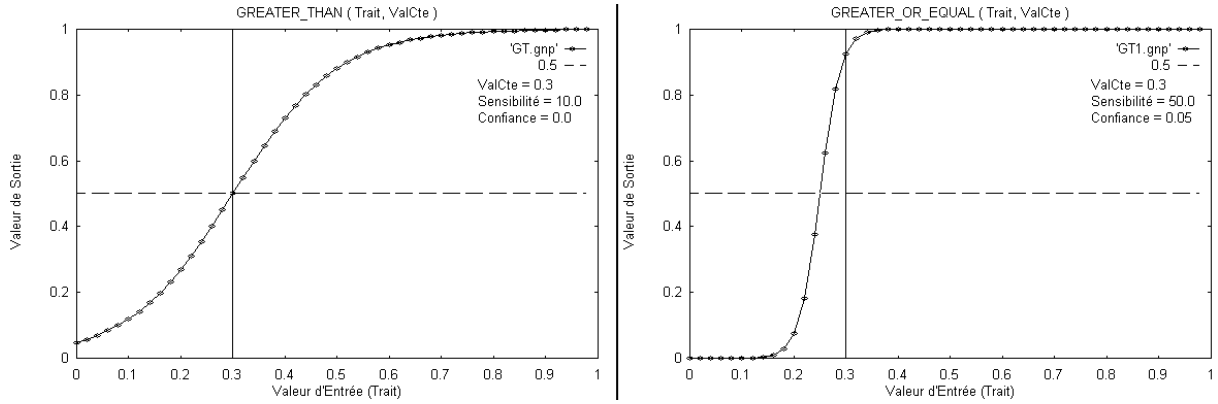


Figure 40 - Fonction Greater_Than (Trait, Valeur)

2. **Greater_Than (Trait1, Trait2, [Paramètres])** : l'unité qui représente cette fonction va s'activer si la valeur de la variable d'entrée *'Trait1'* est supérieure à la valeur de la variable d'entrée *'Trait2'*. Une variante de cette fonction est la fonction *Greater_Or_Equal*, dans laquelle la sortie de l'unité est activée seulement si la variable d'entrée *'Trait1'* est égale ou supérieur à la valeur de la variable d'entrée *'Trait2'*. Les deux fonctions décrites ci-dessus possèdent des noms abrégés, reconnus par le compilateur : GT et GTOE. La seule différence entre cette fonction et la

fonction précédente est au niveau des paramètres, tels que le compilateur va automatiquement déterminer le type de fonction à adopter. Cette unité du type Greater_Than possède deux entrées et une seule sortie, et la valeur de ses poids ainsi que la valeur de son seuil sont obtenus de la façon suivante :

Poids1_GT(T,T) = Sensibilité

Poids2_GT(T,T) = -1.0 * Sensibilité

Seuil_GT(T,T) = Confiance * Sensibilité

Sortie: S = Fct_Sigmoïde(Trait1*Poids1_GT + Trait2*Poids2_GT + Seuil_GT)

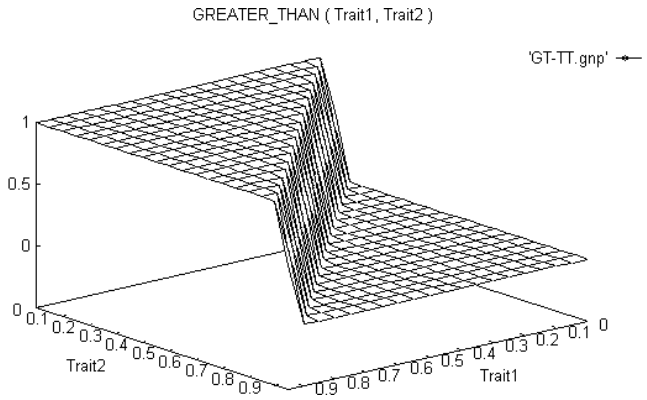


Figure 41 - Fonction Greater_Than (Trait, Trait)

3. **Less_Than (Trait, ValCte, [Paramètres])** : l'unité qui représente cette fonction va s'activer si la valeur de la variable d'entrée *'Trait'* est inférieur à la valeur constante *'ValCte'* spécifiée comme paramètre de cette fonction. Une variante de cette fonction est la fonction *Less_Or_Equal*, dans laquelle la sortie de l'unité est activée seulement si la variable d'entrée *'Trait'* est inférieure ou égale à la valeur constante *'ValCte'*. Les deux fonctions décrites ci-dessus possèdent des noms abrégés, reconnus par le compilateur : LT et LTOE. Cette unité du type Less_Than possède une seule entrée et une sortie, et la valeur de son poids ainsi que la valeur de son seuil sont obtenus de la façon suivante :

Poids_LT(T,V) = -1.0 * Sensibilité

Seuil_LT(T,V) = (ValCte + Confiance) * Sensibilité

Sortie: S = Fct_Sigmoïde (Trait * Poids_LT + Seuil_LT)

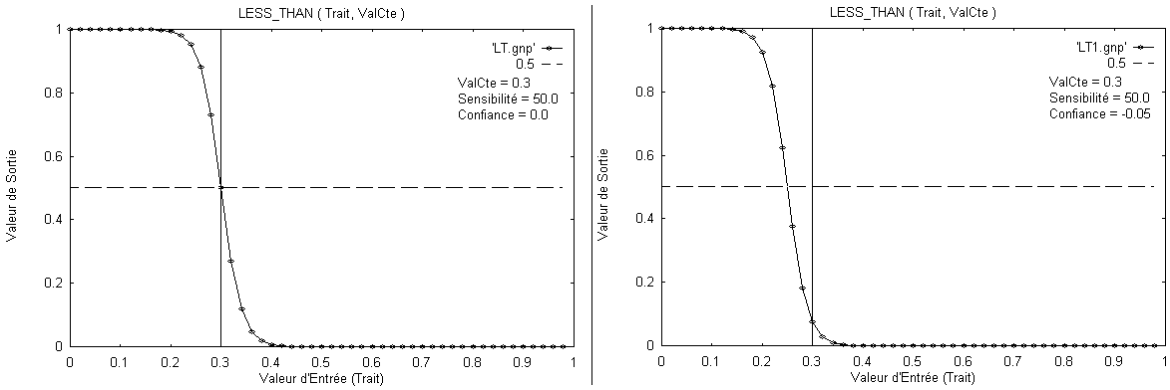


Figure 42 - Fonction Less_Than (Trait, Valeur)

4. **Less_Than (Trait1, Trait2, [Paramètres])** : l'unité qui représente cette fonction va s'activer si la valeur de la variable d'entrée *'Trait1'* est inférieure à la valeur de la variable d'entrée *'Trait2'*. Une variante de cette fonction est la fonction *Less_Or_Equal*, dans laquelle la sortie de l'unité est activée seulement si la variable d'entrée *'Trait1'* est inférieur ou égale à la valeur de la variable d'entrée *'Trait2'*. Les deux fonctions décrites ci-dessus possèdent des noms abrégés, reconnus par le compilateur : LT et LTOE. La seule différence entre cette fonction et la fonction précédente est au niveau des paramètres : le compilateur va automatiquement déterminer le type de fonction à adopter. Cette unité du type Less_Than possède deux entrées et une seule sortie, et la valeur de ses poids ainsi que la valeur de son seuil sont obtenues de la façon suivante :

Poids1_LT(T,T) = -1.0 * Sensibilité

Poids2_LT(T,T) = Sensibilité

Seuil_LT(T,T) = Confiance * Sensibilité

Sortie: S = Fct_Sigmoïde(Trait1*Poids1_LT + Trait2*Poids2_LT + Seuil_LT)

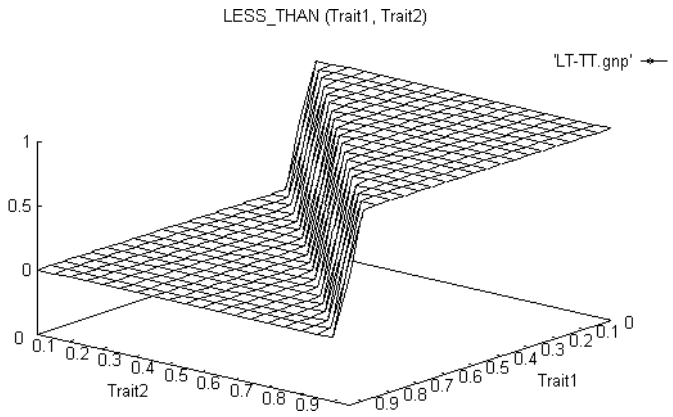


Figure 43 - Fonction Less_Than (Trait, Trait)

5. **In_Range (Trait, ValCte1, ValCte2, [Paramètres])** : l'unité qui représente cette fonction va s'activer si la valeur de la variable d'entrée *'Trait'* est comprise dans l'intervalle spécifié par les valeurs constantes *'ValCte1'* et *'ValCte2'*. La fonction In_Range est réalisé en utilisant la combinaison (et logique) de deux unités *: "Si GT (Trait, ValCte1) et LT (Trait, ValCte2) Alors In_Range (Trait, ValCte1, ValCte2) est vrai"*. Cette fonction possède aussi un nom abrégé, reconnu par le compilateur : IN. Les trois unités qui composent la fonction In_Range sont définies de la façon suivante :

Unité GT (Trait, ValCte1):

Poids_GT(T,V) = Sensibilité

Seuil_GT(T,V) = -1.0 * (ValCte1 - Confiance) * Sensibilité

Sortie GT (Trait, ValCte1) = Fct_Sigmoïde (Trait * Poids_GT + Seuil_GT)

Unité LT (Trait, ValCte2):

Poids_LT(T,V) = -1.0 * Sensibilité

Seuil_LT(T,V) = (ValCte2 + Confiance) * Sensibilité

Sortie LT (Trait, ValCte2) = Fct_Sigmoïde (Trait * Poids_LT + Seuil_LT)

Unité Et (GT, LT):

Poids1_GT(T,V) = WF

Poids2_LT(T,V) = WF

Seuil_GT(T,V) = -1.5 * WF

Sortie Et (GT,LT) = Fct_Sigmoïde (GT(Trait, ValCte1) * Poids1_GT +

LT(Trait, ValCte2) * Poids2_LT + Seuil_GT)

La valeur *'WF'* , utilisée dans une des unités qui représente cette fonction, est une valeur constante spécifiée dans le fichier de configuration du compilateur (*weight_factor*). La valeur du paramètre 'confiance' détermine si l'intervalle considéré doit inclure ou pas les valeurs des extrémités (ValCte1 et ValCte2).

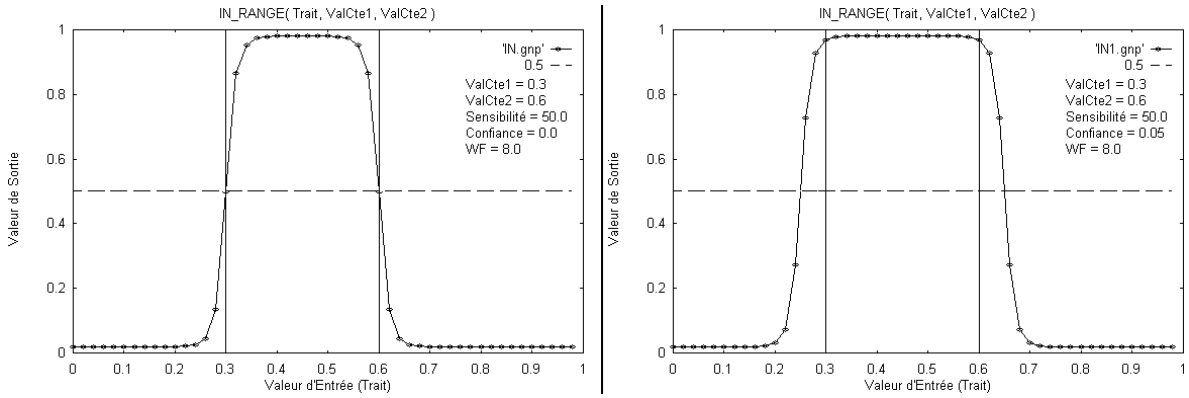


Figure 44 - Fonction Equal (Trait, Valeur)

6. **Equal (Trait, ValCte, [Paramètres])** : l'unité qui représente cette fonction va s'activer si la valeur de la variable d'entrée *'Trait'* est égale à la valeur constante spécifiée par *'ValCte'* . Nous considérons que deux valeurs sont égales s'ils sont "suffisamment proches". La fonction Equal est réalisé en utilisant la combinaison ('et' logique) de deux unités :

"Si GT (Trait, ValCte) et LT (Trait, ValCte) Alors Equal (Trait, ValCte) est vrai"

ou bien,

"Si In_Range(Trait, ValCte, ValCte) Alors Equal (Trait, ValCte)"

Dans ce cas de figure, les fonctions LT et GT doivent obligatoirement inclure la valeur de l'extrémité (ValCte : ValCte1 = ValCte2), sinon la fonction va retourner toujours une valeur de sortie 'fausse' (inactive). Le paramètre 'confiance' détermine le dégréé de proximité qu'on doit avoir par rapport à la valeur constante en question. Les trois unités qui composent la fonction Equal sont définies de la façon suivante :

Unité GT (Trait, ValCte):

Poids_GT(T,V) = Sensibilité

Seuil_GT(T,V) = -1.0 * (ValCte - Confiance) * Sensibilité

Sortie GT (Trait, ValCte) = Fct_Sigmoïde (Trait * Poids_GT + Seuil_GT)

Unité LT (Trait, ValCte):

Poids_LT(T,V) = -1.0 * Sensibilité

Seuil_LT(T,V) = (ValCte + Confiance) * Sensibilité

Sortie LT (Trait, ValCte) = Fct_Sigmoïde (Trait * Poids_LT + Seuil_LT)

Unité Et (GT, LT):

Poids1_GT(T,V) = WF

Poids2_LT(T,V) = WF

Seuil_GT(T,V) = -1.5 * WF

Sortie Et (GT,LT) = Fct_Sigmoïde (GT(Trait, ValCte) * Poids1_GT +
LT(Trait, ValCte) * Poids2_LT + Seuil_GT)

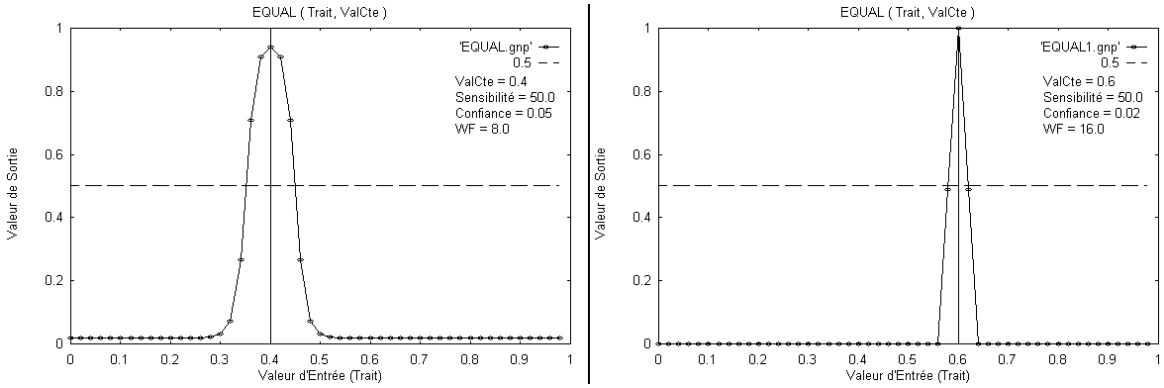


Figure 45 - Fonction Equal (Trait, Valeur)

7. **Equal (Trait1, Trait2, [Paramètres])** : l'unité qui représente cette fonction va s'activer si la valeur de la variable d'entrée *'Trait1'* est égale à la valeur de la variable d'entrée *'Trait2'*. Nous considérons que deux valeurs sont égales s'ils sont "suffisamment proches". La seule différence entre cette fonction et la fonction précédente est au niveau des paramètres, où le compilateur va automatiquement déterminer le type de fonction à adopter. La fonction Equal est réalisée en utilisant la combinaison ('et' logique) de deux unités : *"Si GT (Trait1, Trait2) et LT (Trait1, Trait2) Alors Equal (Trait1, Trait2) est vrai"*. Dans ce cas de figure, les fonctions LT et GT doivent obligatoirement accepter une certaine marge d'erreur (proximité), sinon la fonction va retourner toujours une valeur de sortie 'fausse' (inactive). Le paramètre 'confiance' détermine le degré de proximité que doit avoir une variable par rapport à l'autre. Les trois unités qui composent la fonction Equal sont définies de la façon suivante :

Unité GT (Trait1, Trait2):

Poids1_GT(T,T) = Sensibilité

Poids2_GT(T,T) = -1.0 * Sensibilité

Seuil_GT(T,T) = Confiance * Sensibilité

Sortie GT (Trait1, Trait2) = Fct_Sigmoïde(Trait1 * Poids1_GT +
Trait2 * Poids2_GT + Seuil_GT)

Unité LT (Trait1, Trait2):

Poids1_LT(T,T) = -1.0 * Sensibilité

Poids2_LT(T,T) = Sensibilité

Seuil_LT(T,T) = Confiance * Sensibilité

Sortie LT (Trait1, Trait2) = Fct_Sigmoïde(Trait1 * Poids1_LT +
Trait2 * Poids2_LT + Seuil_LT)

Unité Et (GT, LT):

Poids1_GT(T,V) = WF

Poids2_LT(T,V) = WF

Seuil_GT(T,V) = -1.5 * WF

Sortie Et (GT,LT) = Fct_Sigmoïde (GT(Trait1, Trait2) * Poids1_GT +
LT(Trait1, Trait2) * Poids2_LT + Seuil_GT)

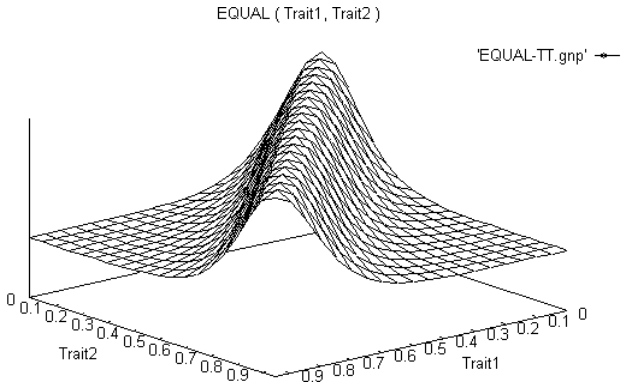


Figure 46 - Fonction Equal (Trait, Trait)

Le paramètre 'Sensibilité' permet d'ajuster la pente de la courbe obtenue à la sortie des unités qui représentent les fonctions décrites ci-dessus. La valeur de la sensibilité doit être une valeur positive égale ou supérieure à 1.0. Nous fixons cette valeur égale à 1.0 dans le cas où le paramètre sensibilité n'est pas indique dans la fonction. Une valeur de sensibilité proche de 1.0 va entraîner l'obtention d'une fonction avec une sortie quasi linéaire, tandis qu'une valeur de sensibilité plus

grande (e.g. sensibilité=100.0) va entraîner l'obtention d'une fonction avec une sortie plus abrupte, qui passe d'un état à l'autre très rapidement (faux ⇒ vrai et vice versa). La Figure 47 montre un exemple des différents comportements obtenus à la sortie d'une fonction du type Greater_Than, étant donné des valeurs différentes attribuées au paramètre 'Sensibilité'.

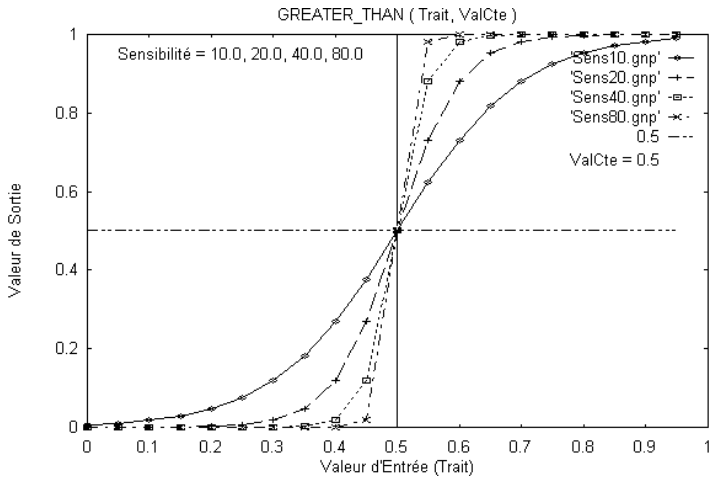


Figure 47 - Sortie de la fonction Greater_Than par rapport à la valeur de la sensibilité

Le paramètre 'Confiance' permet d'ajuster l'erreur acceptée à la sortie des unités (degré de proximité utilisé dans les comparaisons) qui représentent les fonctions décrites ci-dessus. La confiance permet de modifier la sortie des fonctions de façon à inclure ou à exclure les valeurs qui délimitent le changement d'état de la réponse. Ainsi, on peut transformer une fonction du type Greater_Than de façon à obtenir une fonction Greater_Than_Or_Equal, ou une fonction Greater_Than_And_Not_Equal, selon la valeur choisie de la confiance. La valeur de la confiance doit être une valeur positive égale ou supérieure à 0.0. Nous fixons cette valeur égale à 0.0 dans le cas où le paramètre confiance n'est pas indiqué dans la fonction. L'utilisation d'une valeur de confiance égale à 0.0 va entraîner l'obtention d'une fonction avec une valeur de sortie vraie seulement si les valeurs d'entrée satisfont parfaitement la fonction de comparaison définie ; tandis qu'une valeur de confiance plus grande va entraîner l'obtention d'une fonction avec une sortie plus souple, permettant de comparer les valeurs d'entrée avec une certaine marge d'erreur. Par exemple, si la valeur de confiance est égale à 0.1 dans une fonction du type Equal, nous allons considérer les entrées comme étant égales jusqu'à un écart maximum de 0.1 entre leurs valeurs. La Figure 48 montre un exemple des différents comportements obtenus à la sortie des fonctions

du type Greater_Than et In_Range, étant donné des valeurs différentes attribuées au paramètre 'Confiance'.

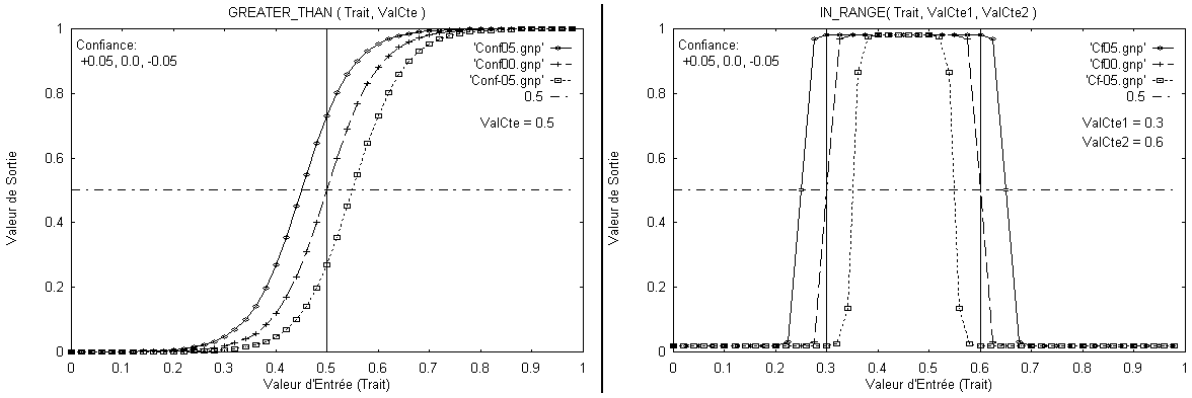


Figure 48 - Sortie des fonctions Greater_Than et In_Range par rapport à la valeur de la confiance

Le paramètre 'WF' est utilisé seulement dans les fonctions qui nécessitent la création de plus d'une unité pour obtenir le comportement de sortie de la fonction, c.-à-d. dans les fonctions du type In_Range et Equal. Ce paramètre fonctionne exactement de la même façon que la valeur constante 'W' utilisée dans les réseaux KBANN, et son rôle est d'augmenter l'écart entre les deux valeurs extrêmes de sortie de la fonction. La Figure 49 montre un exemple des différents comportements obtenus à la sortie d'une fonction du type In_Range, étant donné des valeurs différentes attribuées au paramètre 'WF'.

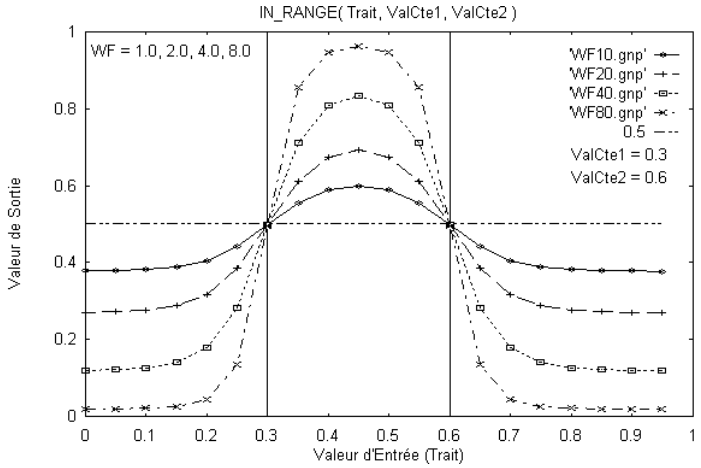


Figure 49 - Sortie de la fonction In_Range par rapport à la valeur de WF

Finalement, toutes ces fonctions que nous venons de présenter sont applicables seulement à des entrées du type 'intervalle' (Feature: range). Il est important de souligner que les paramètres 'Sensibilité', 'Confiance' et 'WF' sont aussi très liées à l'étendue de l'intervalle des valeurs qui sont fournies comme entrées. La méthode actuelle de spécification de ces paramètres n'étant pas très facile à employer, cela nous a amené à la création d'un outil (logiciel 'FnctOut') qui permet de vérifier la courbe de réponse d'une fonctions particulière, étant donné une configuration spécifique des entrées et des paramètres. Cet outil a été employé dans la création des figures présentées dans cette section. Nous envisageons de développer des méthodes capables d'estimer de façon plus automatique les valeurs à mettre dans les paramètres des fonctions.

4.2.2 Discussion sur la Compilation de Règles

Le module de compilation de règles d'INSS nous a permis d'étendre le type de règles que l'on pouvait utiliser sur les réseaux du type KBANN aux règles d'ordre 0^+. Nous avons réalisé des fonctions de conversion de règles en réseau qui nous permettent de travailler directement sur des entrées vraiment continues, ce qui n'était pas le cas des réseaux KBANN. Notre approche de la compilation de règles représente une amélioration significative des capacités de manipulation de connaissances théoriques par les réseaux connexionnistes de type PMC.

Il faut souligner que non seulement des règles d'ordre 0^+ peuvent être introduites dans nos réseaux, mais aussi des 'règles de haut niveau', c'est-à-dire des règles qui expriment des relations directes entre deux entrées. Nous ne sommes plus limités aux règles du type "*'entrée <comparaison> valeur"*, puisque nous pouvons désormais utiliser des règles du type *"entrée <comparaison> entrée"*. La compilation de ces règles d'ordre 0^+ et de haut niveau dans des réseaux peut être contrôlée par plusieurs paramètres, qui permettent le réglage fin de la fonction d'activation obtenue dans la sortie des unités. Nous avons aussi développé des outils qui permettent de visualiser les sorties des unités par rapport aux différents paramètres de compilation utilisés, ce qui va aider beaucoup l'utilisateur dans la tâche de spécification de ce type de règles.

L'extension des règles acceptées par INSS nous a permis d'appliquer notre système à des tâches où la manipulation de valeurs continues est plus importante. Tel est le cas des applications en robotique autonome (voir des exemples pratiques dans la section 5.4 et dans l'Annexe B.2). Toutefois, nous n'avons pas approfondi nos études sur la possibilité d'adapter directement les poids des règles compilées sous la forme d'un réseau, puisque nous les conservons inaltérées dans nos réseaux. Notre choix de conserver les règles inaltérées après la compilation va être discuté dans la section suivante. Donc, l'adaptation directe des règles (adaptation des poids des connexions obtenus par compilation) qui manipulent des valeurs d'entrée continues reste un sujet ouvert à l'étude.

4.3 MODULE CONNEXIONNISTE

Le module connexionniste NeuSim (Neural Simulator) permet de réaliser l'adaptation des poids d'un réseau du type PMC par apprentissage supervisé à partir d'une base d'exemples (fonctionnement en mode d'apprentissage). Ce module permet aussi d'activer un réseau afin d'obtenir la réponse produite à sa sortie, étant donné un ensemble particulier de valeurs présentées en entrée (fonctionnement en mode de consultation). Notre module connexionniste possède aussi des propriétés très intéressantes, telles que la possibilité d'adaptation de la topologie du réseau, ou l'apprentissage à l'aide d'une méthode très performante.

Le choix du modèle de réseau de neurones à utiliser au sein du système hybride INSS a été fondamental pour obtenir un module connexionniste d'acquisition constructive de connaissances. Dès lors que nous cherchions une méthode qui soit adaptée à une méthodologie constructive d'acquisition de connaissances, il nous a fallu un réseau évolutif (capable de faire évoluer son architecture ainsi que ses poids). On trouve dans la littérature plusieurs méthodes de ce type [FIE 94b, CHE 97], parmi lesquelles on peut distinguer trois classes principales:

- les algorithmes constructifs. E.g.: l'algorithme Cascade-Correlation, l'algorithme Tiling, l'algorithme Upstart et les réseaux ARN2.

- les algorithmes de simplification des réseaux. E.g.: les algorithmes d'élagage *'Optimal Brain Damage'* - OBD et *'Optimal Brain Surgeon'* - OBS.
- les algorithmes de recherche de topologie. E.g.: les algorithmes génétiques du type TopGen, Regent et Addemup [OPI 95a].

Nous avons choisi de travailler avec un algorithme du type constructif, car il semble être plus adapté à notre approche d'acquisition constructive de connaissances. Le fait d'avoir choisi ce type d'algorithme pour NeuSim, n'exclut pas la possibilité d'appliquer d'autres algorithmes dans les autres modules du système, comme par exemple pour les algorithmes de simplification qui sont utilisés dans le module d'extraction de règles à partir des réseaux.

Ainsi, nous avons choisi d'utiliser l'algorithme Cascade-Correlation (CasCor) [FAH 90] qui réalise l'adaptation des poids et de la topologie du réseau, contrairement à l'algorithme de la Rétro-Propagation, utilisé dans les réseaux KBANN, qui ne réalise qu'une adaptation des poids. Nous allons présenter l'algorithme CasCor plus en détails dans la section qui suit, et discuter les principaux avantages qu'apporte cette méthode par rapport à la méthode de la Rétro-Propagation [RUM 86].

4.3.1 Méthode d'Apprentissage Cascade-Correlation

L'algorithme CasCor [FAH 90] débute l'apprentissage avec un réseau composé uniquement par des unités d'entrée et de sorties (pas d'unités dans la couche cachée - voir Figure 50). En fait, comme on le verra plus loin, il est aussi possible de commencer l'apprentissage avec un réseau initial plus complexe. Les unités de la couche cachée sont créées petit à petit au cours de l'apprentissage. Cet algorithme réalise en alternance l'adaptation des unités de sortie et l'adaptation des unités de la couche cachée.

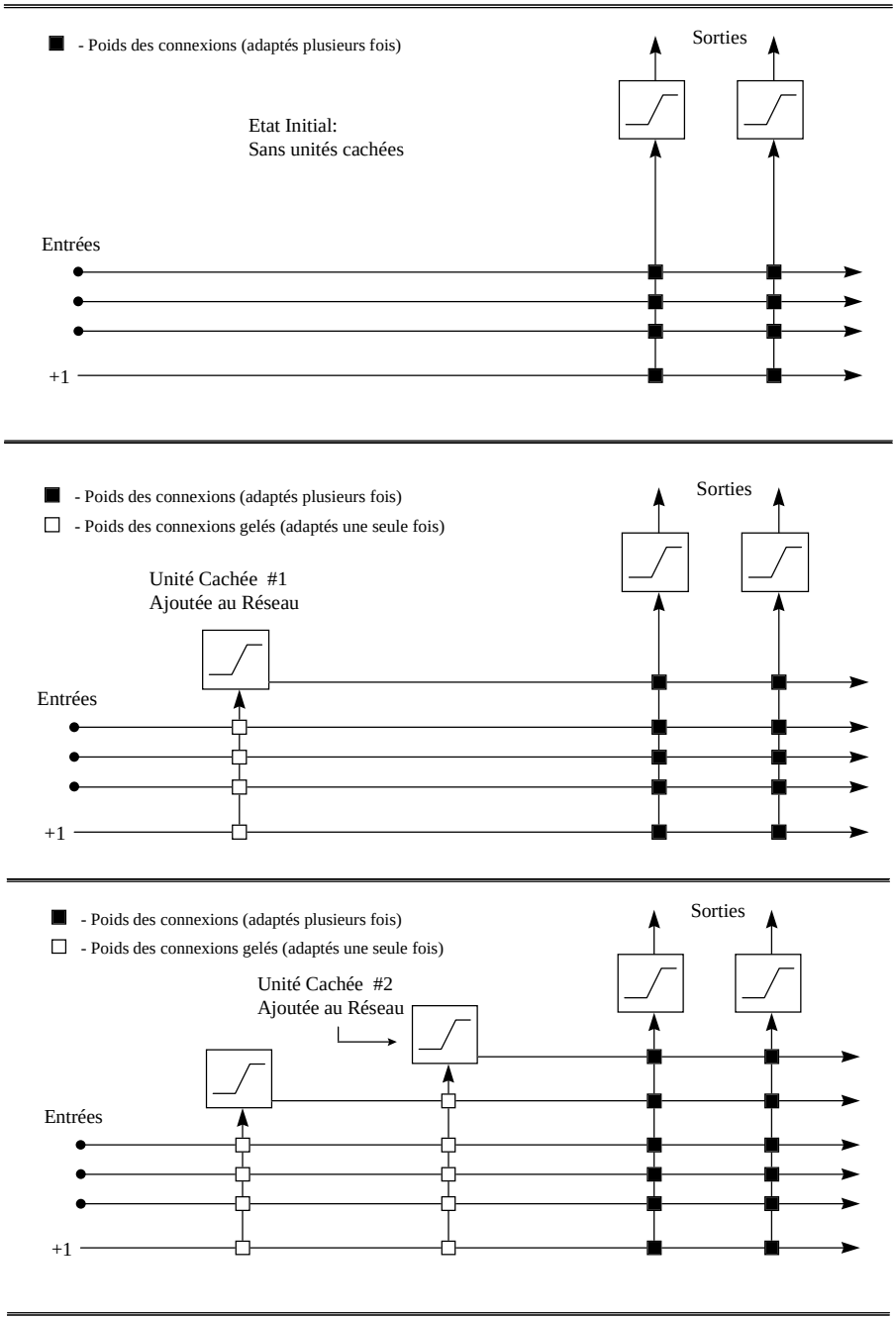


Figure 50 - Architecture des réseaux CasCor (état initial et après l'ajout de deux unités)

Le CasCor commence par l'application de l'algorithme de minimisation de l'erreur des unités de sortie en utilisant la méthode QuickProp [FAH 88]. Cette méthode effectue la minimisation de l'erreur de sortie des unités par rapport aux poids des connexions, en utilisant une méthode de

descente de gradient de l'erreur de deuxième ordre [FAH 88] (voir Eq. 27). L'utilisation de cette méthode d'adaptation des poids permet d'augmenter considérablement la vitesse d'apprentissage.

- Quickprop - Adaptation des poids: (Eq. 27)

$$\Delta W(t) = \frac{S(t)}{S(t-1) - S(t)} . \Delta W(t-1)$$

$S(t) = \frac{\partial E}{\partial w}$ et $S(t-1) = \frac{\partial E}{\partial w}$ dérivés dans le temps courant (t) et précédent (t-1)

Pendant cette étape initiale de l'algorithme CasCor, seuls les poids des unités de sortie sont adaptés. Si l'on atteint un certain niveau d'erreur acceptable, alors l'apprentissage est arrêté et déclaré réussi. Par contre, si l'on ne parvient pas, au bout d'un certain temps prédéterminé (nombre maximal d'époques - *patience*), à minimiser l'erreur résiduelle, alors une nouvelle unité est ajoutée au réseau. L'unité ajoutée reçoit des connexions à partir de toutes les unités d'entrée du réseau et de toutes les unités cachées existantes. Momentanément sa sortie n'est pas reliée au reste du réseau. On adapte alors les poids de ses connexions, en utilisant un algorithme de *maximisation de la corrélation*[1] entre l'erreur résiduelle des unités de sortie et la sortie de l'activation de l'unité ajoutée [FAH 90] (voir Eq. 28). L'algorithme du QuikProp est utilisé cette fois-ci pour adapter les poids de cette unité cachée et ainsi maximiser la covariance entre sa sortie et l'erreur de sortie du réseau. Cette technique de maximisation de la covariance va créer des unités qui seront des détecteurs de traits associés à l'erreur résiduelle du réseau. Une fois que la covariance reste 'stable' pendant un certain nombre d'époques (*patience*), l'unité ajoutée est connectée définitivement au réseau. La sortie de l'unité cachée est connectée aux unités de sorties du réseau avec des poids aléatoires très petits, et les poids d'entrée de cette unité cachée sont "*gelés*". Les poids gelés ne changeront plus et cette unité deviendra un détecteur de traits stable, ce qui évite le problème typique des réseaux à Rétro-Poppagation de l'instabilité des unités (*moving target problem*. - voir section 2.3.3).

[1] La fonction employée est en fait la covariance plutôt que la corrélation.

- CasCor - Calcul de la Covariance: (Eq. 28)

$$Cv = \sum_s \left| \sum_e \left(Sa_e - \overline{Sa} \right) \left(Sr_{es} - \overline{Sr_s} \right) \right|$$

Sa_e: Sortie de l' unité ajoutée pour l' exemple ' e'

$\overline{Sa}$: Moyenne de la sortie de l' unité ajoutée sur tous les exemples

Sr_{es}: Sortie ' s' du réseau pour l' exemple ' e'

$\overline{Sr_s}$: Moyenne de la sortie ' s' du réseau sur tous les exemples

Afin d'éviter d'installer définitivement dans le réseau des unités peu utiles, il est possible d'entraîner plusieurs *unités candidates* en même temps. Les unités cachées candidates sont initialisées avec des poids différents, et ainsi on aura plus de chance de trouver un bon ensemble de poids qui va maximiser la covariance. Seule l'unité qui correspond au maximum de covariance sera retenue.

Après la phase d'adaptation de l'unité cachée et de son ajout au réseau, on recommence l'adaptation des unités de sortie, qui désormais incluent les connexions en provenance de la nouvelle unité ajoutée. Ainsi le cycle d'apprentissage pourra être répété plusieurs fois: (1) adaptation des sorties ; (2) ajout et adaptation d'une unité cachée ; (3) gel des poids de la nouvelle unité ; (4) retour à l'étape d'adaptation des unités de sortie. Ce processus est répété jusqu'au succès, c.-à-d. quand l'erreur de sortie est inférieure à une valeur prédéterminée. On peut aussi aboutir à un échec quand on dépasse une limite maximale d'époques sans arriver à réduire suffisamment l'erreur.

L'algorithme CasCor est très performant. Plusieurs études font état de sa performance supérieure à d'autres algorithmes d'apprentissage [FAH 90, THR 91, SCH 93]. L'algorithme CasCor a beaucoup été étudié, et plusieurs variantes de cet algorithme ont été proposées : Cascade-Correlation du Second Ordre (2CCA) [SJO 91], Cascade-Correlation Modifié [SIM 90], Cascade-Correlation Récurrent [FAH 91], et le *'Sibling/Descendant Cascade-Correlation'* (SDCC) [BAL 94]. L'un des seuls reproches qu'on peut faire à cet algorithme est la profondeur importante que peuvent avoir les réseaux obtenus après l'apprentissage, car chaque nouvelle unité

est ajoutée dans une nouvelle couche. Dans la pratique, on constate que l'algorithme CasCor converge très rapidement et qu'il n'a pas besoin d'ajouter beaucoup d'unités afin de résoudre un problème. La profondeur finale des réseaux ne représente donc pas un facteur nuisible à son utilisation. Par contre, les réseaux créés avec cet algorithme doivent être utilisés plutôt dans des tâches de classification, car les unités de sortie doivent être discrètes (et les unités cachées le sont aussi). *L'application de ce type d'algorithme dans des tâches d'approximation de fonctions doit être envisagée avec beaucoup de précaution.*

4.3.2 Avantages de l'Utilisation de l'Algorithme Cascade-Correlation

Le CasCor apporte des solutions aux problèmes spécifiques des réseaux PMC basés sur l'algorithme de la Rétro-Propagation, décrits dans la Section 2.3.3, et à d'autres problèmes communs à différents types de réseaux. L'algorithme CasCor permet de :

- Construire le réseau au fur et à mesure que le processus d'apprentissage évolue. Donc, il trouve tout seul la bonne structure du réseau qui est adaptée à une application spécifique.

- Utiliser une méthode d'apprentissage incrémentale qui garde figées les connaissances déjà acquises (poids "gelés"), ce qui permet d'ajouter des nouvelles connaissances à une structure qui a déjà appris un ensemble d'exemples. Cette technique est essentielle puisqu'elle nous permet de faire l'insertion préalable de connaissances et de continuer l'apprentissage à partir d'un certain point, le tout en gardant intactes les connaissances initiales.

- Traiter le problème de la paralysie de l'apprentissage (*flat-spot*), grâce à l'implémentation d'une astuce de l'algorithme QuickProp. Il ajoute une valeur constante à la dérivée de la fonction sigmoïde (*'sigmoid-prime offset'*), ce qui permet d'éviter la paralysie.

- Résoudre en grande partie le problème de l'instabilité et de l'oubli catastrophique (*moving target problem*) des unités du réseau, car une fois qu'une unité devient un 'détecteur spécialisé sur un aspect particulier du problème', elle ne changera plus.

- Réaliser un apprentissage très performant, étant donné l'utilisation d'un algorithme de descente du gradient de deuxième ordre - le QuickProp. Il est beaucoup plus rapide que son prédécesseur, la Rétro-Propagation (*BackProp*).

- Simplifier le choix des paramètres, puisque même si le CasCor nécessite encore un réglage de certains paramètres, l'apprentissage est moins sensible aux changements de leurs valeurs. Les paramètres, tels le pas d'apprentissage (*vitesse*) et l'inertie (*momentum*), ne jouent plus un rôle aussi important (et fort) par rapport au processus d'apprentissage.

De plus, le CasCor possède des propriétés très intéressantes en ce qui concerne la modélisation du développement cognitif chez l'homme. On trouve plusieurs travaux réalisés par T. Shultz [MAR 93, SHU 94a, SHU 94b, BUC 94, SHU 96] qui ont mis en valeur les propriétés du modèle CasCor par rapport à la modélisation du développement cognitif chez l'homme. Ces études nous donnent une motivation de plus pour utiliser la méthode CasCor comme algorithme d'apprentissage constructif de base dans notre système. Dans le chapitre des applications, nous allons revenir sur ce sujet, la modélisation du développement cognitif, concernant le problème de la balance étudié par T. Shultz.

4.3.3 L'Algorithme Cascade-Correlation dans NeuSim

Le réseau obtenu par compilation à partir d'un ensemble de règles sert de point de départ pour l'algorithme CasCor. Les règles sont insérées dans un réseau CasCor d'une manière très simple: tous les poids des connexions sont gelés et le réseau initial est introduit comme étant un ensemble d'unités cachées ajoutées antérieurement par l'algorithme CasCor. Le simulateur NeuSim nous permet le choix entre deux possibilités différentes d'insertion d'un réseau pré-construit :

1. Insertion de tout le réseau, où toutes les unités restent figées. Dans ce cas de figure, nous avons besoin de créer une nouvelle couche de sortie afin de préserver inaltérées les unités de sortie du réseau initial. La Figure 51 montre un exemple de cette situation, qui est d'ailleurs le mode de fonctionnement de NeuSim que nous avons le plus étudié.

2. Insertion de tout le réseau, où presque toutes les unités restent figées, à l'exception des unités de sortie. Dans ce cas de figure, les unités de sortie du réseau initial sont utilisées comme des unités de sortie du réseau CasCor, et par conséquent elles sont postérieurement modifiées pendant le processus d'apprentissage. Cette approche nous oblige à considérer que les unités de sortie (conclusions finales de la base de règles) pourront être modifiées. Un réseau ainsi créé est moins complexe qu'un autre créé en utilisant la méthode décrite au paragraphe précédent, mais on ne peut plus considérer que les nouvelles connaissances acquises par apprentissage restent totalement séparées des connaissances de départ.

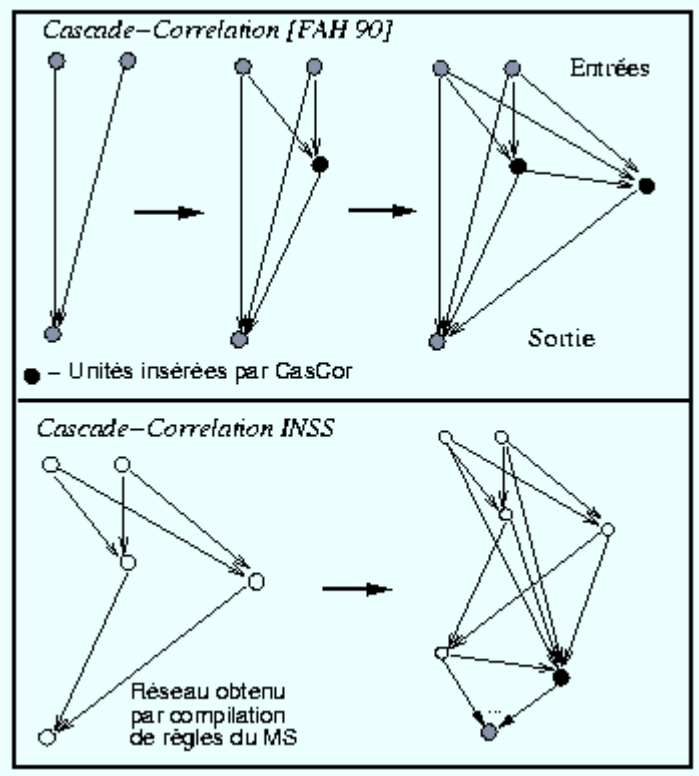


Figure 51 - L'algorithme Cascor et l'évolution de la structure d'un réseau dans INSS

Le simulateur NeuSim offre plusieurs options de configuration, parmi lesquelles on en trouve une qui permet de sélectionner le mode d'insertion d'un réseau initial que nous venons de décrire. D'autres options configurables liées à l'algorithme CasCor sont : la valeur de la patience pour les unités de sortie, la valeur de la patience pour les unités cachées et le nombre d'unités candidates à utiliser pendant l'apprentissage.

4.3.4 Le Module NeuSim en tant que simulateur de réseaux

Il s'avère que le module NeuSim, développé dans le cadre du système INSS et de la méthode connexionniste CasCor, peut être utilisé de manière plus générale comme un simulateur d'autres types de RNAs à base de PMC. Il offre la possibilité de simuler des réseaux avec l'une de ces trois méthodes d'apprentissage : Rétro-Propagation, QuickProp ou CasCor.

NeuSim est un simulateur très flexible qui accepte différents réglages des paramètres de simulation, et qui possède les propriétés nécessaires pour pouvoir être intégré dans le cadre du système hybride neuro-symbolique INSS. Il possède une interface très simple et non graphique, ce qui rend facile son utilisation et aussi son port sur d'autres machines. La simulation peut être faite de façon interactive (interaction par menus) ou de façon plus automatique (utilisation de fichiers de configuration).

Nous allons donner ici un bref aperçu des paramètres de configuration du simulateur et de son fonctionnement. Quand il est en mode d'apprentissage, NeuSim utilise trois fichiers principaux d'entrée : un fichier *.learn (données d'apprentissage supervisé incluant la valeur de la sortie désirée), un fichier *.test (données de test de généralisation), et un fichier *.cfg (spécifie les paramètres et le déroulement de la simulation). Quand il est en mode de rappel, le fichier *.learn n'est pas utilisé. Deux autres fichiers importants sont les fichiers *.top et *.wts, qui décrivent la topologie et les poids respectivement d'un réseau préalablement construit. Ces deux fichiers peuvent être utilisés en mode d'apprentissage (connaissances de départ), seulement si la méthode d'apprentissage choisi est le CasCor. Les fichier *.top et *.wts sont indispensables quand on passe en mode de rappel (sauf si le réseau vient d'être créé et entraîné). Nous présentons des exemples de tous ces fichiers dans l'Annexe B. La Figure 52 présente un schéma des fichiers d'entrée et de sortie utilisés par NeuSim.

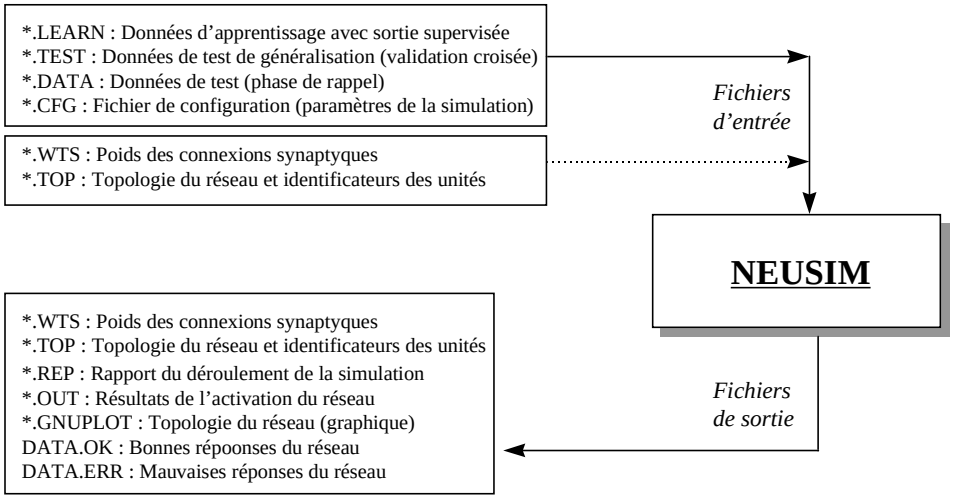


Figure 52 - Fichiers utilisés par NeuSim

Le fichier de configuration de NeuSim contient les informations suivantes (nous listons à côté les principaux paramètres décrits dans ces fichiers - reportez-vous à l'Annexe B pour obtenir plus d'informations sur ces paramètres) :

- *La définition du type de réseau utilisé, de sa structure, et des ses unités* : le nombre d'entrées (NInputs), le nombre de sorties (NOutputs), le choix des valeurs aléatoires (RndRange, RndSeed), le type de fonction d'activation (FunctActv, SigmTemp), le type de sortie (OutType, OutNorm);

- *La définition du type d'algorithme d'apprentissage et de ses paramètres respectifs* : la méthode d'apprentissage (Learning - BackProp, QuickProp ou CasCor), le nombre d'unités cachées (BackProp - réseau à trois couches), le nombre d'unités candidates (CasCor), le pas d'apprentissage (Epsilon), l'inertie (Momentum), les critères d'arrêt (StopCrit, StopErr, MaxEpochs), la façon et l'ordre de présentation des données (EpLearn, DataOrder), le type de sortie utilisée pour évaluer le nombre d'erreurs (NetOutput, MaxErr), 'sigmoid prime offset' (SPOffset), 'weight decay' (WtsDecay), la patience du CasCor (CCOutPat, CCOutECh, CCHidPat, CCHidECh), l'ajout d'une couche de sortie supplémentaire (CCAddOut);

- *La définition du type de suivi de la simulation à enregistrer dans le fichier de rapport et à présenter à l'utilisateur* : le type d'information et la fréquence d'enregistrement dans le fichier de rapport (RepLevel, RepFreq, ErrLevel - fichier *.rep), le type d'information et la fréquence

d'affichage d'information à l'utilisateur (UserRep, UserFreq), l'origine des données utilisées pour créer le fichier de sortie avec les réponses du réseau (OutFile - fichier *.out);

- *La définition des principales tâches à réaliser pendant la simulation* (Task);

```
%%
%% Fichier de Configuration de NeuSim
%% Fichier Xor.Cfg - Apprentissage de la fonction XOR avec
%%                   la méthode Cascade-Correlation
%%
Task      LT       % Tâche: Réaliser l'apprentissage ('L') et après
                   %        le test final de généralisation ('T')
MaxEpochs 1500     % Nombre maximale d'époques
NInputs   2        % Nombre d'entrées du réseau
NOutputs  1        % Nombre de sorties du réseau
Learning  1        % Méthode d'apprentissage: Cascade-Correlation
Cascor    8        % Utiliser CasCor avec 8 unités candidates
CCOutPat  16       % Patience - Unités de sortie (# époques)
CCOutECh  0.01     % Patience - Unités de sortie (erreur)
CCHidPat  16       % Patience - Unités cachées (# époques)
CCHidECh  0.03     % Patience - Unités cachées (erreur)
EpLearn   1        % Apprentissage par paquets (par époques)
RndRange  0.001    % Valeurs aléatoires dans l'intervalle +/- 0.001
Epsilon   0.0001   % Pas d'apprentissage (vitesse de convergence)
MaxErr    0.4      % Sortie binaire - erreur acceptée = 0.4
StopCrit  1        % Arrêt avec 100% de réponses correctes
SPOffset  0.1      % Valeur du 'Sigmoid prime offset'
RepLevel  128      % Présenter les résultats finaux
UserRep   1        % Activer la sortie des résultats sur l'écran
%% Fin
```

Figure 53 - Exemple d'un fichier de configuration de NeuSim

Le paramètre du fichier de configuration qui spécifie les tâches à réaliser est un élément très important de notre simulateur. C'est à travers de ce paramètre qu'on va identifier la séquence d'opérations à réaliser. Les tâches réalisées par le simulateur peuvent être les suivantes :

- Charger un réseau ('P') : le simulateur va réaliser la lecture des informations qui décrivent un réseau pré-construit, pour ensuite l'utiliser afin de le raffiner par apprentissage, ou de simplement réaliser une consultation (mode de rappel). Cette option va charger les deux fichiers qui décrivent le réseau, le fichier <net>.wts et le fichier <net>.top;

- Créer un réseau ('R') : le simulateur va créer un réseau d'après la description de la topologie indiqué dans le fichier <net>.top. Les poids des connexions vont ensuite être initialisés avec des valeurs aléatoires;

- Réaliser l'apprentissage ('L') : le simulateur va lancer l'algorithme d'apprentissage du réseau (selon la méthode choisie) en utilisant le fichier de données <data>.learn. L'algorithme d'apprentissage va s'arrêter dans un des cas suivants : après avoir dépassé un maximum d'époques d'apprentissage, après avoir obtenu 100% de bonnes réponses sur le fichier d'apprentissage, après

avoir obtenu 100% de bonnes réponses sur le fichier de test de généralisation, ou après avoir réduit l'erreur de sortie jusqu'à une valeur inférieure à la valeur définie dans le fichier de configuration. La sélection de la méthode d'arrêt à employer est aussi indiquée dans le fichier de configuration;

- Réaliser des tests de généralisation ('G') : le simulateur fait un test de généralisation (fichier <data>.test) après chaque époque d'apprentissage (fichier <data>.learn). Chaque époque est donc constituée par : une présentation de toute la base d'apprentissage, l'adaptation des poids des connexions, et une présentation des données de test de généralisation;

- Réaliser un test final ('T') : le simulateur réalise un test sur les données d'un fichier spécifié comme le fichier de test. Cela se produit soit après la phase d'apprentissage (si l'option 'L' est indiqué), ou directement après le chargement d'un réseau pré-construit;

- Enregistrer les sorties du réseau ('O') : le simulateur va créer un fichier <data>.out contenant les réponses données par le réseau lors de son activation. Cette option doit être utilisée avec l'option 'T';

- Enregistrer séparément les bonnes et mauvaises réponses ('E') : le simulateur va créer deux fichiers de sortie, un fichier <data>.ok contenant les données qui ont été classées correctement, et un fichier <data>.err contenant les données qui n'ont pas été classées correctement. Les données utilisées dans ce test sont les données de test de généralisation. Cette option nous offre la possibilité d'analyser les 'cas problématiques' et de mieux valider le problème en question;

- Compter les réponses fausses ('F') : le simulateur présente les totaux des bonnes et des mauvaises réponses du réseau. Ces résultats sont basés uniquement sur des valeurs de sortie du type binaire (les valeurs continues sont discrétisées). Il va classer les réponses dans quatre groupes : sortie désirée vraie avec une sortie du réseau fausse, sortie désirée fausse avec une sortie du réseau vraie, sortie du réseau correcte, et sortie du réseau dans l'intervalle défini comme 'sans réponse'. Cette option nous permet de mieux interpréter les erreurs de réponse du réseau qui sont divisées en trois groupes (faux-0, faux-1, indéterminé);

- Enregistrer les poids du réseau ('W') : une fois terminé l'exécution des autres tâches, le simulateur va enregistrer les poids des connexions du réseau dans un fichier (<net>.wts). Cette option nous permet de reprendre l'apprentissage ou l'activation du réseau plus tard;

- Enregistrer la topologie du réseau ('S') : une fois terminée l'exécution des autres tâches, le simulateur va enregistrer la topologie du réseau dans un fichier (<net>.top);
- Enregistrer dans un fichier graphique la topologie du réseau ('V') : une fois terminé l'apprentissage, le simulateur va créer un fichier compatible avec Gnuplot contenant une description graphique de la topologie finale du réseau. Cette option nous permet de visualiser la topologie du réseau utilisée par le simulateur;
- Activer le menu ('I') : le simulateur, après avoir exécuté toute autre tâche définie dans le fichier de configuration, passe dans un mode d'interaction par menus. L'utilisateur peut ainsi exécuter toute une série d'opérations réalisées de façon automatique et ensuite passer dans un mode interactif afin de réaliser des tâches complémentaires.

Une des options présentes dans la liste de tâches, l'option 'G', nous permet de trouver le point optimal dans la courbe d'évolution de l'erreur de généralisation (voir Figure 10). Cette option est très intéressante puisqu'elle nous permet de trouver la meilleure configuration du réseau obtenue pendant tout le processus d'apprentissage. La simulation est faite de la façon suivante :

1. Après chaque époque d'apprentissage, le réseau est activé avec la base d'exemples de test afin de mesurer l'erreur de généralisation;
2. Le simulateur doit poursuivre l'apprentissage même en dépassant le point optimal, car on n'a pas les moyens de l'estimer par avance;
3. Chaque mesure de l'erreur de généralisation est comparée avec l'erreur de l'époque précédente. Si l'erreur actuelle est inférieure aux erreurs précédentes, le simulateur va garder cette valeur et l'indication de l'époque dans laquelle elle s'est produite;
4. A la fin de la simulation on aura l'indication de l'époque ('BestEpoch') où s'est produite la plus petite erreur de généralisation ('BestScore') de toute l'évolution du processus d'apprentissage;
5. Etant donné que le simulateur permet de configurer la valeur qui contrôle la génération des nombres aléatoires ('RndSeed'), il suffit de la régler exactement avec la même valeur utilisée dans une simulation précédente pour qu'on puisse avoir exactement la même évolution du réseau

pendant le processus d'apprentissage. La valeur de 'RndSeed' utilisée dans la dernière simulation peut être retrouvée dans le fichier de rapport (*.rep);

6. Par conséquence, si l'on veut arrêter le processus d'apprentissage à l'époque correspondant à la meilleure généralisation, il suffit d'utiliser la même valeur de 'RndSeed' utilisée lors d'une simulation précédente puis d'arrêter l'apprentissage à l'époque qui a permis le meilleur score lors de cette simulation précédente. (On donne à 'MaxEpochs' la valeur 'BestEpoch' antérieure).

Ce type de procédure nous permet de mieux exploiter les capacités d'apprentissage des réseaux afin d'obtenir le réseau le plus intéressant possible : celui qui donne la meilleure généralisation par rapport aux données disponibles.

Pour conclure la discussion sur les options de configuration, il est intéressant de souligner que le simulateur permet d'évaluer le pourcentage de bonnes réponses selon un critère prédéterminé par l'utilisateur. L'utilisateur doit indiquer l'erreur maximale ('MaxErr') qu'il acceptera par rapport aux valeurs de sortie du réseau, car même si ces unités donnent des valeurs continues comme résultat, la plupart du temps on cherche à interpréter les résultats obtenus dans les sorties comme des valeurs discrètes (le réseau est utilisé dans des tâches de classification). Ainsi, une sortie avec une valeur égale à X est reconnue comme 'correcte', si X est compris dans l'intervalle donné par la valeur de sortie désirée plus/moins une erreur (intervalle [Sd-MaxErr, Sd+MaxErr]). Par exemple, si l'on fixe MaxErr=0.3 et étant donné la sortie désirée égale à 1.0, on va considérer les réponses du réseau comme incorrectes pour toute valeur de sortie inférieure à 0.7. Donc, la valeur spécifiée pour le paramètre MaxErr joue un rôle très important par rapport à l'évaluation des erreurs commises par le réseau, ainsi que pour la définition de la précision des réponses exigée par l'utilisateur.

4.3.5 Discussion sur le Module Connexionniste

Nous avons présenté ici une description des principales caractéristiques et du mode de fonctionnement du module connexionniste ***NeuSim*** qui constitue une partie très importante de

notre système hybride. Ce module présente des caractéristiques très intéressantes, telles que : il est flexible ; il est performant (méthode de descente de la courbe d'erreur de deuxième ordre) ; il est incrémental (adaptation des poids et aussi de la topologie du réseau) ; il permet d'utiliser des connaissances théoriques compilées sous la forme d'un réseau ; il ne mélange pas les connaissances de départ avec les nouvelles connaissances ; il ne modifie pas la signification des unités après l'apprentissage ; il permet d'ajouter des nouvelles unités afin de corriger les problèmes existants dans les connaissances insérées dans le réseau. De plus, notre simulateur offre des possibilités très intéressantes de configuration et d'accompagnement des simulations, ainsi que des fonctions d'aide à la validation des résultats fournis par le simulateur.

Le grand nombre de paramètres réglables disponibles ne pose pas de difficultés supplémentaires aux utilisateurs, au contraire de ce qu'on pourrait imaginer. Pour simplifier encore plus la tâche de création d'un fichier de configuration, le simulateur NeuSim possède des fichiers modèles, lesquels peuvent être appliqués à d'autres problèmes sans avoir besoin de régler tous les paramètres. Une nouvelle application ne nécessite pas un expert pour réaliser la configuration du réseau, tout pouvant être facilement réglé à partir des fichiers modèles. D'un autre côté, tous ces paramètres vont permettre l'utilisation de ce simulateur dans un grand nombre de situations, où l'utilisateur peut les adapter selon ses besoins spécifiques.

NeuSim s'est avéré un outil facilement adaptable à des applications très spécifiques. Tel est le cas d'une interface Web (accès via l'Internet) de consultation à un système d'aide au diagnostic médical que nous avons développée (voir Section 5.2) ; et aussi d'un système réactif de contrôle d'un petit robot autonome - Khepera (voir Section 5.4). Finalement, ce simulateur sert aussi à réaliser des tâches de validation de connaissances et il s'intègre parfaitement à notre système INSS, d'une façon telle que nous allons pouvoir, dans la suite, 'ouvrir la boîte noire' que constitue le réseau et en extraire des règles propositionnelles.

4.4 EXTRACTION DE CONNAISSANCES

Le module Extract permet d'obtenir des règles symboliques à partir d'un réseau créé par INSS. Nos méthodes d'extraction de règles à partir des réseaux sont basées sur les algorithmes employés avec les réseaux KBANN (SUBSET et NofM - voir Section 3.3.2.3) auxquels nous avons apporté des améliorations. Ces algorithmes sont des méthodes du type décompositionnel. Ils ont été implémentés par Loïc Decloedt dans le cadre de son stage de D.E.A. [DEC 95] réalisé sous la supervision de F. Osório. Nous avons cherché a développer un outil d'extraction de règles, en nous donnant comme buts principaux :

- Obtenir une réduction de la complexité du processus d'extraction. L'extraction de règles par des méthodes comme SUBSET ou NofM peut demander un très grand effort computationnel selon le nombre d'unités et de connexions de chaque unité. Des expériences pratiques montrent que l'extraction de toutes les connaissances d'un réseau complexe peut être impraticable, et qu'il faut donc essayer de simplifier ce processus. Nous avons choisi de réaliser une simplification du réseau avant de réaliser l'extraction;

- Augmenter l'intelligibilité des règles. Un très grand nombre de règles peut être difficile à interpréter. Il faut trouver des moyens de représenter les connaissances d'une façon simple et synthétique;

- Réaliser une extraction sélective de règles. Les règles représentent une généralisation des connaissances décrites dans la base de cas. Il faut donc trouver tout d'abord les règles les plus générales possibles;

- Offrir la possibilité de valider les connaissances acquises et déjà représentées sous la forme de règles symboliques. Il faut pouvoir extraire des connaissances, mais pour qu'elles soient utiles, il faut les valider. Rappelons que nous partons du principe que ni les connaissances théoriques, ni les connaissances empiriques ne sont complètement correctes.

Le module Extract que nous avons conçu est un outil capable d'obtenir des règles symboliques (fichier *.symb) à partir d'un réseau préalablement entraîné (fichiers *.wts et *.top). Pour ce faire, nous réalisons tout d'abord une simplification du réseau composée d'une étape de sélection d'unités et d'une étape de sélection de connexions. Une fois simplifié le réseau, nous allons appliquer l'algorithme d'extraction SUBSET afin d'obtenir les règles symboliques. Nous

avons implémenté aussi la méthode NofM, mais cette méthode n'a pas été utilisée dans le cadre des expériences pratiques développées dans cette thèse.

La méthode d'extraction de règles que nous avons mise au point et qui utilise une simplification préalable du réseau, a été nommée ***RULE_OUT*** [DEC 96].

4.4.1 Sélection des Unités pour l'Extraction de Règles

Nous avons mis à la disposition de l'utilisateur trois méthodes de sélection des unités pour l'extraction de règles. Ces méthodes permettent de réduire significativement la quantité globale d'informations (unités) à analyser. Nous arrivons ainsi à simplifier notablement le processus d'extraction de règles. Les trois méthodes sont les suivantes :

1. *Sélection des unités qui représentent les nouvelles connaissances acquises.* Compte tenu de ce que INSS offre la possibilité d'intégrer des connaissances théoriques dans un réseau par compilation de règles, et que le processus d'apprentissage permet de les préserver inaltérées, nous n'avons pas besoin d'extraire à nouveau les connaissances théoriques initiales. Nous allons réaliser l'extraction des connaissances seulement sur les unités qui ont été modifiées (les unités ajoutés par l'algorithme CasCor) et qui représentent les nouvelles connaissances acquises. Ainsi, nous allons obtenir un ensemble de règles beaucoup plus compact contenant seulement les règles qui représentent les changements du réseau par rapport aux connaissances de départ. La sélection des unités modifiées par le processus d'apprentissage est très simple puisque nous les étiquetons lors de leur création. Il suffit alors de ne pas extraire des règles à partir des unités 'gelées' du réseau, ce qui permet de réduire significativement le temps de calcul. De plus, cette sélection d'unités nous permet d'implémenter une méthodologie d'extraction incrémental de connaissances. Ainsi, notre méthode d'extraction de règles s'intègre parfaitement dans le concept que nous proposons d'un système hybride d'apprentissage constructif.

2. *Sélection des unités d'entrées les plus significatives*. Nous proposons à l'utilisateur la possibilité de réaliser un prétraitement du réseau afin de réduire encore plus la quantité d'informations à traiter lors de l'extraction de règles. Nous avons implémenté la méthode d'élagage d'unités d'entrée du réseau basée sur l'analyse de sensibilité proposée par J. Moody [MOO 94] (*Sensitivity-based prunning method for input units - SBP algorithm*). Cette méthode est basée sur le calcul d'une 'mesure de sensibilité' S_i, qui va évaluer le changement infligé sur l'erreur d'apprentissage étant donné l'élagage d'une entrée X_i du réseau. Plus précisément, la mesure S_i (voir Eq. 29) indique l'effet du remplacement de l'entrée X_i par sa valeur moyenne (calculée sur tous les exemples de la base d'apprentissage) sur l'erreur quadratique (SE).

$$Si = \frac{1}{N}\sum_{j=1}^{N}\frac{\partial SE(j)}{\partial x_i(j)} \quad .dx_i(j) \qquad \text{(Eq. 29)}$$

$$d\,x_i(j) = \left(x_i(j) - \bar{x}_i\right)$$

N : nombre d'exemples dans la base d'apprentissage

$\bar{x}_i$: valeur moyenne de l'entrée X_i

$x_i(j)$: valeur de l'entrée 'i' pour l'exemple 'j' de la base d'apprentissage

Le module Extract fait le calcul de l'influence des entrées sur l'erreur globale, offre la possibilité de les afficher à l'écran, et propose à l'utilisateur d'établir un seuil d'influence. Le système coupe alors toutes les connexions vers ces unités d'entrée (élimination des entrées) dont l'influence est inférieure au seuil considéré.

3. *Sélection des unités du réseau qui ont le plus d'influence sur le comportement du réseau.* Nous avons développé une méthode empirique de simplification du réseau, qui consiste à détecter les unités qui ont une valeur de sortie d'activation pratiquement constante pour tous les exemples contenus dans une base de connaissances indiquée par l'utilisateur. Ces unités peuvent donc être approximées par des unités à sortie constante, c'est-à-dire, avec une sortie toujours égale à 0.0 ou 1.0. Ensuite, nous pouvons les éliminer en changeant le seuil des unités auxquelles elles sont connectées [DEC 95]. De cette façon, nous avons la possibilité de simplifier le réseau sans affecter significativement sont comportement.

C'est l'utilisateur qui va choisir le seuil d'influence à partir duquel les unités vont être éliminées. Le seuil va être indiqué par la valeur de la fréquence d'activation des unités à considérer (voir Figure 54).

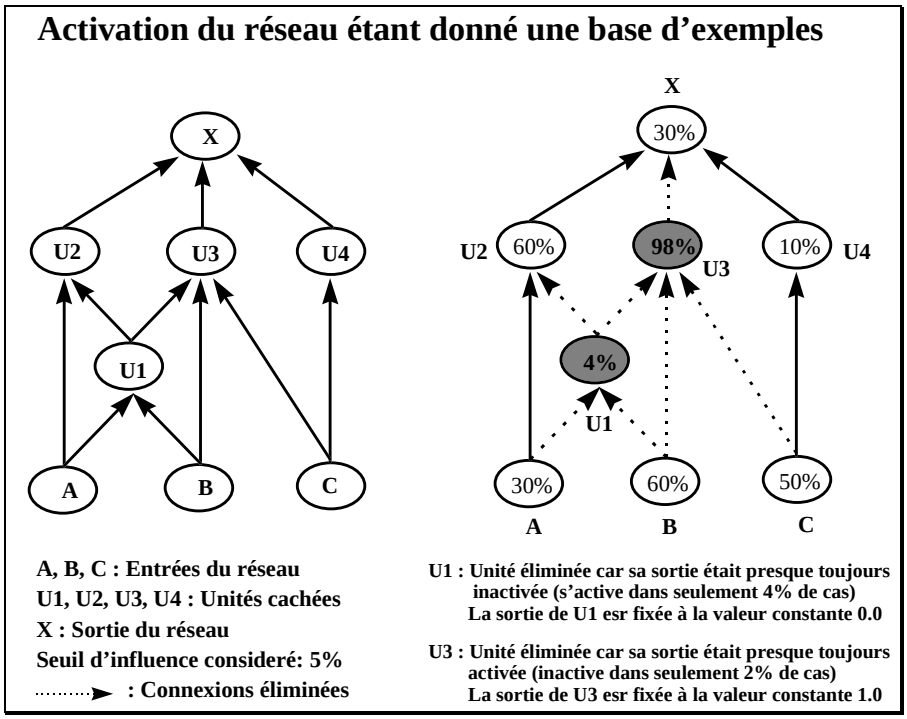


Figure 54 - Sélection des unités basée sur la pourcentage d'activation

La fréquence d'activation d'une unité indique le pourcentage des cas de la base de d'exemples pour lesquels la sortie de cette unité est égale à 1.0 (activée). Si la base d'exemples couvre bien l'espace d'entrées et si les exemples sont bien repartis dans les classes (dans le cas de la Figure 54, les classes sont X=0.0 et X=1.0), alors la fréquence d'activation peut être un bon paramètre pour estimer l'influence d'une unité sur les réponses données par le réseau. Le module Extract laisse à l'utilisateur le choix de l'application de cette méthode de simplification du réseau ainsi que de la valeur du seuil d'influence à utiliser.

Le module Extract offre une interface à l'utilisateur basée sur des menus, qui lui permet de définir jusqu'à quel point il veut simplifier le réseau. L'interface lui offre le choix d'application des différentes méthodes que nous présentons ici, et en plus, il a la possibilité de mesurer à tout

moment l'influence d'une simplification sur l'erreur des réponses du réseau par rapport à une base d'exemples donnée. Si l'utilisateur ne veut pas réaliser la simplification du réseau avant l'extraction, il peut passer directement à l'application de la méthode d'extraction. Dans ce dernier cas, le processus d'extraction va être beaucoup plus lent, demander plus de ressources (mémoire) de la machine, et doit conduire à l'obtention d'un ensemble de règles beaucoup plus grand.

4.4.2 Sélection des Connexions pour l'Extraction de Règles

Nous avons aussi développé des méthodes pour mesurer l'influence des poids par rapport aux sorties du réseau, de façon à pouvoir éliminer celles qui ont le moins d'influence. La possibilité de réaliser cette simplification du réseau est très importante, car la complexité du processus d'extraction est fortement liée au nombre de connexions d'entrée de l'unité analysée. Le nombre de règles obtenues par extraction peut lui aussi être réduit de façon significative. Nous avons mis à la disposition de l'utilisateur deux méthodes de sélection des connexions pour élaguer le réseau avant l'extraction de règles. Les deux méthodes sont les suivantes :

1. *Sélection des connexions les plus faibles.* Nous avons développé une méthode empirique de simplification des connexions, basée sur ce que nous avons appelé l'étude du pourcentage des poids. Cette méthode est basée sur la supposition que, parmi les connexions qui arrivent à une même unité, celles qui ont les poids les plus faibles (valeurs très petites) par rapport aux autres, sont les connexions qui ont le moins d'influence sur la sortie de l'unité considérée. Cette hypothèse s'est avérée valable dans nos expériences pratiques portant sur les réseaux créés avec l'algorithme CasCor. L'algorithme CasCor semble bien attribuer des poids très faibles aux connexions moins importantes.

La méthode que nous avons développée consiste à mesurer l'importance relative de chaque poids par rapport aux autres poids de même signe liés à la même unité [DEC 96]. Pour faire cela, nous calculons le pourcentage de chaque poids par rapport à la somme globale des autres poids (positifs et négatifs). Ainsi, l'utilisateur peut spécifier un seuil d'influence (pourcentage) utilisé pour la sélection des poids à éliminer. Il pourra aussi consulter l'effet de cet élagage sur l'erreur globale d'activation du réseau.

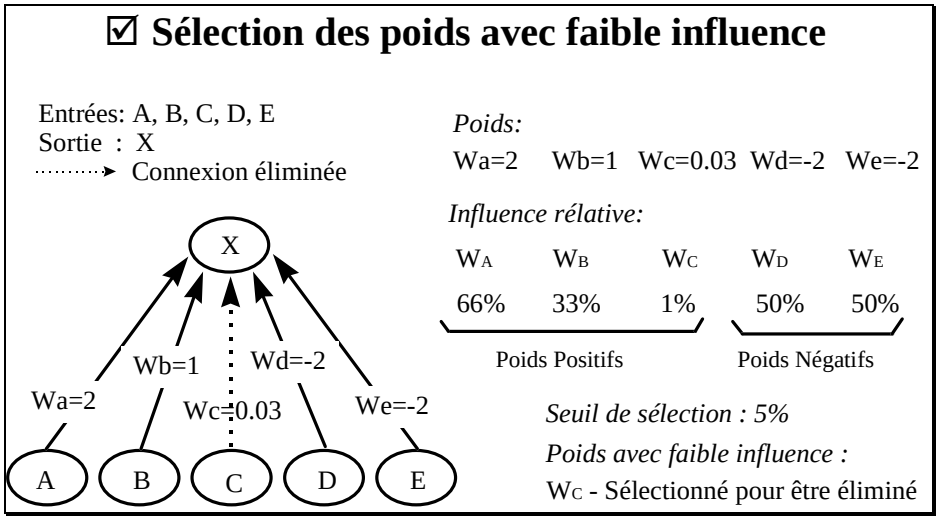


Figure 55 - Sélection des connexions les plus faibles

2. *Sélection des connexions par rapport à leur influence sur l'erreur globale.* La supposition faite que l'influence des connexions est proportionnelle à la valeur relative des poids ne prend pas en compte un autre facteur important : si un poids très petit a peu d'influence sur la sortie de l'unité, par contre il a été aussi observé que parfois des connexions avec des poids assez grands n'avaient pas un rôle significatif par rapport aux réponses globales du réseau. Ainsi, nous avons implémenté une autre méthode inspirée par l'algorithme OBD (*Optimal Brain Damage*) [LEC 90], qui lui va prendre en compte cet aspect. Cet algorithme a été développé afin de mesurer l'influence de chaque poids sur l'erreur global, à travers le calcul de la dérivée seconde de l'erreur globale. Cette mesure de l'influence d'un poids Wk sur l'erreur quadratique moyenne ASE (*Averege Squared Error*) d'activation d'un réseau, appelée *'Saliency'*, est définie par l'équation suivante :

$$S(w_k) = \frac{1}{2} \frac{\partial^2 ASE}{\partial w_k \partial w_k} . (w_k)^2 \qquad \text{(Eq. 30)}$$

Les poids avec une faible 'Saliency' peuvent donc être éliminés du réseau, et cela va permettre à l'utilisateur d'optimiser encore plus le processus d'extraction de règles.

4.4.3 Application de l'algorithme SUBSET dans INSS

Une fois terminée la sélection des unités et des connexions qui vont être considérées pour l'extraction de règles, nous passons à l'application de l'algorithme SUBSET. L'algorithme SUBSET réalise l'extraction de règles d'ordre 0 à partir du réseau simplifié, selon les procédures que nous avons décrites dans la section 3.3.2.3. Il n'y a pas de différence significative entre l'implémentation de SUBSET dans INSS et la version d'origine utilisée dans KBANN. La différence entre l'approche KBANN et notre approche se trouve dans le post-traitement qu'on peut appliquer aux règles extraites. Ainsi, le module Extract met à disposition de l'utilisateur des procédures de simplification du réseau avant extraction, d'extraction de règles par l'algorithme SUBSET, et de simplification de la base de règles extraite à partir du réseau.

Notre méthode de post-traitement des règles extraites consiste à calculer la représentativité de chaque règle par rapport à une base d'exemples fournie par l'utilisateur. Nous obtenons une mesure qui indique le pourcentage des exemples qui activent une règle spécifique (le nombre de fois que ses antécédents sont satisfaits). Cette mesure est alors présentée à l'utilisateur afin qu'il puisse se décider sur le niveau de simplification à appliquer sur cette base de règles obtenue par l'algorithme SUBSET.

La justification d'une telle approche est qu'il peut arriver que certaines règles ne soient pas utiles dans un contexte particulier, car elles ne sont pas applicables dans la pratique. Par exemple, prenons le cas d'un réseau possédant deux entrées booléennes E1 et E2 dont les valeurs sont *toujours* complémentaires. Il peut arriver que le système d'extraction obtienne en résultat une règle dont la condition soit une conjonction de ces entrées : *Si E1 et E2 alors X*. Cette règle peut être valide, mais elle ne peut s'appliquer dans le cas présent, car sa condition ne sera jamais valide. L'inconvénient est que, si l'on obtient un grand nombre de règles de ce type, on risque de se retrouver avec beaucoup de règles inutiles qui risquent alors de diminuer la compréhensibilité du résultat. Pour pallier à ce problème, l'utilisateur peut faire appel à la méthode de post-traitement que nous venons de décrire. En conséquence, nous sommes capables alors d'éliminer les règles inutiles qui ne sont jamais, ou presque, utilisées dans la pratique, et d'augmenter ainsi la compréhensibilité du résultat.

Cette fonctionnalité est plus importante qu'il y paraît au premier abord, car le nombre de règles inutiles de ce type peut être très grand. En effet, si l'on modélise un problème dont un des paramètres est une variable discrète (type 'nominal') pouvant prendre un nombre de valeurs finies, alors on va créer une unité d'entrée pour chaque valeur possible de ce paramètre (codage adopté par INSS pour les variables discrètes). En pratique, les valeurs de ces unités s'excluent mutuellement, c'est-à-dire qu'une et une seule de ces unités peut être activée à cet instant donné. Or, ce type d'informations n'apparaît pas au niveau du réseau, et on traite toutes les entrées de manière équivalente. Donc, le système d'extraction, lors de la recherche des règles possibles, va étudier les combinaisons de ces entrées qui s'excluent mutuellement, et peut donc en déduire des règles tout à fait correctes dans l'absolu, mais qui ne seront jamais utilisées. Par exemple :

Variable - Couleur : nominal # 3 : rouge, verte, bleue;

Règles extraites - informations redoublés :

x := couleur(rouge), not(couleur(verte)), not(couleur(bleue));

x := couleur(rouge), not(couleur(verte));

x := couleur(rouge), not(couleur(bleue));

x := not(couleur(verte)), not(couleur(bleue));

Règles extraites - informations invalides :

x := couleur(rouge), couleur(verte), couleur(bleue);

x := couleur(rouge), couleur(verte);

x := couleur(rouge), couleur(bleue);

x := couleur(verte), couleur(bleue);

Ce nombre de règles parasites peut devenir très élevé si un paramètre de départ discret peut prendre beaucoup de valeurs. Dans certains cas, le module d'extraction finit par trouver des méta-règles, qui n'expriment pas des connaissances sur le problème en soi, mais des connaissances sur les connaissances (liées à la forme de codage et à la validité des variables).

4.4.4 Discussion sur le Module d'Extraction de Règles

Nous avons présenté ici la méthode d'extraction de règles ***RULE_OUT***, capable de réaliser une simplification préalable des réseaux afin de réduire la complexité du processus d'extraction de règles. La méthode *RULE_OUT* est une amélioration de la méthode SUBSET d'extraction de règles, cependant cette méthode peut aussi être utilisée avec d'autres algorithmes décompositionnels d'extraction de règles, tel est le cas de l'algorithme NofM. Notre méthode s'intègre parfaitement au système INSS, car elle est une méthode d'extraction incrémentale de connaissances. L'utilisation de *RULE_OUT* nous permet d'obtenir juste les nouvelles connaissances acquises, tout en préservant inaltérées les connaissances de départ.

Les méthodes d'extraction de règles avec l'élagage des réseaux que nous avons décrites dans les sections précédentes, ont été testées sur différents problèmes (e.g. les problèmes du Moine - "Monk's problem", voir Section 5.1) et comparées avec l'extraction de règles sans simplification des unités et des connexions. Les résultats obtenus montrent bien une réduction très sensible de la complexité du processus d'extraction après la simplification du réseau, ainsi qu'une amélioration de la qualité des règles obtenues (augmentation de la compréhensibilité par la réduction du nombre de règles obtenues) [DEC 95]. Nous allons présenter les résultats de l'utilisation du module d'extraction de règles dans le chapitre qui traite des applications du système INSS (Chapitre 5), dont les problèmes du Moine [THR 91] composent une des applications qui nous a permis de bien mettre en valeur les principales propriétés de notre implémentation.

Les travaux que nous avons développés sur ce sujet, nous ont amené à poser un certain nombre de questions à propos de la représentation de connaissances. En réalité, un des nos principaux buts a depuis toujours été le développement d'outils pour la manipulation et la conversion des différents types de représentation de connaissances : la transformation de règles en réseaux, l'acquisition et la représentation de connaissances empiriques par des réseaux, l'extraction de connaissances des réseaux (on devrait plutôt dire 'explicitation de connaissances'). Ces différents processus servent à manipuler les connaissances afin de les transformer d'une forme de représentation à l'autre (e.g.: symbolique - règles propositionnelles, symbolique - base d'exemples, numérique - connexions et poids). Les questions qui se posent sont donc : pourquoi

doit-on choisir une méthode de représentation plutôt qu'une autre ? Existe-t-il une méthode de représentation plus puissante que toutes les autres ? Quelles sont les limitations de la capacité de représentation d'une méthode spécifique ? Nous n'avons pas l'intention de répondre précisément à ce type de questions dans cette thèse, car nous pensons que çe serait trop ambitieux de notre part d'essayer de répondre à des questions qui jusqu'ici sont restées sans réponse définitive. Nous pensons que chaque formalisme a sa particularité par rapport à sa capacité de représenter (et aussi de généraliser) des connaissances. Par conséquence, ce qu'on doit essayer de faire c'est de mieux connaître chaque type de méthode de représentation afin de pouvoir l'exploiter de la meilleure façon.

Dans cet esprit, nous avons proposé un stage de D.E.A. afin de mieux étudier 'la représentation et l'apprentissage de règles de haut niveau'. Cette étude a été développée par Gerardo Reyes-Salgado dans le cadre de son stage de D.E.A. [REY 97] réalisé sous la supervision de F. Osório au Laboratoire LEIBNIZ. Nous avons cherché à étudier des règles d'un niveau supérieur aux règles d'ordre 0 ou 0^+ traditionnellement employées avec les systèmes hybrides neuro-symboliques. Nous avons approfondi nos études sur les règles contextuelles et sur les règles avec des relations d'ordre supérieur entre leurs variables, c'est-à-dire, des règles où la relation entre deux variables est plus forte qu'une simple relation booléenne du type 'Et/Ou' (e.g. plus A est grand par rapport à B, plus il va influencer la prise d'une décision X). Nous allons décrire plus en détail ces recherches dans la section qui traite des résultats des applications d'INSS dans le problème de la balance (voir Section 5.3).

Les recherches que nous avons menées sur les règles de haut niveau ont été fructueuses [REY 97a] Elles nous ont en particulier permis de mieux nous situer par rapport aux problèmes de représentation de connaissances des systèmes hybrides neuro-symboliques. Rappelons que le système KBANN permet d'insérer et d'extraire seulement des règles d'ordre 0 (les entrées continues et les règles d'ordre 0^+ sont traitées en dehors du réseau par des techniques de pré-traitement) et le système SYNHESYS permet d'insérer et d'extraire seulement des règles simples d'ordre 0^+ (les variables sont considérées indépendamment, comme aussi dans les arbres de décision). Ces deux systèmes ne permettent pas de traiter des relations entre deux variables continues, comme par exemple : 'plus_grand_que' ou 'plus_petit_que'. Dans INSS, nous avons

proposé des méthodes d'insertion de règles d'ordre 0^+ et des règles de haut niveau (voir Section 4.2.1) et nous avons aussi réalisé des études afin d'étendre le pouvoir de représentation de connaissances en utilisant des réseaux d'ordre supérieur, comme par exemple, les réseaux *Sigma-Pi* [REY 97, REY 97a].

Le point faible de notre système qui reste à améliorer est relatif aux type de règles que nous pouvons obtenir à partir de la partie connexionniste. Actuellement, nous pouvons extraire seulement des règles d'ordre 0, puisque nous nous sommes basés sur les méthodes d'extraction de règles utilisées dans KBANN (SUBSET et NofM). Cela représente une limitation non négligeable de notre système, car les règles extraites n'ont pas le même pouvoir de représentation de connaissances qu'au niveau de l'insertion de connaissances dans les réseaux. C'est pourquoi nous avons orienté nos recherches dans la direction de l'extraction de règles de haut niveau. Nous sommes en train de chercher des solutions à ce problème (G. Reyes a débuté une thèse cette année qui devra aborder ce sujet). Etant donné que les méthodes d'insertion de règles d'ordre 0^+ sont déjà opérationnelles, nous pensons que cela doit nous aider dans la recherche des méthodes capables d'extraire des règles plus complexes.

4.5 VERIFICATION ET VALIDATION DE CONNAISSANCES

Le module Valid permet de vérifier et de valider des règles symboliques et des données d'apprentissage. Nos méthodes de vérification et de validation ont été développées à partir de la nécessité de mettre en relation les règles et les exemples afin de trouver des inconsistances entre ces deux ensembles de connaissances. Ces algorithmes ont été réalisés et testés par Pierre-Yves Blond dans le cadre de son stage de D.E.A. [BLO 96] en utilisant le langage/moteur d'inférence CLIPS [GIA 93]. Les travaux développés par P.-Y. Blond, dans son stage de D.E.A., ont été réalisés sous la supervision de F. Osório.

Nous avons commencé nos recherches sur ce sujet à la suite des résultats obtenus dans l'extraction de règles à partir des réseaux. Comme nous l'avons déjà présenté dans la section précédente, les règles extraites peuvent contenir des information redondantes ou incohérentes,

d'où l'idée d'appliquer des méthodes de vérification et validation des connaissances. De plus, nous avons choisi comme principe de base de notre système hybride, l'idée qu'il doit être conçu pour travailler avec des connaissances qui sont usuellement incomplètes et imparfaites. Or, si les connaissances ne sont ni complètes, ni parfaites, cela veut dire que ni les bases de règles, ni les bases d'exemples, ne doivent être considérées comme des connaissances absolues. Par conséquence, nous allons contester le mode de fonctionnement adopté par les réseaux KBANN, où les connaissances théoriques sont modifiées par apprentissage, mais où les connaissances empiriques (base d'exemples) ne sont jamais mises en question du point de vue de leur qualité. Nous avons choisi d'utiliser une approche qui est plutôt similaire à celle employée par le système SYNHESYS (acquisition constructive et interactive des connaissances). Dans ce cas de figure, un module comme Valid devient un outil très important qui permet l'implémentation de notre système hybride d'acquisition constructive de connaissances.

Avant de passer à la description du fonctionnement du module Valid, il faut définir de façon plus précise ce que l'on entend par vérification et validation de connaissances [FU 94]. La *vérification de connaissances* consiste à tenter de débusquer des erreurs structurales dans une base de règles (ou d'exemples), telles que : redondance, conflit, circularité, impasses, buts que l'on ne peut pas atteindre, attributs inutiles, attributs incohérents. Nous avons concentré nos efforts sur la vérification du type statique (cohérence statique) [FU 94, AYE 90], où nous cherchons à étudier l'adéquation entre les éléments des bases de connaissances et un modèle conceptuel du domaine défini à l'avance. Un exemple de vérification que l'on peut faire est le test de validité des valeurs d'entrée par rapport au type de la variable représentée : âge = valeur entière entre 0 et 130. Le langage CLIPS nous offre la possibilité d'établir ce type de contrôle d'une façon automatique.

D'un autre côté, la *validation de connaissances* porte sur la qualité des connaissances, ce qui va affecter la performance du système en termes de correction, d'efficacité, de robustesse, de sensibilité et d'adaptabilité. Nous nous sommes concentrés sur les aspects concernant les incohérences entre les deux bases de connaissances : la base de connaissances théoriques et la base de connaissances empiriques. Pour valider les connaissances, nous allons consulter la base de règles par inférence symbolique et comparer les réponses obtenues par rapport aux exemples

contenus dans la base d'apprentissage. Dans le cas où les réponses sont contradictoires, nous faisons une analyse des données disponibles (activation par chaînage avant et arrière), afin de fournir à l'utilisateur le plus d'information possible qui puisse l'aider à identifier la cause de ce comportement anormal. Un exemple de validation que l'on peut faire est le cas où l'activation par chaînage avant avec un des exemples de la base d'apprentissage produit une réponse positive (sortie activée) et ce même exemple possède comme sortie désirée une valeur de réponse négative (sortie inactivée). Les statistiques sur le nombre de fois qu'un certain comportement anormal se produit, ainsi que l'indication précise des règles et des exemples liés aux cas problème, vont permettre à l'expert de décider de l'action à prendre (e.g.: donner priorité aux connaissances théoriques et ignorer l'exemple en l'effaçant de la base, corriger la base de règles selon les indications du système, obtenir plus d'exemples pratiques apparentés au cas problème, identifier une limitation imposée par rapport à la façon dont le problème a été posé, adapter le réseau par apprentissage des exemples et extraire des nouvelles règles pour remplacer les anciennes).

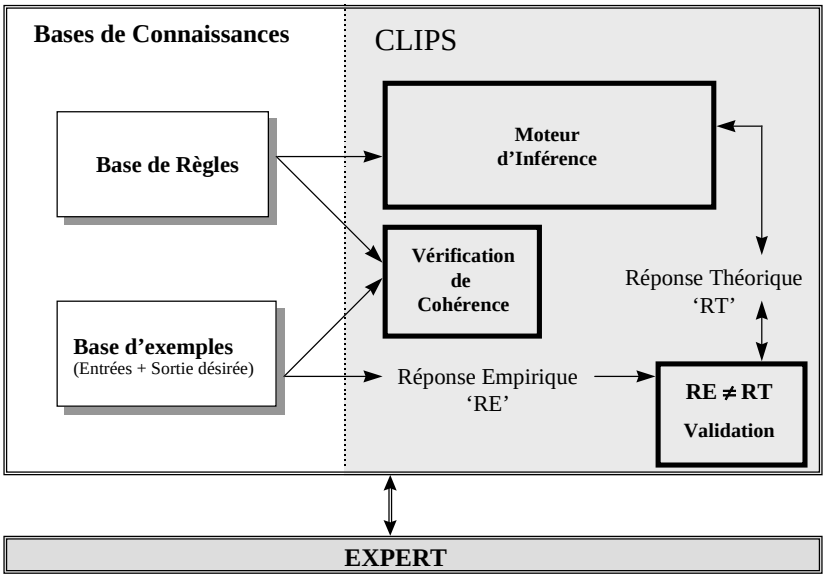


Figure 56 - Processus de vérification et de validation de connaissances

Le processus de vérification et validation de connaissances, tel qu'il a été étudié dans INSS (voir Figure 56), suit les étapes suivantes :

1. Vérification de la cohérence des règles et des exemples par une analyse du type statique. Dans cette phase, on va charger les règles et les exemples dans la mémoire de travail, en effectuant des vérifications sur leurs attributs. Le langage CLIPS permet de spécifier et de gérer des erreurs d'exception causées par l'utilisation de valeurs incorrectes dans les exemples. Nous pouvons aussi détecter facilement les règles qui ne sont jamais activées et les redondances.

2. Pour chaque exemple dans la base d'apprentissage (base de données constituée par les exemples avec leurs valeurs de sortie désirées), les règles sont activées par chaînage avant. Nous obtenons ainsi la 'réponse théorique', basée sur l'ensemble des règles disponibles. La valeur de la sortie désirée de l'exemple est appelée 'réponse empirique'. Nous avons quatre cas de figure possibles selon les valeurs de la 'réponse théorique' (sortie calculée - Sc) et de la 'réponse empirique' (sortie désirée - Sd)* :

   I. Sortie calculée active et correcte ('hit') : Sortie Sc=Sd=1.0, passer à l'exemple suivant;

   II. Sortie calculée inactive et correcte ('reject') : Sortie Sc=Sd=0.0, passer au suivant;

   III. Sortie calculée active et fausse ('false-alarm' - Faux_0): Sortie incorrecte, Sc≠Sd=0.0;

   IV. Sortie calculée inactive et fausse ('omission' - Faux_1): Sortie incorrecte, Sc≠Sd=1.0;

   * *Il faut retenir ici que les sorties dans INSS sont toujours binaires avec des valeurs approximées à 0.0 (inactivée) ou 1.0 (activée).*

3. Si la 'réponse théorique' n'est pas en accord avec la 'réponse empirique' (cas III et IV), le cas en question est signalé à l'expert et ensuite analysé plus en détails. Cette analyse consiste à faire une activation des règles par chaînage arrière et à établir à chaque pas en arrière l'importance des antécédents d'après l'état du conséquent. Les règles et ses antécédents sont ainsi classés en trois groupes : activation nécessaire ('want'), activation indéfinie ('nnns') et activation interdite ('forbid'). Le tableau ci-dessous indique la propagation en arrière de l'état des règles vers ses antécédents :

Etat d'activation de la Règle	Antécédents de la Règle	
	Opérateur du type 'OU'	Opérateur du type 'ET'
Want	NNNS	Want
NNNS	NNNS	NNNS
Forbid	Forbid	NNNS

Les règles et leurs antécédents qui ont des états d'activation incompatibles avec les états 'want' et 'forbid' établis par le chaînage arrière sont indiquées à l'expert/utilisateur.

4. Nous mettons à jour les statistiques de : nombre de 'hits', nombre de 'rejects', nombre de 'false-alarms' et nombre de 'omissions', ainsi que la mesure de l'utilisation ('utility') et du degré de correction ('correctness') relative des règles. La valeur 'utility' indique le nombre de fois qu'une règle à été déclenchée par rapport au nombre total d'exemples. La valeur 'correctness' indique le nombre de fois que la règle à été déclenchée et a conduit à une bonne réponse (cas I et II), par rapport au nombre total d'exemples.
5. Ensuite, le cas suivant de la base d'exemples est analysé (retour à l'étape '2').

Après avoir analysé tous les cas de la base d'exemples, nous présentons un rapport sur chaque exemple et sur chaque règle avec les totaux généraux cumulés dans l'étape '4' décrite ci-dessus (Pour obtenir une description plus détaillée des résultats affichés à la sortie du module Valid, voir le rapport [BLO 96]).

4.5.1 Discussion sur le Module de Vérification et Validation de Connaissances

Le module Valid reste le module le moins développé du système INSS. Cependant les fonctions de base actuellement disponibles se sont déjà montrées utiles en plusieurs occasions (voir le Chapitre 5). Le module Valid a été réalisé en utilisant le langage CLIPS. D'un point de vue pratique il n'est pas encore bien intègré à notre système (en particulier au module connexionniste). Nous avons décidé de mettre à disposition des utilisateurs certaines des fonctionnalités de ce module directement sur le module connexionniste (NeuSim), ce qui a simplifié la façon d'utiliser les fonctions basiques de validation de connaissances. Le module NeuSim nous offre lui aussi la possibilité de :

- séparer les exemples bien classés par rapport à une base de règles disponible, de ceux qui ont une réponse incompatible avec celle donnée par l'ensemble de règles (Pour ce faire, il suffit de coder une base de règles sur un réseau et de configurer le simulateur afin d'enregistrer les exemples incorrects dans un fichier à part);

- indiquer le nombre d'erreurs commises par un réseau étant donné une base d'exemples, tout en faisant la séparation des cas de 'faux_0' et de 'faux_1' (les réponses sont groupées dans 4 classes : correctes, incorrectes par 'faux_0', incorrectes par 'faux_1', incorrectes par indécision);

- indiquer la séquence de règles qui sont activées suite à la présentation d'un exemple particulier (règles compilées sous la forme d'un réseau), ce qui nous permettra de mieux analyser les résultats obtenus par l'activation du réseau.

D'un autre côté, le module Extract lui aussi offre des possibilités de simplification des règles extraites à partir d'un réseau, à travers une vérification des règles obtenues et l'élimination des règles inutiles. Ce module nous permet aussi d'identifier des attributs d'entrée qui n'ont pas d'influence sur les résultats obtenus à la sortie.

Toutes ces fonctions de vérification et de validation de connaissances nous ont permis d'obtenir des résultats intéressants dans deux cas d'application du système INSS. Dans une application d'aide au diagnostic médical des comas toxiques (voir section 5.2), nous avons trouvé des anomalies d'apprentissage, alors même que le réseau étudié avait un bon taux de réussite. Une analyse plus détaillée des résultats (nombre de 'faux_0' et 'faux_1') nous a permis d'identifier que le réseau donnait une réponse unique dans sa sortie, avec des valeurs toujours égales à 0 (absence d'une substance toxique X) ou toujours égales à 1 (présence d'une substance toxique X), selon le type d'exemples dominants dans la base d'apprentissage. Dans une autre application, le problème de la balance (voir Section 5.3), nous avons pu identifier des problèmes liés à l'incapacité du réseau de bien représenter de façon interne une règle de haut niveau [REY 96]. Une étude des 'cas problème' nous a permis de mieux comprendre pourquoi nous avions des problèmes pour apprendre tous les exemples. Nous allons aborder ces deux cas d'application du système INSS dans le chapitre suivant.

Nous pensons que le module de vérification et de validation de connaissances doit être encore mieux étudié, car c'est justement à partir des 'conflits cognitifs' (e.g.: contradictions et erreurs commis) que nous faisons évoluer nos connaissances, d'où la grande importance de cette étape dans un processus global d'acquisition constructive de connaissances. Le grand nombre de recherches en cours sur la vérification et la validation des bases de connaissances [CRA 97,

NAZ 97, FU 94, AYE 90] montre bien l'importance de ce domaine qui reste ouvert à la recherche.

Finalement, soulignons que les outils de vérification et de validation que nous avons mis à disposition des utilisateurs dans le système INSS ne représentent qu'une partie des méthodes que l'on aimerait avoir a sa disposition. Nous laissons à des études futures certains aspects tels que : la détection de boucles dans les règles, la possibilité d'associer un facteur de certitude aux règles et aux exemples, la détection de contradictions internes à la base d'exemples ou de règles, la validation de règles avec des antécédents non-discrets (validation des intervalles d'activation pour les valeurs continues).

4.6 CONSIDERATIONS FINALES SUR LE SYSTEME INSS ET L'ACQUISITION CONSTRUCTIVE DE CONNAISSANCES

Le système INSS a été conçu afin d'utiliser toutes les connaissances disponibles sur un problème: théoriques et empiriques. Or, ces connaissances sont généralement incomplètes et peuvent contenir des incorrections, et en plus, elles peuvent évoluer avec le temps. En prenant ces idées comme nos concepts de base, nous avons construit notre système hybride neuro-symbolique d'acquisition constructive de connaissances.

Le système INSS s'est inspiré d'autres systèmes, comme KBANN et SYNHESYS, et il a apporté plusieurs améliorations par rapport à eux. De plus, la façon d'intégrer les connaissances théoriques et empiriques et de les faire évoluer, telle que nous l'avons réalisé dans INSS, à été aussi inspiré par ce que nous avons appelé de la "*méthodologie de l'élève-chercheur*" [OSO 95]. La méthodologie de l'élève-chercheur est une caricature du comportement d'un individu qui commence comme un élève mais qui pourra aussi devenir un jour un chercheur. Le côté *élève* représente la phase de l'acquisition de connaissances où il reçoit des connaissances théoriques, c'est-à-dire, des connaissances a priori concernant le domaine traité. Ces connaissances lui sont fournies par un maître (expert). Par exemple, un élève dans un cours de physique va commencer par l'apprentissage des lois (règles) physiques qui représentent ce domaine de connaissances.

L'élève doit ensuite procéder à des expériences pratiques pour valider et améliorer ses connaissances, et peut être ainsi il va devenir un *chercheur*. Alors, le chercheur est symbolisé par une personne qui essaye d'utiliser ses connaissances théoriques initiales et ses expériences pratiques afin d'arriver à des nouvelles connaissances théoriques sur le domaine de recherche. Une fois que les nouvelles connaissances proposées par le chercheur deviennent 'acceptables' (validées), elles vont être ensuite transmises aux élèves ; et ainsi la boucle est bouclée. Notre "méthodologie de l'élève-chercheur" est donc un modèle assez générique de l'acquisition constructive de connaissances. Une approximation simpliste de ce modèle peut être retrouvé dans la façon de fonctionner de notre système INSS.

Nous sommes conscients que notre système ne représente qu'un modèle simplifié de cette méthodologie, puisque les connaissances sont exprimées par un langage très simple. Cependant, le système INSS présente déjà certaines propriétés très intéressantes, dues au modèle d'acquisition constructive de connaissances adopté, qui font de ce système un outil intéressant pour l'étude des processus cognitifs relatifs à l'apprentissage humain.

Comme nous l'avons vu tout au long de ce chapitre, le système INSS apporte toute une série d'améliorations à son prédécesseur KBANN. Les propriétés le plus intéressantes d'INSS sont donc :

- INSS a la possibilité d'insérer par compilation des règles d'ordre 0^+ (et des règles de haut niveau) dans un réseau;

- INSS offre la possibilité de travailler avec des données qualitatives aussi bien qu'avec des données quantitatives;

- INSS utilise un réseau constructif, ce que lui permet de faire évoluer les connaissances du réseau ainsi que son structure. L'algorithme d'apprentissage utilisé est reconnu par sa performance supérieure à l'algorithme classique de la Rétro-propagation;

- INSS utilise une méthode d'extraction de règles incrémentale. Le processus d'extraction de règles que nous avons réalisé est plus performant que celui de son prédécesseur KBANN, car il

permet d'extraire seulement les nouvelles connaissances acquises, tout en préservant les connaissances préalablement introduites dans le réseau;

- INSS permet de simplifier le réseau (couper connexions et éliminer unités) avant l'extraction de règles. Nous essayons d'extraire les règles les plus représentatives des connaissances codées dans le réseau;

- INSS offre la possibilité de valider les nouvelles connaissances acquises par rapport aux anciennes;

- INSS est un outil qui peut être facilement adaptable à des applications dans différentes domaines.

Toutefois, il reste encore des problèmes à résoudre et des recherches à réaliser afin de perfectionner encore plus cet outil. Deux points principaux à développer sont :

- le développement de nouvelles méthodes d'extraction de règles afin de pouvoir extraire des règles de plus haut niveau à partir des réseaux et de mieux exploiter les connaissances qui y sont représentées;

- le perfectionnement des outils de vérification et de validation de règles disponibles.

Il y a d'autres perspectives d'étude sur le système INSS, tels les deux sujets que l'on vient de décrire ci-dessus, mais aussi :

- la possibilité d'utiliser un moteur d'inférence plus puissant à la place du langage CLIPS, comme pour exemple l'ECLIPSE (une version améliorée de CLIPS qui inclut la possibilité de réaliser un chaînage arrière);

- l'étude des possibilités d'adaptation des intervalles définis pour les variables d'entrée continues. Nous avons réalisé des fonctions qui permettent de coder des règles d'ordre 0^+ sur un réseau, mais nous n'avons pas approfondi les possibilités d'adaptation de ces règles vers un niveau plus fin que celui de l'ajout de nouvelles unités au réseau;

- l'insertion par compilation de fonctions plus complexes dans les réseaux. Nous avons envisagé la possibilité d'ajouter des unités qui auraient des liens (interface) vers des fonctions en 'C'. L'activation d'une unité pourra déclencher l'exécution d'une fonction complexe qui serait codée dans le simulateur du réseau. Cette approche nous permettrait d'avoir des unités spécialisées dans des tâches complexes et très spécifiques;

- l'incorporation du temps dans les réseaux. INSS ne peut traiter le temps qu'en changeant la façon de coder les données (fenêtre temporelle - contexte). On peut envisager le développement d'un système hybride comme INSS basé sur des réseaux récurrents.

Pour compléter, l'application d'une *méthode d'apprentissage actif* pourrait être envisagé dans le cadre du système INSS. Par apprentissage actif, nous entendons la capacité d'un système d'apprentissage automatique d'interagir avec l'environnement (expert, sources de données). Par contre, un système va être considéré comme passif quand il n'a pas de réactions (Il accepte tout) face à des informations qui lui arrivent. Par exemple, le système INSS pourrait demander automatiquement à l'expert d'avantage d'exemples autour de certaines connaissances mal apprises. Le système peut ainsi faciliter et améliorer la tâche d'acquisition de ses connaissances.

Dans un cadre plus général, à propos des systèmes hybrides, nous pensons qu'une voie intéressante de recherche est l'étude d'un système qui puisse combiner différentes approches dans un seul outil. L'approche utilisée par SCANDAL pourrait servir à combiner des systèmes tel que INSS (réseaux de type PMC), SYNHESYS (réseaux à prototypes) et ProBis (CBR+ANN, avec la possibilité de traiter les cas atypiques).

Finalement, un des points très importants de l'étude du système INSS a été son application à des problèmes pratiques pour mieux connaître ses propriétés ainsi que ses limitations. Nous avons testé le système sur différents types d'applications que nous allons présenter dans le chapitre suivant : les problèmes du Moine, l'aide au diagnostic médical, le problème de la balance et la robotique autonome.

5. APPLICATIONS

Le système INSS a été testé sur 4 problèmes différents. Nous avons d'abord choisi d'appliquer INSS à un problème académique (*toy problem*), les problèmes du Moine [THR 91], afin de pouvoir mieux mesurer les résultats de son utilisation et de comparer sa performance par rapport à d'autres systèmes d'apprentissage automatique. Le fait d'avoir choisi un problème artificiel, nous a permis de mieux exploiter l'application, car le problème était bien connu et nous avions le contrôle sur toutes les données du problème. Ensuite, nous sommes passés à un autre problème plus complexe et réel, l'aide au diagnostic médical des comas toxiques [AMY 97]. Ce problème est beaucoup plus difficile à traiter, et d'ailleurs il n'a pas toujours été résolu de façon très satisfaisante. C'est sa difficulté qui faisait son intérêt.

Les autres applications sont très différentes. Les deux premières applications étaient des problèmes de classification 'classiques', tandis que les deux autres applications abordent des sujets plus spécifiques. La troisième application, le problème de la balance, a été étudié par des chercheurs afin de modéliser le développement cognitif chez l'enfant [SCH 94a]. Pour nous, l'intérêt de ce problème était plutôt lié au type de règles symboliques (règles de haut niveau) utilisées pour coder les connaissances sur le sujet. Finalement, la dernière application a été l'utilisation d'INSS dans un problème de contrôle d'un robot autonome, le Khepera [MON 93]. Cette application nous a permis d'étudier certains aspects liés au contrôle sensori-moteur d'un robot, d'abord sur un simulateur, puis sur un vrai robot.

En utilisant INSS sur ces quatre applications, nous avons essayé de mettre en valeur les propriétés de notre système, ainsi que de valider l'implémentation qui a été faite.

Dans les sections qui suivent, nous allons décrire plus en détails ces applications et les résultats obtenus en utilisant INSS. Avant de passer aux applications, nous voulons souligner d'une part que l'évaluation des systèmes connexionnistes est une tâche difficile étant donné le manque d'une méthodologie adaptée[PRE 94a] ; d'autre part que l'évaluation des systèmes

hybrides comme INSS est encore plus difficile, car il faut trouver des problèmes comportant une base de règles et une base d'exemples. Les quatre applications sur lesquelles nous avons travaillé, sont composées de problèmes qui englobent des connaissances théoriques (qui peuvent êtres exprimées par des règles du type accepté par INSS) et des connaissances empiriques (base d'exemples).

5.1 LES PROBLEMES DU MOINE (THE MONK'S PROBLEMS)

Nous avons choisi d'utiliser un problème très connu dans le domaine de l'apprentissage automatique, car il a été conçu pour l'évaluation de différents algorithmes. Les problèmes du Moine (*Monk's problems*) ont été proposés dans la 2ème école d'été sur l'apprentissage automatique, afin de comparer différentes méthodes, telles que : les arbres de décision (ID3, ID5R, TDIDT, Assistant Professional, PRISM), les algorithmes de la famille AQ (AQ15, AQ17, AQR, CN2), les méthodes de programmation logique inductive (ILP - mFOIL), les méthodes de construction de prototypes (CLASSWEB, COBWEB), et les réseaux de neurones artificiels (Rétro-Propagation, Cascade-Correlation) [THR 91]. Les "problèmes du Moine" sont ainsi appelés simplement parce que le groupe de travail qui a travaillé sur ces problèmes s'est réuni dans un monastère pendant le déroulement de l'école.

5.1.1 Description du Problème

Ce problème est constitué par trois sous-problèmes de classification de 'robots artificiels' : monk1, monk2, monk3. Etant donné une base d'apprentissage, nous essayons d'apprendre la fonction qui nous permettra de classifier correctement les exemples dans deux catégories : ceux qui appartiennent à la classe Monk (c'est un robot) et ceux qui n'appartiennent pas à cette classe (ce n'est pas un robot). La Figure 57 donne une description des six variables d'entrées applicables aux trois sous-problèmes.

Variables d'entrée:

```
$Features:
        HEAD_SHAPE  :  nominal # 3  : Octagon, Square, Round;
        BODY_SHAPE  :  nominal # 3  : Octagon, Square, Round;
        IS_SMILING    :  nominal # 2  :  No, Yes;
        HOLDING        :  nominal # 3  : Flag, Balloon, Sword;
        JACKET_COLOUR  :  nominal # 4  : Blue, Green, Yellow, Red;
        HAS_TIE          :  nominal # 2  :  No, Yes.
$End_Feat.
```

Règles Symboliques - Monk1:

```
$Rules:
   Monk1 <- HEAD_SHAPE(Round), BODY_SHAPE(Round);          Règle I
   Monk1 <- HEAD_SHAPE(Square), BODY_SHAPE(Square);        Règle II
   Monk1 <- HEAD_SHAPE(Octagon), BODY_SHAPE(Octagon);      Règle III
   Monk1 <- JACKET_COLOUR(Red).                            Règle IV
$End_rules.
```

Figure 57 - Description du problème du Moine: "Monk's Problem 1"

Le problème Monk1 est constitué par une base d'apprentissage de 124 exemples. Cette base est obtenue à partir de la base complète qui comporte 432 exemples, c'est-à-dire toutes les possibilités de combinaison des valeurs des six entrées. Dans le Monk1, il n'y a pas d'exemples classés incorrectement, et la règle que nous devons apprendre est (voir aussi Figure 57) :

Si Head_Shape = Body_Shape ***ou*** Jacket_Color = Red

Alors Monk1 est vrai (c'est un robot)

Le problème Monk2 est constitué par une base d'apprentissage de 169 exemples, sans exemples mal classés. La règle codée dans les exemples est :

Si deux et seulement deux des attributs d'entrée prennent leurs premières valeurs

Alors Monk2 est vrai (c'est un robot)

Le problème Monk3 est constitué par une base d'apprentissage de 122 exemples, dont 5% des cas sont des exemples mal classés (la classe qui leur est attribuée est incorrecte). La règle codée dans les 'bons' exemples est :

Si (Jacket_Color = Green ***et*** Holding = Sword) ***ou***

(Jacket_Colort ≠ Blue ***et*** Body_Shape ≠ Octagon)

Alors Monk3 est vrai (c'est un robot)

5.1.2 Les Résultats Expérimentaux obtenus avec INSS

Dans nos expériences, nous nous sommes concentrés sur le premier ensemble de règles et d'exemples proposés dans les problèmes du Moine - le problème Monk1. La Figure 57 montre la définition symbolique complète de ce problème dans le langage accepté par NeuComp. Les deux autres sous-problèmes, le Monk2 et Monk3, ont été aussi étudiés, mais cela a été fait plutôt afin de tester le bon fonctionnement des algorithmes que nous avons réalisé (Nous avons aussi utilisé d'autres problèmes d'apprentissage tels que le 'Xor' et la 'double spirale' [FAH 90]). Les résultats que nous avons obtenus avec le problème Monk1 servent à montrer les principales caractéristiques de notre système. Nous avons réalise deux groupes principaux d'expériences avec ce problème.

Premièrement, nous avons utilisé INSS afin de mettre en valeur deux de ses propriétés principales : la possibilité d'utiliser et d'intégrer des connaissances théoriques et empiriques ; et la capacité de profiter des connaissances théoriques de départ afin d'accélérer le processus d'apprentissage. Pour faire cela, nous avons pris l'ensemble d'apprentissage composé de 124 exemples (ensemble d'exemples originaux) et l'ensemble de tests de généralisation composé de 432 exemples (ensemble d'exemples originaux qui couvrent tout l'espace d'entrées). Nous avons commencé par la réalisation de l'apprentissage à partir des exemples, en utilisant toute la base d'apprentissage (124 exemples) et après en réduisant cette base à 75% de sa taille originale (93 exemples) et à 50% de sa taille originale (62 exemples). Ensuite, nous avons testé la capacité de généralisation du système en réalisant la compilation de règles sous la forme d'un réseau, et en

faisant son activation pour tous les 432 exemples de la base de test. Dans ces expériences, nous avons utilisé tout d'abord 100% des règles (les 4 règles), après 75% des règles (3 règles), et finalement seulement 50% des règles (2 règles). Finalement, nous avons utilisé des combinaisons des partitions de la base d'exemples et de la base de règles, afin de voir le comportement de notre système en présence d'un ensemble incomplet de connaissances théoriques et empiriques. Dans ces dernières expériences, nous avons tout d'abord inséré la base de règles disponible dans un réseau, et ensuite nous avons raffiné les connaissances par apprentissage à partir des exemples disponibles. Les résultats obtenus dans ces expériences, sont présentés dans le Tableau 7, où nous présentons les taux de généralisation (mesurés par l'activation du réseau en utilisant tous les 432 exemples de la base de test) et les nombres d'époques nécessaires pour arriver à ces résultats ('BestEpoch').

Portion des Règles	Portion des Exemples	Généralisation avec les règles (MS)	Généralisation* avec les exemples (MC)	Généralisation* avec INSS (Syst. Hybride)
-	100%	-	100% - 108.2 époques	**100% - 108.2 époques**
-	75%	-	89.21% - 383.4 époques	**89.21% - 383.4 époques**
-	50%	-	70.92% - 303.8 époques	**70.92% - 303.8 époques**
100%	-	100%	-	**100%**
75%	-	83.33%	-	**83.33%**
50%	-	72.22%	-	**72.22%**
75%	100%	83.33%	100% - 108.2 époques	**100% - 59.4 époques**
50%	100%	72.22%	100% - 108.2 époques	**100% - 57.6 époques**
75%	75%	83.33%	89.21% - 383.4 époques	**100% - 57.4 époques**
75%	50%	83.33%	70.92% - 303.8 époques	**100% - 61.2 époques**
50%	75%	72.22%	89.21% - 383.4 époques	**100% - 61.0 époques**
50%	50%	72.22%	70.92% - 303.8 époques	**89.86% - 62.0 époques**

* Les taux de généralisation représentent la moyenne obtenue à partir de 5 tests différents (poids initiaux différents)

Tableau 7 - Problème Monk1 : Utilisations simultanées de règles et d'exemples pour améliorer le taux de généralisation final et la vitesse d'apprentissage

En se basant sur les résultats présentés dans le Tableau 7, nous pouvons affirmer que le système INSS a bien su exploiter la complémentarité entre les connaissances des bases d'exemples et de règles. La Figure 58 montre plus nettement *l'avantage de l'utilisation de notre méthode hybride sur des méthodes isolées (MS ou MC).* L'INSS surpasse toujours les résultats de

généralisation obtenus par l'utilisation d'un seul type d'approche. De plus, on peut remarquer dans le Tableau 7 le faible nombre d'époques nécessaires pour l'apprentissage avec INSS. On remarque que *l'apprentissage est toujours plus rapide quand on utilise des connaissances a priori.*

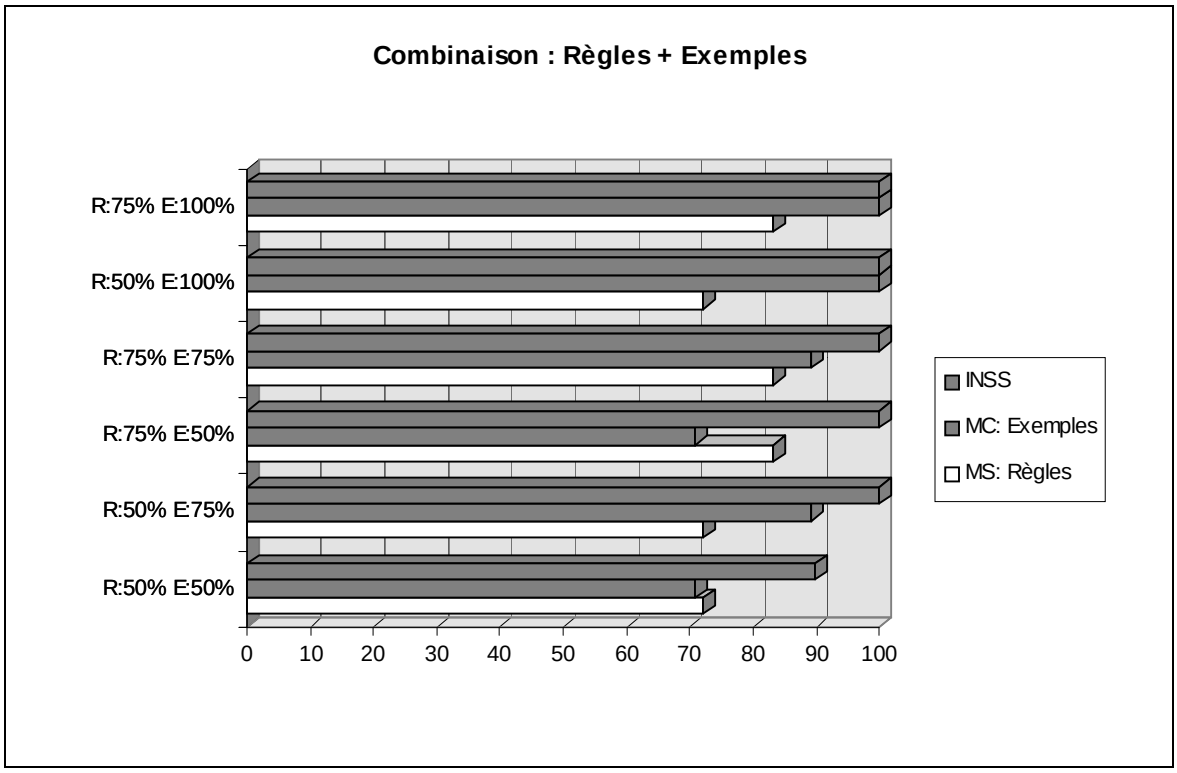


Figure 58 - Monk1 : Taux de généralisations du MS, MC et Système Hybride

Notre deuxième groupe d'expériences avec le problème Monk1 portait sur l'extraction de règles. Nous avons fait des expériences pour savoir si le système était capable de retrouver l'ensemble complet des règles à partir des connaissances incomplètes raffinées par l'apprentissage d'une base d'exemples complète. Notre première expérience a consisté à créer un réseau avec la compilation de 75% des règles (3 parmi les 4 règles disponibles), puis à appliquer la méthode d'explicitation de règles. Le processus d'extraction a été appliqué seulement sur deux unités, la sortie et une unité cachée, car seulement une unité a été ajoutée au réseau pendant l'apprentissage. Nous avons répété ce test pour toutes les différentes configurations des ensembles incomplets de règles : une règle éliminée parmi les quatre disponibles. Dans tous les cas testés, *la méthode d'extraction nous a permis de retrouver la bonne règle qui avait été éliminée de l'ensemble d'origine.* Cette règle éliminée se trouvait bien représentée dans les nouvelles connaissances

acquises par le réseau. La Figure 59 montre un exemple des règles obtenues dans une des expériences que nous avons réalisées. Il faut remarquer que les règles extraites sont en fait des règles équivalentes aux règles recherchées.

Règle éliminée : règle III

Monk1 <- Head_Shape (Octagon), Body_Shape (Octagon);

Règles extraites :

New_Unit <- Body_Shape (Round);
New_Unit <- Body_Shape (Square);
New_Unit <- Head_Shape (Round);
New_Unit <- Head_Shape (Square);
New_Unit <- not (Head_Shape (Octagon)), not (Body_Shape(Octagon));

Final_Monk1 <- Monk1_75%;
Final_Monk1 <- not (NewUnit);

Règles après manipulation : simplification de la négations final

New_Unit <- Head_Shape (Octagon)),
not (Head_Shape(Round)), not (Head_Shape (Square)),
not (Body_Shape(Round)), not (Body_Shape (Square));
New_Unit <- Body_Shape (Octagon)),
not (Head_Shape(Round)), not (Head_Shape (Square)),
not (Body_Shape(Round)), not (Body_Shape (Square));

Final_Monk1 <- Monk1_75%;
Final_Monk1 <- NewUnit;

Règles finales :

Nous savons que :

Si Head_Shape (Octagon)
Alors not (Head_Shape (Round)) et not (Head_Shape (Square))

et

Si Body_Shape (Octagon)
Alors not (Body_Shape (Round)) et not (Body_Shape (Square))

Donc, les règles peuvent être simplifiées encore plus :

New_Unit <- Head_Shape (Octagon)), Body_Shape (Octagon)),
Final_Monk1 <- Monk1_75%;
Final_Monk1 <- NewUnit;

Figure 59 - Problème Monk1 : Règles Extraites

Dans une autre expérience, nous avons éliminé 50% des règles (règles III et IV). Après l'apprentissage des exemples (la base d'apprentissage utilisée est toujours la base d'origine composée de 124 exemples), nous avons pu retrouver les deux règles éliminées à travers l'utilisation du module Extract du système INSS.

En conclusion, nous avons réussi a montrer avec ce dernier groupe d'expériences que le système INSS est capable de prendre une base de connaissances initiales (règles), de la convertir dans un réseau, d'ajouter de connaissances par apprentissage, et enfin d'expliciter les nouvelles connaissances acquises.

5.1.3 Discussion Finale sur l'étude des Problèmes du Moine

Nous avons montré avec les expériences réalisées sur les problèmes du Moine que :

- Le système INSS donne des très bons résultats quand de sont utilisation avec ces problèmes. Nous pouvons comparer nos résultats avec ceux qui ont été obtenus dans l'école européenne d'apprentissage automatique en utilisant d'autres méthodes [THR 91]. Notre système est au moins tout aussi performant que les autres systèmes étudiés dans cette école (il arrive à dépasser plusieurs méthodes). De plus, il offre tous les avantages de l'utilisation d'une approche hybride neuro-symbolique;

- Nous avons montré ici que la combinaison des connaissances théoriques (règles) et des connaissances empiriques (exemples), en utilisant une approche hybride, nous permet d'améliorer les performances du système. Le système INSS est capable d'utiliser les différentes connaissances disponibles sur le problème de façon à accélérer le processus d'apprentissage et à augmenter sa capacité de généralisation par rapport à l'utilisation d'une seule source de connaissances;

- Nous avons montré que notre système est capable d'expliciter les nouvelles connaissances acquises à travers l'extraction de règles. INSS est capable de recevoir des connaissances initiales théoriques décrites par des règles, de les raffiner par apprentissage et de trouver les nouvelles connaissances acquises. Dans nos expériences, nous avons montré qu'il est possible de retrouver

les règles que nous avons intentionnellement effacées de l'ensemble complet de connaissances sur le problème;

- Nos expériences montrent que le système INSS est un outil qui possède les principales caractéristiques nécessaires à un système d'acquisition constructive de connaissances : l'insertion de connaissances a priori sur le problème, le raffinement des connaissance avec la possibilité de faire évoluer la structure des réseaux, et finalement l'extraction des nouvelles connaissances acquises.

Une description complémentaire des expériences que nous avons réalisée sur l'apprentissage et l'extraction de règles dans les problèmes du Moine, peut être obtenue dans d'autres publications corrélées avec cette thèse [OSO 95a, DEC 95, DEC 96].

5.2 DIAGNOSTIC MEDICAL - COMAS TOXIQUES

Le système INSS a été testé sur l'application médicale : aide au diagnostic des comas toxiques induits par psychotropes. Après avoir décrit le problème, nous présenterons les résultats obtenus avec l'utilisation de notre système.

5.2.1 Description du Problème

Quand un patient comateux est admis à l'unité de traitement intensif, le médecin fait une tentative de diagnostic préalable basé sur certains paramètres cliniques et biologiques. Ce diagnostic permet d'appliquer des soins initiaux urgents au patient, en attendant la confirmation (ou l'infirmation) postérieure du diagnostic à l'aide d'examens toxicologiques. Etant donné l'état critique dans lequel peuvent se trouver les patients, et compte tenu du fait que les examens d'analyse toxicologique peuvent tarder à être obtenus, ce diagnostic préalable doit impérativement être fait très rapidement. L'utilisation d'un système expert d'aide au diagnostic peut être très utile dans ce type de situation.

En relation avec ce problème, il faut souligner que :

- il n'existe pas aujourd'hui de modèle complet décrivant à l'aide de règles les connaissances sur ce domaine de diagnostic;

- Le problème consiste en une tâche de classification. On donne en entrée les symptômes du patient, en sortie on essaie d'obtenir une liste des substances qui ont été probablement ingérées par la personne;

- Nous pouvons disposer d'exemples obtenus à partir des cas préalablement traités, et dans lesquels le diagnostic a été confirmé ou invalidé par des examens toxicologiques (base de données supervisées);

- Notre but n'est pas de remplacer le médecin (celui-ci ne pourra jamais être remplacé par un système automatisé), mais de créer un outil *d'aide* au diagnostic. Les systèmes experts représentent l'image d'un "conseiller virtuel".

5.2.2 Description des données disponibles sur le problème

La base de cas a été conçue par le docteur Vincent Danel et Philippe Cony à l'unité de toxicologie clinique dans le Centre Hospitalier Universitaire de Grenoble - CHU Grenoble. Les études réalisées par le docteur Danel ont été faites en partie dans le cadre de sa thèse [DAN 97]. La base de cas qu'il a mise à notre disposition est une base de 505 cas bien analysés. Un cas est constitué de deux parties :

- La description du problème : le problème est décrit par 13 paramètres ou symptômes qui ont été sélectionnés par les médecins comme des attributs significatifs par rapport au diagnostic à effectuer (paramètres médicalement pertinents, et si possible discriminants). De plus, tous ces paramètres peuvent être obtenus immédiatement dès l'admission du patient. Nous présentons la liste de définition des 13 attributs dans le Tableau 8;

- Le psychotrope ou la combinaison de psychotropes étant absorbés par le patient : il existe 28 combinaisons différentes des 7 substances toxiques que nous nous proposons d'identifier. La liste des 7 substances (psychotropes + alcool) et de leurs combinaisons est donnée dans le Tableau 9. Dans cette figure, nous fournissons les abréviations utilisées (nomenclature utilisée dans la base de données), ainsi que le nombre de cas de la base où chaque substance a été détectée.

	Symptômes généraux		
	Variable	**Nature**	**Valeurs**
1	Sexe - SEXE	Discrète	{féminin, masculin} - 0/1
2	Température - TEMP	Continue	{ ... }
	Temp_Low	Discrète	0/1
	Temp_Normal	Discrète	0/1
	Temp_High	Discrète	0/1
	Symptômes	*neurologiques*	
3	Etat - CALME	Discrète	{calme, agité} - 0/1
4	Photomoteurs - PHOTO (réaction des pupilles)	Discrète	{présent, absent} - 0/1
5	Regard - REGARD	Discrète	{normal, errance} - 0/1
6	Etat des Pupilles - PUPIL	Discrète	{myosis, intermédiaire, mydriase}
	Pupil_myosis	Discrète	0/1
	Pupil_intermed	Discrète	0/1
	Pupil_mydriase	Discrète	0/1
7	Tonus musculaire - TONUS	Discrète	{hypertonie, hypotonie}
8	Réflexe des tendons - ROT	Discrète	{vifs, diminués}
	Symptômes	*cardiologiques*	
9	Pression Artérielle - PAS	Continue	{...}
	PAS_Low	Discrète	0/1
	PAS_Normal	Discrète	0/1
	PAS_High	Discrète	0/1
10	Fréquence Cardiaque - FC	Continue	{...}
	FC_Low	Discrète	0/1
	FC_Normal	Discrète	0/1
	FC_High	Discrète	0/1
11	Mesure ECG - QRS	Continue	{...}
	QRS_Normal	Discrète	0/1
	QRS_Prolonged	Discrète	0/1
12	Intervalle QT - QT	Discrète	{normal, allongé} - 0/1
		Urine	
13	Globe vésical - GLOBE	Discrète	{oui, non} - 0/1

Tableau 8 - Les 13 paramètres d'entrée décrivant un cas d'intoxication

	Abréviations	***Subst. Toxique***	***Occurrence***
1	adt, a	antidépresseurs tri-cycliques	265
2	B	barbituriques	86
3	ben, b	benzodiazépine	414
4	c	carbamate	68
5	p	phénothiazine	130
6	m	morphine	13
7	E	alcool	137
Combinaisons			
a (25), aBb (12), aBbp (9), ab (104), abc (11), abm (5), abp (38), ap (5), B (8), Bb (23), Bbcp (5), Bcp (13), ben (37), bc (24), bm (8), bp (31), c (5), p (5), Ea (12), Eab (36), Eabp (8), E (12), EB (6), EBb (5), EBbp (5), Eb (32), Ebc (10), Ebp (11)			

Tableau 9 - Les différentes substances toxiques avec les nombres respectifs d'occurrences dans la base

La base de données que nous avons utilisée initialement comptait 13 attributs, dont 8 sont des attributs binaires, 1 est un attribut ternaire et 4 autres sont des attributs a valeurs continues. Une autre base de données a été obtenue par discrétisation des attributs. On obtient ainsi 21 attributs tous binaires. La base de 13 attributs d'entrée est la base *Coma13* , la base avec 21 attributs d'entrée est la base *Coma21*.

Les données sur ce problème ont été étudiées au cours des recherches menées par :

- Les équipes intégrant le projet ESPRIT MIX [AMY 97, HIL 94a], dont l'équipe Réseaux d'Automates du laboratoire Leibniz;
- Le Docteur V. Danel, dans le cadre de sa thèse [DAN 97];
- Maria Malek, dans le cadre de sa thèse sur les systèmes hybrides (système PROBIS : CBR+ANN);
- Les personnes attachées au projet SYMEDIA (projet en cours avec la participation des Laboratoires Leibniz et TIMC de l'IMAG, du C.E.A de Grenoble et du C.H.U. de Grenoble). F. Osório fait partie ce groupe de recherche.

Plusieurs techniques d'apprentissage automatique ont été appliquées sur ces données dont : les graphes de décision et le sytème SIPINA (V.Danel), le système hybride Probis (M.Malek) et le système hybride INSS (F.Osório). Initialement, nous avons suivi le même type de procédure

adopté par M. Malek pour traiter ce problème. Dans sa thèse, M. Malek démontre que la base de données qui comporte 28 classes différentes en sortie, n'est pas traitable telle qu'elle. Les relations entre le nombre de cas, le nombre d'attributs d'entrée et le nombre de classes ne permettent pas de bien discerner les classes, car "l'ensemble des cas dont nous disposons ne couvre pas le domaine, et n'est pas suffisant pour définir les frontières entre les 28 classes différentes" [MAL 96b]. M. Malek a proposé de décomposer le problème en plusieurs sous-tâches, afin de faciliter sa résolution. Donc, au lieu d'essayer de classer les 505 cas dans 28 classes, nous avons essayé d'identifier, pour chacune des 7 substances toxiques, si elle a été absorbée ou non par le patient. Nous avons ainsi transformé un problème de classification à 28 classes en 7 sous-problèmes, chacun contenant une sortie unique qui indique la présence ou l'absence d'une des 7 substances toxiques.

5.2.3 Les Résultats Expérimentaux obtenus avec INSS

Nous avons réalisé différentes expériences avec les données disponibles sur ce problème. Notre première expérience a été faite avec la base *Coma13*, afin de comparer les performances (capacité de généralisation) du système INSS avec celles des autres systèmes. Nous avons repris les résultats publiés par Malek dans sa thèse [MAL 96b], et nous avons reproduit les mêmes expériences avec INSS. Le Tableau 10 présente les résultats d'apprentissage de la base Coma13 par rapport au taux de généralisation obtenu pour chacune des 7 substances toxiques. Le système INSS est comparé avec les méthodes K-PPV (K-plus-proches-voisins), C4.5 (arbres de décision) et Probis (système hybride CBR+ANN).

Classe \ Méthode	K-PPV	C4.5	ProBIS	INSS
Alcool -E	66.56%	65.40%	68.94%	**74.50%**
ADT - a	55.39%	55.26%	57.63%	**60.79%**
Barbituriques - B	65.65%	63.32%	64.60%	**82.45%**
Benzodiazepines - b	62.37%	64.34%	63.95%	**83.37%**
Carbamates - c	81.58%	**87.64%**	84.87%	87.28%
Morphine - m	97.23%	97.50%	**97.97%**	97.88%
Phenothiazines - p	66.45%	71.26%	68.95%	**75.36%**

Tableau 10 - Comparaison des taux de généralisation après apprentissage

Dans ces expériences, nous avons utilisé les données et procédures suivantes :

- La base Coma13 est composé de 505 cas décrits à l'aide de 13 attributs d'entrée et un seul de sortie. Les 505 cas sont dupliqués dans 7 fichiers, chacun contenant une sortie indiquant le diagnostic relatif à une des 7 substances toxiques à identifier;

- Nous avons divisé aléatoirement les 505 cas disponibles en deux parties : un fichier d'apprentissage avec 70% des cas (354 exemples) et un fichier de test de généralisation avec les 30% restants des cas (151 exemples);

- Les résultats obtenus avec INSS représentent une moyenne de 10 tests différents. Chaque test a été réalisé à partir de la création d'un réseau initialisé avec des valeurs des poids choisies aléatoirement (différents initialisations pour chacun des tests);

- Les résultats indiqués dans les colonnes K-PPV, C4.5 et Probis sont une reproduction des meilleurs résultats obtenus pour chacune des méthodes, tels que nous les trouvons publiés dans la thèse de M.Malek [MAL 96b]. Ces résultats ont été obtenus en utilisant des fichiers d'apprentissage et de test réalisés à partir de la même base de données d'origine (Coma13) et repartis de la même façon que dans nos tests (70% apprentissage - 30% test). La seule différence est que les résultats finaux représentent une moyenne de seulement 5 tests différents;

- Dans tous nos tests avec INSS, nous avons utilisé la méthode d'apprentissage CasCor, sans insertion de connaissances a priori. Le critère d'arrêt de l'apprentissage était : soit après 1200 époques, soit avec une erreur de sortie inférieure à 0.05 (StopErr). Nous avons réalisé une époque de test après chaque époque d'apprentissage afin de trouver quelle a été la meilleure époque (BestEpoch) et le meilleur taux de généralisation obtenu pendant l'apprentissage.

Une description plus détaillée des résultats obtenus avec INSS dans cette expérience peut être trouvée dans le rapport MIX sur l'application médicale [AMY 97]. La première conclusion sur ces expériences est que le système INSS a une performance très convenable par rapport à ce problème. La performance d'INSS est remarquable vis-à-vis des autres techniques. On peut constater aussi que cette application est assez complexe, puisque pour certaines classes toutes les méthodes appliquées n'ont pas réussi a avoir des résultats plus satisfaisantes. Ainsi pensons-nous avoir réussi à montrer que le système INSS peut être appliqué à des problèmes réels complexes tout aussi bien qu'à des problèmes académiques.

Toutefois, les résultats que nous venons de présenter ne permettent pas de mettre en valeur les autres propriétés du système, qui font la différence entre notre approche et les autres techniques classiques d'apprentissage automatique. C'est pourquoi nous avons essayé d'approfondir nos recherches sur ce problème. Notre but était aussi de mieux comprendre pourquoi les résultats obtenus n'étaient pas très satisfaisants dans certains cas. Notre démarche a été la suivante : faire l'extraction de règles après l'apprentissage, afin de pouvoir valider les connaissances acquises par le réseau auprès d'un expert du domaine.

Comme l'extraction de règles faite par INSS, avec la version actuelle du système, ne supporte que des règles d'ordre 0, nous avons été obligés d'utiliser la base de données *Coma21*. Dans cette base, tous les attributs d'entrée ont été discrétisés (entrées binaires). La base Coma21 est la base de données qui a été choisie pour être testé dans le projet MIX.

Nous avons commencé nos tests avec les données relatives à la classe de psychotrope qui a obtenu le meilleur taux de généralisation - la morphine avec 97.88% de bonnes réponses. De plus, nous savions que les cas de diagnostic positif de cette classe étaient très peu présents dans la base de données (seulement 13 cas parmi les 505). Nous avions des doutes sur la qualité des réponses du réseau relatives à l'identification de l'absorption de morphine, qui ont été vite confirmés par l'extraction des règles. Voici la règle extraite à partir du réseau préalablement entraîné avec la base Coma21 dans le cas de d'identification de la classe morphine - 'm' :

```
$RULES:
$END_RULES.
```

Elle montre qu'il n'y a pas de règles extraites, c'est-à-dire que le réseau ne va jamais identifier un cas comme appartenant à la classe 'm'. Les réponses du réseau analysé sont toujours négatives par rapport à l'identification de la présence de la morphine. Cela n'a rien d'étonnant, car le petit nombre de cas de la classe 'm' présents dans la base d'apprentissage ne nous permet pas d'arriver à apprendre vraiment à bien les classer. En ce qui concerne le taux de 97.88% de bonnes

réponses, il est assez proche du taux de bonnes réponses obtenu si l'on ne donne que des réponses négatives, car la base est composée par 97.43% de cas négatifs.

Ensuite, nous sommes passés à l'application des outils de validation de connaissances disponibles dans INSS. Nous avons donc sorti le nombre de cas 'faux_0' et de 'faux_1' obtenus en activant le réseau après l'apprentissage avec toute la base de 505 cas. Le résultat était évident : toutes les erreurs produits par le réseau étaient d'un seule type - 13 cas de 'faux_1'. Le fichier 'data.err' créé par INSS ne contenait que des cas positifs de la classe 'm'. Le réseau a toujours répondu avec une sortie négative, même pour les 13 cas positifs existants dans la base de données complète. Il n'y a pas de doute, le meilleur taux de bonnes réponses qu'on peut avoir pour la classe 'm' avec l'apprentissage de la base Coma21 est celui d'une réponse unique toujours négative.

Classe	Cas Positifs	Cas Négatifs	X	Non(X)
Alcool - E	137	368	27.12%	**72.88%**
ADT - a	265	240	**52.47%**	47.53%
Barbi - B	86	419	17.02%	**82.98%**
Benzo - b	414	91	**81.98%**	18.02%
Carba - c	68	437	13.46%	**86.54%**
Morph - m	13	492	2.57%	**97.43%**
Pheno - p	130	375	25.74%	**74.26%**

Tableau 11 - Distribution des cas dans la base de données (Coma13/Coma21)

Nous avons voulu aller encore plus loin dans les expériences faites à partir de cette base. Tout d'abord, nous avons établi une liste du nombre de cas positifs et négatifs de chaque classe (voir Tableau 11). Ensuite, nous avons constaté que le taux de bonnes réponses de INSS pour la classe 'b' (83.37%) était assez proche du pourcentage de cas positifs liés à cette classe ('benzo' = 81.98%). Nous avons donc fait l'extraction de règles à partir du réseau Coma21-Benzo. La règle obtenue est la suivante :

```
$RULES:
        Benzo <- .
$END_RULES.
```

Elle indique que la conclusion 'Benzo' va être toujours positive, car il n'y a pas de conditions qui s'imposent pour que la sortie soit activée. Tous les cas seront classés comme des cas appartenant à la classe 'b' (benzo positif). On peut constater dans le Tableau 11 que cette classe possède une majorité d'exemples de cas positifs, par opposition à la classe 'm' que nous avons analysée précédemment. Comme prévu, le réseau entraîné avec la base Coma21-Benzo ne présente que des sorties du type 'Faux_0' pour tous les 505 cas de la base.

Pour conclure nos expériences sur l'extraction de règles dans cette application, nous avons choisi de travailler sur un réseau qui a appris la base Coma21-Adt (un des trois psychotropes choisis pour être étudié plus en détails dans le projet MIX [AMY 97]). Après application des méthodes de simplification du réseau, d'extraction de règles, et de sélection des règles obtenues, nous avons obtenu l'ensemble de règles présenté ci-dessous :

```
%%
%% Extracted rules
%% Score over 505 examples: 64.35% correct answers [19 rules]
%%
$Rules:

ADT <- GLOBE, PUPIL_INTERMED, not(FC_LOW), not(PUPIL_MYOSIS), not(REGARD), not(TEMP_HIGH),
       not(SEXE) ;
ADT <- QTC, PUPIL_INTERMED, not(FC_LOW), not(PUPIL_MYOSIS), not(REGARD), not(TEMP_HIGH) ;
ADT <- GLOBE, TEMP_LOW, not(PUPIL_MYOSIS), not(REGARD), not(TEMP_HIGH) ;
ADT <- QTC, TEMP_LOW, not(FC_LOW), not(REGARD), not(TEMP_HIGH), not(SEXE) ;
ADT <- QTC, TEMP_LOW, not(REGARD), not(TEMP_HIGH), not(TEMP_NORMAL), not(SEXE);
ADT <- QTC, TEMP_LOW, not(FC_LOW), not(PUPIL_MYOSIS), not(REGARD) ;
ADT <- QTC, TEMP_LOW, not(PUPIL_MYOSIS), not(REGARD), not(TEMP_HIGH);
ADT <- QTC, TEMP_LOW, not(PUPIL_MYOSIS), not(REGARD), not(TEMP_NORMAL);
ADT <- PUPIL_INTERMED, TEMP_LOW, not(FC_LOW), not(PUPIL_MYOSIS), not(REGARD), not(TEMP_NORMAL),
       not(SEXE);
ADT <- PUPIL_INTERMED, TEMP_LOW, not(PUPIL_MYOSIS), not(REGARD), not(TEMP_HIGH) ;
ADT <- PUPIL_INTERMED, TEMP_LOW, not(PUPIL_MYOSIS), not(REGARD), not(SEXE) ;
ADT <- QTC, not(FC_LOW), not(PUPIL_MYOSIS), not(REGARD), not(TEMP_HIGH), not(TEMP_NORMAL) ;
ADT <- QTC, not(PUPIL_MYOSIS), not(REGARD), not(TEMP_HIGH), not(SEXE) ;
ADT <- PUPIL_MYDRIASE, not(FC_LOW), not(PUPIL_MYOSIS), not(REGARD), not(TEMP_HIGH) ;
ADT <- PUPIL_INTERMED, not(PUPIL_MYOSIS), not(REGARD), not(TEMP_HIGH), not(TEMP_NORMAL),
       not(SEXE) ;
ADT <- TEMP_LOW, not(FC_LOW), not(PUPIL_MYOSIS), not(REGARD), not(TEMP_HIGH) ;
ADT <- TEMP_LOW, not(PUPIL_MYOSIS), not(REGARD), not(TEMP_HIGH), not(TEMP_NORMAL) ;
ADT <- TEMP_LOW, not(PUPIL_MYOSIS), not(REGARD), not(TEMP_HIGH), not(SEXE) ;
ADT <- TEMP_LOW, not(PUPIL_MYOSIS), not(REGARD), not(TEMP_NORMAL), not(SEXE).

$End_rules.
```

Figure 60 - Règles obtenues à partir d'un réseau (apprentissage de la base Coma21-adt)

Les règles obtenues à partir du réseau entraîné avec la base Coma21-Adt ont été présentées à un expert du domaine. Selon lui, ces règles peuvent être considérées comme un ensemble de connaissances valables capturant certaines relations implicites entre les symptômes du patient et le psychotrope en question. De plus, nous avons compilé les règles présentées sur la Figure 60 sous la forme d'un réseau, et ensuite nous avons activé ce réseau avec la base de 505 cas, obtenant un taux de 64.35% de bonnes réponses (résultat supérieur à celui obtenu avec les autres méthodes d'apprentissage automatique, cf. Tableau 10).

Une fois terminées les expériences sur l'extraction de règles, nous avons fait une dernière expérience avec la base de données disponible. Les résultats antérieurs nous ont amenés à mettre en question les attributs d'entrée choisis (sont-ils suffisants ?) et la division du problème en 7 sous-problèmes (sont-ils vraiment indépendants ?). Nous avons fait une expérience simple : prendre la base de données Coma21-Adt et ajouter les 6 autres sorties des autres classes comme des entrées du réseau. Ainsi, nous avons obtenu un réseau avec 27 entrées (les 21 symptômes + les 6 sorties : E, B, b, c, m, p) et une seule sortie (Adt). Notre but était de vérifier si les 6 autres sorties ont une influence significative sur la septième et dernière classe. Les résultats se sont avérés positifs, puisque nous sommes passés d'un taux moyen de réussite de 60.79% à 73.50% (avec le meilleur résultat obtenu égal à 79.40% de bonnes réponses). Les résultats obtenus sont non négligeables, car si l'on ajoute des nouvelles entrées totalement non corrélées à un problème, l'apprentissage du réseau doit se faire avec plus de difficulté. Nos résultats montrent exactement le contraire. Notre conclusion est : soit le réseau a augmenté sa performance à cause des interrelations entre les 7 classes de sortie (sorties non-indépendantes), soit il existe des variables d'entrée cachées qui n'ont pas été explicitées et qui ont été prises en compte pour le diagnostic des 6 autres classes (maintenant fournies comme entrées). Bien que nous n'ayons pas approfondi nos recherches sur ce sujet, les résultats décrits ci-dessus apportent une contribution importante à l'étude du problème et donnent des nouvelles pistes à suivre.

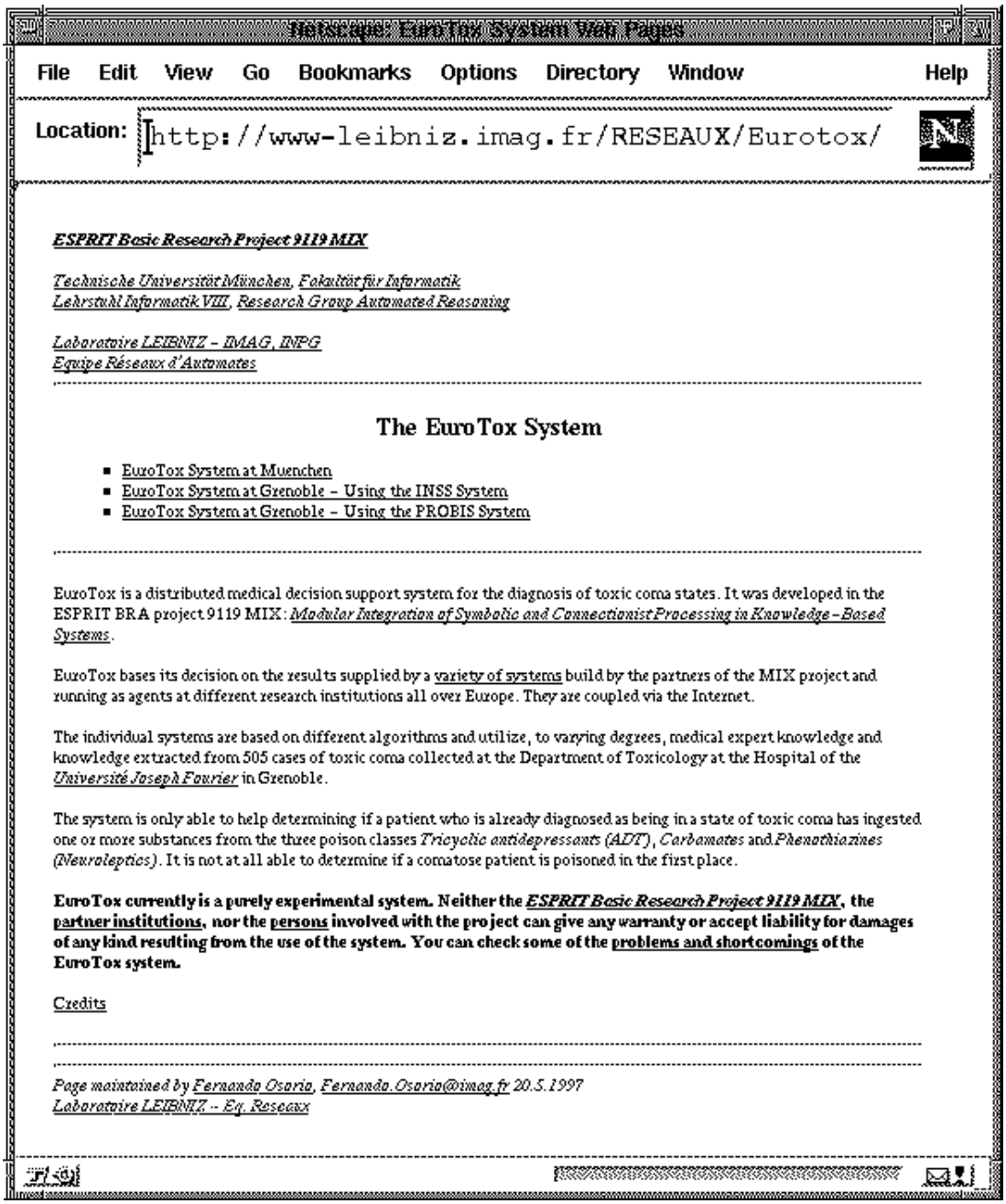


Figure 61 - Le Système Eurotox-INSS, Projet MIX

Finalement, nos recherches ont abouti au développement d'un outil expérimental de consultation de notre système à travers le WWW (World-Wide-Web). L'interface avec l'utilisateur été conçue dans le cadre du projet MIX (voir Figure 61). Elle consiste en un formulaire électronique qui permet à l'utilisateur de saisir les symptômes du patient et d'interroger plusieurs systèmes d'aide au diagnostic à travers le réseau Internet. Dans la Figure 62, nous

présentons un exemple de l'écran d'interface du système avec le formulaire. On peut accéder à cette interface à l'adresse suivante : *http://www-leibniz.imag.fr/RESEAUX/Eurotox/* .

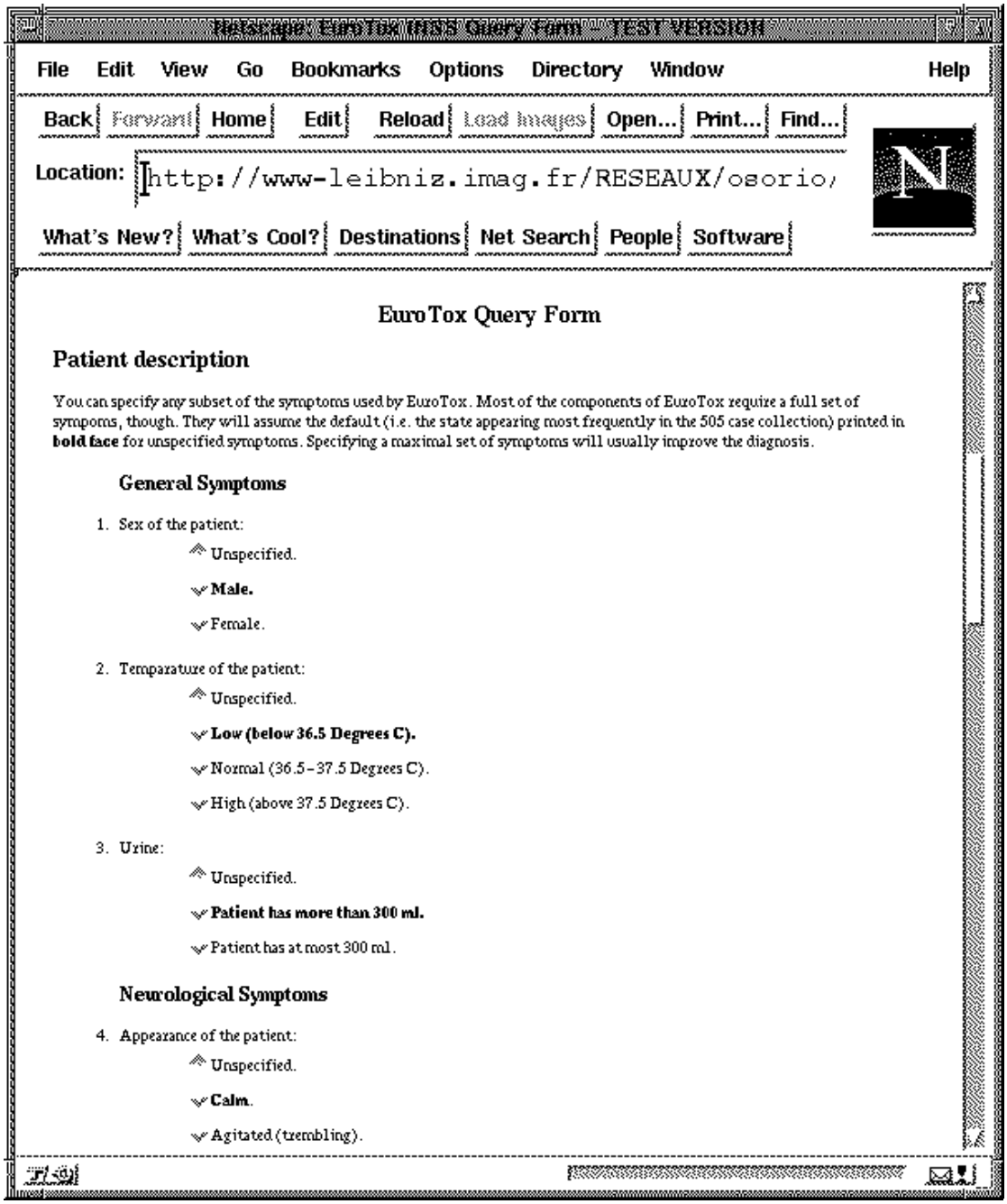


Figure 62 - Formulaire électronique pour la consultation du système Eurotox-INSS

L'intégration des outils du système INSS à cette interface a été faite très rapidement. Cela nous montre que le système INSS est un outil de conception modulaire et qu'il est assez simple de l'intégrer dans différentes applications.

5.2.4 Discussion Finale sur l'Application Médicale

Avec l'utilisation d'INSS, nous avons pu constater que la taille et la nature même de la base de données disponible ne permet pas de faire le diagnostic des 7 classes de substances toxiques d'une façon indépendante. Les outils de validation de connaissance nous ont permis de détecter et d'expliciter certains problèmes relatifs à l'apprentissage de la base disponible. Dans certaines situations, les autres méthodes d'apprentissage automatique ne permettent pas de déceler ces problèmes, qui passent alors inaperçus. L'exemple le plus évident des problèmes rencontrés est le cas où les outils d'aide au diagnostic d'une classe spécifique ont donné toujours une seule et unique réponse. Les taux de bonnes réponses étant satisfaisants, les utilisateurs des systèmes d'apprentissage avaient une tendance à négliger les signes montrant de que ce taux n'était pas une bonne mesure de la qualité du système. Les outils disponibles dans le système INSS, ainsi que la façon de présenter les résultats, amènent l'utilisateur à détecter plus facilement un problème comme celui-ci.

Nous avons montré que le système INSS a une performance très satisfaisante par rapport à d'autres outils d'apprentissage automatique. De plus, notre système s'est montré capable d'exploiter plus profondément les informations disponibles, grâce en particulier à la validation des connaissances et à l'extraction de règles. En utilisant INSS, nous avons pu aussi mettre en question certains choix qui ont été faits par les chercheurs impliqués dans l'étude de cette application. Les résultats obtenus avec INSS montrent qu'on doit étudier ce problème plus en détail, en réalisant une analyse plus approfondie des symptômes choisis en entrées et de la dépendance qui peut exister entre les classes de sortie.

En conclusion, notre approche de l'acquisition constructive de connaissances semble être très intéressant quand on est confronté à des applications réelles. La constante mise en question

des connaissances disponibles, leur validation et leur évolution, sont des points très importants dans un système d'acquisition de connaissances. Le problème d'aide au diagnostic médical des comas toxiques est un exemple de problème qui requiert l'utilisation d'une approche comme la nôtre. Il est en effet très complexe, et il présente des aspects qui peuvent être mieux traités en utilisant une approche d'acquisition constructive de connaissances.

5.3 PROBLEME DE LA BALANCE

Le système INSS a été utilisé dans un problème d'apprentissage ayant servi à modéliser le développement cognitif chez les enfants. Il s'agit du problème de la prévision de l'état d'une balance (*balance scale problem*) [SIE 76, MCC 89, SHU 94a]. Nous nous sommes intéressés à ce problème pour les raisons suivantes :

- Il porte sur une tâche de modélisation du développement cognitif chez l'homme (acquisition constructive de connaissances), cette modélisation étant basée sur des données réelles obtenus à partir d'études pratiques avec des enfants Les premières expériences sur le sujet ont été réalisées par des psychologues;

- Le problème d'apprentissage de la prévision de l'état de la balance a été en partie déjà modélisé en utilisant des techniques d'apprentissage automatique (arbres de décision et réseau de neurones);

- Les connaissances sur le domaine sont exprimées à l'aide d'une théorie (ensemble de règles) et de connaissances empiriques (ensemble de cas pratiques);

- Les résultats obtenus avec l'utilisation de l'algorithme CasCor ont attiré l'attention de la communauté des chercheurs du domaine de l'apprentissage automatique. Ces expériences, menées par l'équipe de Thomas Shultz, ont été publiées dans une des revues le plus prestigieuses du domaine de l'apprentissage automatique [SHU 94a].

Notre intérêt par ce problème et les premiers résultats obtenus dans nos expériences, nous ont conduits à proposer un sujet de stage de D.E.A. sur ce thème. L'étude de l'apprentissage du problème de la balance a été réalisée par Gerardo Reyes Salgado dans le cadre de sont stage de

D.E.A. [REY 97] réalisé sous la supervision de F. Osório. G. Reyes a étudié l'apprentissage du problème de la balance, ainsi que le problème de la représentation et de l'extraction de règles de haut niveau, un sujet fortement lié au problème en question.

5.3.1 Description du Problème

Le *problème de la balance équilibrée* (Balance Scale Problem) [SIE 76], [MCC 89], [SHU 94a] trouve son origine dans l'étude psychologique du développement cognitif chez l'homme. Dans les expériences sur lesquelles sont basée cette étude, une balance sur laquelle différents poids sont disposés à différentes distances à gauche et à droite d'un pivot est présentée à des enfants. La tâche est de prédire vers quel côté la balance va pencher quand les appuis qui la maintiennent en équilibre sont retirés. Normalement, de chaque côté la totalité des poids sont placés sur une seule position (voir la Figure 63, où nous montrons une balance de 5 positions et 5 poids maximum).

Siegler a trouvé à propos de cette balance six sous-problèmes différents [SIE 76] :

1. Le problème de la balance *"équilibrée"* : la même quantité de poids est placée à égale distance de chaque côté ;

2. Le problème des *"poids"* : les distances étant égales, la balance penche du côté doté du poids le plus élevé ;

3. Le problème de la*"distance"* : les poids étant égaux, la balance penche du côté où la distance est la plus grande.

4. Le *"conflit-poids"* : on a plus de poids d'un côté, une distance plus grande de l'autre côté, et le côté qui penche est celui qui a le plus grand poids. Les deux autres problèmes, appelés *"conflits"*, ont aussi plus de poids d'un côté et une distance plus grande de l'autre côté;

5. Le *"conflit-distance"* : on a un *"conflit"*, et de plus le côté qui penche est celui qui a la plus grande distance.

6. Finalement, le *"conflit-équilibrée"* : on a un *"conflit"*, et de plus la balance reste équilibrée, c.-à-d. que la balance est en équilibre même si les poids et les distances sont différents de chaque côté.

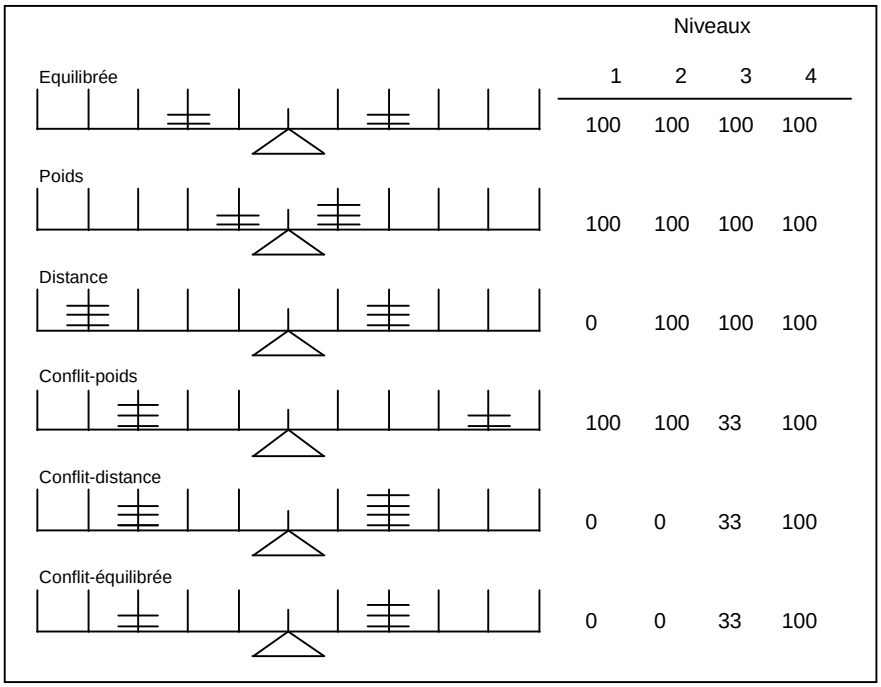


Figure 63 - Les sous-problèmes de la balance et les taux de réussite (en %) pour chacun des quatre niveaux.

Siegler a montré aussi que le développement cognitif chez les enfants se fait à travers 4 niveaux différents (voir Figure 63) basés sur les règles suivantes[2] :

• ***Niveau 1*** : l'enfant utilise seulement les poids pour savoir si la balance est équilibrée : si les poids sont égaux, alors la balance est en équilibre;

Si Pg = Pd, alors la balance est équilibrée ;
Si Pg > Pd, alors la balance penche à gauche ;
Si Pg < Pd, alors la balance penche à droite

[2] Pg = Poids à gauche ; Pd = Poids à droite ; Dg = Distance à gauche ; Dd = Distance à droite

• ***Niveau 2*** : l'enfant donne priorité au poids, mais il prend en considération la distance quand les poids des deux côtés sont égaux;

Si Pg > Pd, alors la balance penche à gauche ;
Si Pg < Pd, alors la balance penche à droite ;
Si Pg = Pd et Dg > Dd, alors la balance penche à gauche ;
Si Pg = Pd et Dg < Dd, alors la balance penche à droite ;
Si Pg = Pd et Dg = Dd, alors la balance est équilibrée.

• ***Niveau 3*** : l'enfant prend en considération poids et distance, mais il y a confusion quand un côté a un poids plus grand et l'autre côté une distance plus grande;

Si Pg = Pd et Dg = Dd, alors balance est équilibrée ;
Si Pg > Pd et Dg > Dd, alors la balance penche à gauche ;
Si Pg < Pd et Dg < Dd, alors la balance penche à droite ;
Si Pg > Pd et Dg < Dd, alors il y a confusion ;
Si Pg < Pd et Dg > Dd, alors il y a confusion ;

• ***Niveau 4*** : l'enfant multiplie distance par poids de chaque côté et compare les produits : si les produits sont égaux alors la balance est en équilibre, sinon, la balance penche du côté où la valeur du produit est la plus grande.

Si Pg*Dg = Pd*Dd, alors la balance est équilibrée ;
Si Pg*Dg > Pd*Dd, alors la balance penche à gauche ;
Si Pg*Dg < Pd*Dd alors la balance penche à droite.

Chacun de ces groupes de règles représente un des 4 niveaux cognitifs associés au problème. La Figure 63 montre les taux de prédictions associés à chacun des niveaux et par conséquence à chacun des ensembles de règles. Cette expérience, à l'origine conçue pour être utilisée dans l'étude du comportement humain, est devenue une source d'inspiration pour l'étude de la modélisation par ordinateur du développement cognitif humain. Les travaux connexionnistes le plus connus ont été développés par McClelland et par Shultz [MCC 89, SHU 94a].

Les études développées dans le domaine de l'apprentissage automatique ont utilisé des bases d'exemples obtenues généralement à partir d'une base complète de 625 cas (toutes les combinaisons possibles des entrées) décrits de la façon suivante :

Entrées : Pg, Dg, Pd, Dd (4 entrées qui peuvent assumer les valeurs 1,2,3,4 ou 5) où

Pg, poids à gauche ; Dg, distance à gauche ;

Pd, poids à droite ; Dd, distance à droite ;

Sorties : état de la balance (2 sorties qui peuvent assumer les valeurs -0.5, 0.0 ou +0.5)

état 1 = +0.5, -0.5, quand la balance penche à gauche ;

état 2 = 0.0, 0.0, quand la balance est équilibrée, et

état 3 = -0.5, +0.5, quand la balance penche à droite.

Par exemple, une balance qui a un poids égal à 2 unités à la position 3 du côté gauche et un poids égal à 4 unités à la position 2 du côté droite et qui par conséquence penche du côté droit, est représentée par : *2, 3, 4, 2, -0.5, +0.5*

Il faut souligner que les différentes études de modélisation de l'apprentissage du problème de la balance [SCH 96, SHU 94a, MCC 89] auxquels nous faisons référence ici, ont pour principal but la modélisation du développement cognitif, c'est-à-dire l'étude du passage d'un niveau à l'autre pendant l'apprentissage de la tâche (voir les 4 niveaux de Siegler dans la Figure 63).

5.3.2 Les Résultats Expérimentaux Obtenus avec INSS

Nous avons commencé notre étude sur le problème de la balance par une étude du type de règles utilisées dans la définition de chacun des 4 niveaux de Siegler. Nous avons constaté que les règles utilisées ne sont pas de simples règles d'ordre 0 du type utilisé par KBANN. Les règles du problème de la balance sont des règles qui mettent en relation deux entrées, avec des opérateurs de comparaison comme : plus grand, plus petit et égal. Nous avons appelé ce type de règles qui présentent une forte relation entre leurs entrées *règles de haut niveau* [REY 97, REY 97a]. Les règles de haut niveau peuvent présenter des 'opérateurs continus' (e.g. plus grand, plus petit) entre deux entrées, alors que les règles simples d'ordre 0 n'utilisent que des 'opérateurs binaires' entre

deux entrées (e.g. et, ou). Le développement des recherches afin d'introduire des règles d'ordre 0^+ (haut niveau) dans notre compilateur NeuComp (voir Section 4.2.1), a été fortement influencé par nos études et expériences sur le problème de la balance et aussi sur la robotique autonome.

Les résultats que nous avons obtenus dans nos tests préliminaires ont indiqué que les réseaux avaient des grosses difficultés pour apprendre parfaitement à résoudre le problème de la balance. Cela nous a amené à diriger nos recherches sur l'apprentissage parfait du problème, c'est-à-dire l'apprentissage du niveau 4 (100% de bonnes réponses dans les six sous-problèmes). Nous étions convaincus qu'il fallait tout d'abord comprendre d'où venait la difficulté de l'apprentissage parfait du niveau 4, avant de passer à d'autres études comme celui du passage d'un niveau à l'autre. Nos premières expériences ont portées sur la capacité d'un réseau à bien apprendre le problème complet. Nous avons voulu vérifier si les réseaux étaient bien capables de représenter des règles de haut niveau, comme celles du niveau 4.

Pour commencer, nous avons essayé de reproduire les expériences réalisées par Shultz avec le problème de la balance [SHU 94]. Shultz a utilisé une base d'exemples d'apprentissage initiale de 100 exemples (choisis parmi les 625 exemples) et il a ajouté un nouvel exemple dans la base à chaque époque d'apprentissage. L'ajout progressif d'exemples a été fait avec le seul but de garantir le passage progressif d'un niveau cognitif à l'autre. Après l'apprentissage, il a fait un test de généralisation avec une base de 24 exemples : ces 24 exemples sont choisis en prenant au hasard 4 exemples de chacun des 6 sous-problèmes différents. L'auteur signale que l'apprentissage se termine après 300 époques parce que le réseau a appris la 4ème règle, en ajoutant en moyenne 2 unités cachées. En résumé, il utilise 400 exemples d'apprentissage, 24 de test et il a besoin de 300 époques d'apprentissage.

Avant de continuer, il faut dire que pour les *règles du niveau 4*, l'apprentissage a été considéré comme acceptable par Shultz à condition d'obtenir *un minimum de 20 bonnes réponses* sur la base de test de 24 exemples ; alors que pour nous, l'intérêt est de trouver un réseau qui va apprendre parfaitement les règles du niveau 4 (100 % de bonnes réponses). C'est pourquoi nous parlerons ici de *"4ème niveau Shultz"* et *"4ème niveau parfait"*.

Pour répéter les expériences de Shultz, nous avons utilisé le système INSS avec l'algorithme d'apprentissage CasCor sans introduire de connaissances a priori. En ce qui concerne l'apprentissage, nous avons choisi au hasard 400 exemples dès le départ (la seule différence par rapport aux expériences de Shultz) et comme base de test 24 exemples, choisis exactement comme dans le travail de Shultz. Nous avons commencé par la réalisation de trois expériences. Premièrement, faire apprendre au réseau le *"4ème niveau parfait"* ; deuxièmement, arrêter l'apprentissage à 300 époques ; troisièmement, réaliser l'apprentissage avec toute la base d'exemples disponible sur le problème (625 exemples). Pour chaque expérience, nous avons répété l'apprentissage 5 fois en prenant des poids initiaux différents. Les résultats sont les suivants :

- Expérience 1 : pour apprendre le *4ème niveau parfait*, il a fallu attendre en moyenne 1544 époques et on a ajouté 14 unités cachées au réseau pendant l'apprentissage. Toutes les simulations ont requis plus de 1000 époques d'apprentissage et plus de 10 unités cachées ont toujours été ajoutées au réseau.

- Expérience 2 : quand le réseau est arrêté à 300 époques (*4ème niveau Shultz*), on obtient une moyenne de 90.84% de bonnes réponses sur les 24 exemples de test, usuellement sans aucune unité ajoutée. Le meilleur taux de généralisation a été obtenu à peu près à la 35ème époque. Dans aucune des simulations nous ne sommes arrivé à un taux de généralisation égal à 100%.

- Expérience 3 : les résultats n'ont pas beaucoup changé par rapport à l'expérience 1. En moyenne, nous avons ajouté 18 unités au bout de 1871 époques, pour pouvoir arriver à un taux de 100% de réponses correctes.

Etant donné ces résultats, nous avons conclut qu'il n'est pas possible d'arriver à l'apprentissage du 4ème niveau parfait sans ajouter plusieurs unités et seulement après un très grand nombre d'époques d'apprentissage. Cela montre bien que les réseaux de type PMC ont des problèmes pour apprendre cette tâche. Ces réseaux n'arrivent pas à bien représenter les règles de

niveau 4 étant donné le grand nombre d'unités cachées qu'ils sont obligés d'ajouter au réseau. Il semble que le réseau, capable de représenter les règles plus simples, ne parvient pas à bien apprendre les règles multiplicatives du niveau 4 (Pg*Dg comparé à Pd*Dd). Il est obligé de réaliser un apprentissage par coeur en ajoutant beaucoup d'unités. Nous avons constaté aussi que Shultz n'est probablement jamais arrivé au niveau 4 parfait.

Cette question sur la capacité des réseaux de représenter de façon interne les règles du niveau 4 est-elle si importante ? Nous pensons que oui. Nous nous sommes donc intéressés aux règles de haut niveau, à leur représentation interne dans les réseaux (compilation de règles), et à leur extraction à partir d'un réseau. L'étude sur la capacité de représenter les connaissances décrites par les règles du niveau 4 est très intéressante d'autant plus qu'on n'a jamais compris comment un arbre de décision (algorithme du type C4.5) peut représenter ce type de règles. Usuellement les arbres de décision ne prennent en considération qu'un seul attribut d'entrée à la fois, et pourtant on trouve des travaux sur l'apprentissage du problème de la balance qui s'utilisent de cette méthode [SCH 96]. Notre but n'était pas de vérifier si les arbres de décision peuvent ou non apprendre cette tâche, mais il fallait quand même vérifier si les réseaux que nous utilisons étaient capables d'apprendre les règles du niveau 4.

Dans la suite de nos expériences, nous avons essayé de mieux comprendre les connaissances acquises par les réseaux dans la tâche de la balance, afin de pouvoir proposer des alternatives pour améliorer l'apprentissage des règles du niveau 4. Une fois que nous n'avons pas encore des outils pour extraire des règles de ce type à partir des réseaux, nous avons décidé d'analyser plus en détails les réponses obtenues et de valider le comportement du réseau. Dans le cas de l'expérience 2, nous avons constaté qu'on arrive au 4ème niveau Shultz avec une moyenne de 2 ou 3 exemples (sur les 24) mal classés, ces exemples étant toujours du type "conflit". Avec l'expérience 3, nous avons constaté que le taux de bonnes réponses, après avoir augmenté très rapidement jusqu'à plus de 90% de bonnes réponses, se met ensuite à augmenter très lentement au fur et à mesure qu'on ajoute des nouvelles unités au réseau (voir Figure 64).

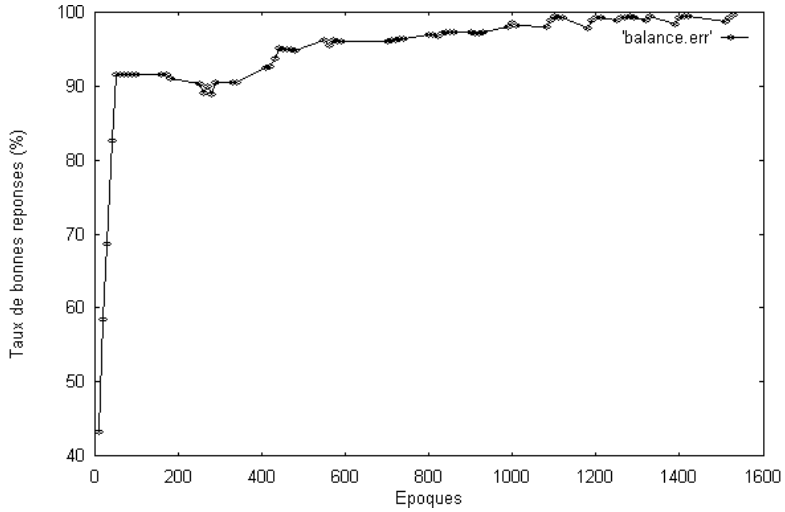


Figure 64 - Exemple de l'évolution du taux de bonnes réponses dans l'apprentissage du problème de la balance

Nous avons conclu que le réseau peut apprend sans problèmes des règles de niveau 1, 2, 3 ; mais il n'arrive pas au vrai niveau 4, car il n'est pas capable de représenter parfaitement la relation multiplicative existante entre les entrées. Nous pensons que le réseau arrive en réalité à un niveau 3.5, c'est-à-dire à un niveau intermédiaire entre les niveaux 3 et 4. Nous supposons que pour arriver à un taux de 100% de bonnes réponses, le réseau va arriver tout d'abord jusqu'au niveau 3.5, puis qu'il réalise un apprentissage par coeur des "cas problème". Des études réalisées avec la participation d'une autre personne de notre équipe, Benjamin Deuker, ont permis de trouver un nouvel ensemble de règles qui semblent représenter ce niveau 3.5. Les règles ont été trouvées par une analyse humaine, car le module d'extraction de règles n'est pas capable d'extraire ce type de règles à partir d'un réseau. Cependant, le système INSS a eu un rôle très important dans l'obtention de ce nouvel ensemble de règles, car il a permis l'analyse détaillée des réponses des réseaux. Les règles du niveau 3.5 sont les suivantes :

Si Pg = Pd et Dg = Dd, alors la balance est équilibrée ; Si Pg = Dd et Pd = Dg, alors la balance équilibrée ; Si Pg > Pd et Dg > Dd, alors la balance penche à gauche ; Si Pg > Dd et Dg > Pd, alors la balance penche à gauche ; Si Pg < Pd et Dg < Dd, alors la balance penche à droite ; Si Pg < Dd et Dg < Pd, alors la balance penche à droite.
* Cette base de règles peut être compilé dans un réseau avec 3 sorties binaires (0/1)

On peut remarquer que ces règles utilisent des comparaisons directes entre poids et distances, ce qui n'a pas de sens d'un point de vue de la physique. Toutefois, cet ensemble de règles permet d'obtenir un taux de bonnes réponses de 87.20% sur la base de 625 cas, et de 91.66% sur une base de 24 cas (seulement deux exemples sont mal classés). Ces résultats sont tout à fait compatibles avec les résultats obtenus par Shultz. Nous n'avons pas encore les moyens pour prouver avec certitude que les règles du niveau 3.5 sont réellement les règles que les réseaux apprennent à la place des règles du niveau 4, mais les expériences que nous avons réalisées nous amènent à considérer sérieusement cette possibilité. Des recherches en cours pourront nous apporter de nouvelles données sur ce sujet.

En ce qui concerne le problème de l'apprentissage des relations multiplicatives comme celles du niveau 4, nous avons conclu que les réseaux de type PMC n'ont pas une capacité de représentation interne de connaissances adéquate pour bien généraliser dans un cas pareil. D'autres expériences que nous avons faites ont montré qu'il est très difficile de réussir l'apprentissage d'une relation multiplicative entre deux entrées avec des réseaux de type PMC. Dans le cas du problème de la balance, cette difficulté d'apprentissage augmente avec le nombre de poids et des distances utilisées pour placer les poids (l'apprentissage par coeur devient de plus en plus nécessaire). Nous avons proposé pour ce problème une solution qui utilise un autre type de réseaux, les *réseaux Sigma-Pi* [RUM 86b, REY 97].

Type du Réseau	nombre d'exemples de apprentissage	nombre d'exemples de test	nombre maximum d'époques	Moyenne sur 5 simulations		
				Test de généralisation	Meilleure époque	CasCor Unités ajoutées
Shultz PMC	400	24	≤ 300	90.84%	35	0
Parfait PMC	400	24	sans limite	**100.00%**	**1544**	**14**
Shultz Sigma-Pi	400	24	≤ 300	**100.00%**	**157**	**0**
Shultz* PMC	400	625	≤ 300	92.16%	206	2
Parfait* PMC	400	625	sans limite	**100.00%**	**1500**	**13**
Parfait* Sigma-Pi	400	625	sans limite	**100.00%**	**396**	**1**
Complet PMC	625	625	sans limite	**100.00%**	**1871**	**18**
Complet Sigma-Pi	625	625	sans limite	**100.00%**	**91**	**0**

* Utilise tous les exemples disponibles dans la base de test de généralisation

Tableau 12 - Résultats des simulations du problème de la balance obtenus avec différents ensembles de données et différents types de réseaux

Nous avons utilisé le système INSS pour simuler ce type de réseaux, et les résultats obtenus sont encourageants. Le Tableau 12 montre les résultats de plusieurs expériences que nous avons faites avec la base de données du problème de la balance. On observe que les résultats des expériences avec la simulation des réseaux Sigma-Pi sont nettement supérieurs à ceux obtenus avec des réseaux de type PMC.

5.3.3 Discussion Finale sur le Problème de la Balance

Nous avons utilisé le système INSS pour étudier le problème de la balance présenté par Th. Shultz. Notre système a permis d'identifier des problèmes liés à l'apprentissage des règles qui composent le 4ème niveau du problème. Nous avons fini par nous concentrer spécifiquement sur cette étude de l'apprentissage des connaissances qui décrivent le 4ème niveau du problème de la balance. Notre intérêt pour cet aspect du problème est dû au fait que nous voulions mieux comprendre comment sont acquises et codées les connaissances de haut niveau (règles de haut niveau) dans les réseaux. En utilisant nos outils, nous avons pu constater que probablement Shultz [SHU 94] n'est jamais vraiment arrivé au 4ème niveau parfait. Avec l'aide de INSS et de l'interaction avec un expert humain, nous avons trouvé un ensemble de règles d'un niveau intermédiaire entre les niveaux 3 et 4 - le niveau 3.5. Les tests réalisés avec INSS montrent que cet ensemble de règles, plus simple que les règles du niveau 4, peut en réalité représenter le niveau final atteint par les autres chercheurs dans ses expériences sur l'apprentissage du problème de la balance.

Ensuite, nous avons étudié des solutions pour résoudre le problème de l'apprentissage des règles multiplicatives du niveau 4. Nous avons proposé une solution basée sur l'utilisation des réseaux de type Sigma-Pi. Nous avons pu tester l'apprentissage de ce type de réseau à travers des simulations réalisées en utilisant le système INSS. Les résultats obtenus montrent que ce type de réseau permet d'apprendre les règles du niveau 4 de façon plus rapide et avec une meilleure généralisation (utilise moins d'unités) que les réseaux de type PMC.

En conclusion, l'utilisation d'INSS pour traiter le problème de la balance a apporté deux contributions majeures : l'identification des problèmes liés à l'apprentissage des règles du 4ème niveau et l'aide à l'expert afin d'identifier les règles du niveau 3.5. De plus, nous avons proposé une alternative qui permet aux réseaux d'apprendre parfaitement les règles multiplicatives de haut niveau présentes dans le problème de la balance. Finalement, en ce qui concerne le problème de l'extraction de règles de haut niveau à partir des réseaux, on trouve dans les travaux réalisés par Reyes [REY 97] une proposition d'un système d'extraction qui pourra être capable d'extraire ce type de règles (basée sur des algorithmes génétiques comme ceux cités dans la Section 2.2.2.2).

5.4 ROBOTIQUE AUTONOME

Pour cette dernière application, nous avons choisi d'utiliser notre système dans une tâche de contrôle d'un robot mobile autonome. Des études avaient déjà été réalisées par des chercheurs de notre équipe dans le cadre d'un projet financé par la DRET [FES 97, FES 96]. Ces travaux portaient sur le contrôle d'un robot autonome, le robot Robuter, avec utilisation des réseaux de neurones.

La réalisation d'un système de contrôle pour la robotique autonome utilisant des réseaux de neurones est un sujet de recherche intéressant, car il permet d'aborder des questions telles que :

- Les problèmes liés au contrôle sensori-moteur;

- La réalisation de tâches de bas niveau (e.g. évitement d'obstacles) ainsi que de tâches de haut niveau (e.g. planification);

- Les interactions possibles entre un module connexionniste et un module symbolique;

- Les informations du niveau sensoriel sont représentées usuellement par des valeurs continues (signaux obtenus à partir des capteurs), ce qui pose des problèmes pour réaliser des traitements purement symboliques;

- La possibilité d'implémentation de plusieurs types de comportements différents, tels que : l'évitement d'obstacles, le suivi d'un mur, le suivi d'un couloir, le passage d'une porte, le déplacement vers une direction spécifiée;

- Les possibilités d'application d'approches modulaires (combinaison de multiples comportements) ou constructives (ajout incrémental de nouvelles compétences).

De plus, la robotique autonome est en plein développement actuellement, car les applications de ce type de système peuvent être multiples, à savoir : l'exploration d'environnements hostiles comme la planète Mars et les volcans, le pilotage automatique d'engins industriels, le pilotage automatique de voitures. Il faut bien reconnaître que les avances dans ce domaine ne nous permettent pas encore d'avoir des systèmes vraiment autonomes très performants (e.g. le robot *Sojourner* envoyé sur la planète Mars était en réalité télécommandé la plupart du temps). Ainsi, la robotique autonome reste-elle un sujet de recherche très intéressant et ouvert à de nouvelles études.

Nous avons choisi d'utiliser le robot Khepera [MON 93], le laboratoire Leibniz ayant à disposition un simulateur de ce robot, ainsi que le robot lui-même. Notre but était de tester INSS dans le cadre d'une nouvelle application réelle basée sur l'utilisation de valeurs d'entrée continues. Cette application est aussi parfaitement adaptée à l'étude et à la validation de notre système, car :

- Nous pouvons obtenir des exemples d'apprentissage et de test plus facilement, sans avoir des limitations comme c'était le cas pour la base de données des comas toxiques limitée, elle, à un certain nombre de cas connus;
- Nous pouvons générer de nouveaux exemples pour les situations qui posent des problèmes à l'apprentissage d'une tâche;
- Nous pouvons "jouer le rôle de l'expert" sur le sujet traité, car les connaissances sur des tâches comme l'évitement d'obstacles ne sont pas très complexes. Ainsi, nous pouvons facilement définir des bases de règles sur le problème;

• Nous avons besoin d'un système capable de traiter les valeurs sensorielles des capteurs (valeurs continues). Nous avons besoin d'un système capable de représenter les connaissances théoriques sur le problème à travers des règles d'ordre 0^+;

Nous allons présenter ici seulement les résultats dont nous pensons qu'ils sont les plus significatifs et qu'ils complétent le mieux les résultats déjà obtenus dans les autres applications.

5.4.1 Description de l'Environnement de Tests Utilisé

Nous avons utilisé le robot Khepera [MON 93] dans nos expériences en robotique autonome. Initialement, les expériences ont été développées avec un simulateur du robot, le *Khepera Simulator 2.0* développé par Olivier Michel à l'université de Nice Sophia-Antipolis [MIC 96]. Ce logiciel *'freeware'* peut être téléchargé à partir du WWW (Internet) à l'adresse : *http://wwwi3s.unice.fr/~om/khep-sim.html*.

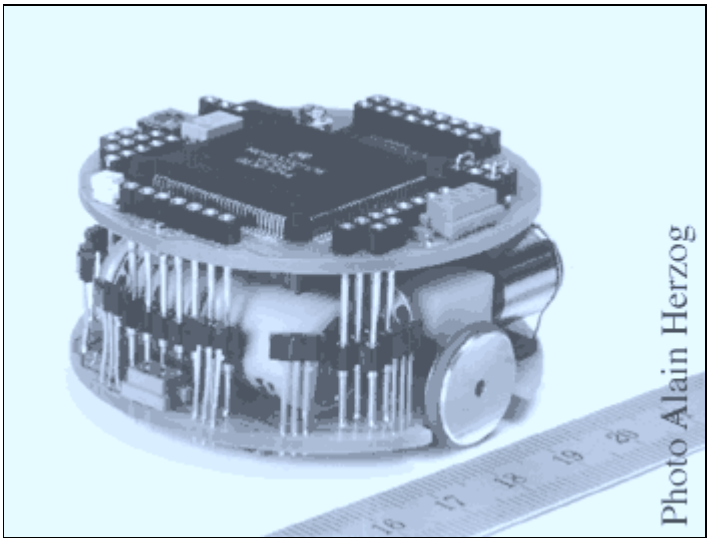


Figure 65 - Le robot Khepera

5.4.1.1 Le Robot Khepera

Khepera est un robot miniature de forme circulaire destiné à la recherche (voir Figure 65). D'un diamètre de 55mm, d'une hauteur de 30mm, Khepera pèse 86g. Il a été développé afin de tester les algorithmes de contrôle en robotique dans le monde réel. Il est constitué de deux roues et 8 capteurs infrarouges (IR). Ces derniers permettent la détection des obstacles (émission/réception IR) et des sources lumineuses (réception IR). Six capteurs sont placés à l'avant du robot et les deux autres à l'arrière (voir Figure 66). Pour la mesure de distance, les capteurs renvoient des valeurs numériques comprises entre 0 (lorsque le robot est à une distance supérieure à 5cm de l'obstacle) et 1023 (lorsque le robot est à une distance inférieure à 2cm de l'obstacle). Pour la mesure d'intensité lumineuse, l'intervalle est de 50 (intensité forte) à 520 (intensité nulle).

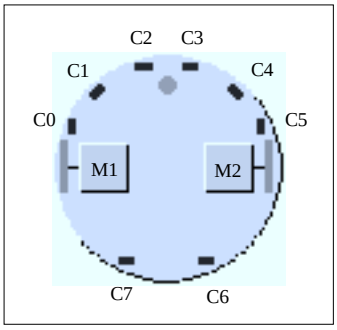


Figure 66 - Emplacement des capteurs sur Khepera

La commande des moteurs entraînant les roues est donnée par des valeurs comprises dans un intervalle entre -10 (tout vitesse vers l'arrière) et +10 (tout vitesse vers l'avant). Les deux moteurs sont indépendants, ce qui permet de faire des rotations par l'intermédiaire d'une commande utilisant deux valeurs opposées (e.g. +5,-5 = réaliser une rotation en vitesse moyenne dans le sens horaire). Khepera peut être utilisé sur un bureau, connecté à une station de travail par un câble de liaison série du type RS232. Cette liaison permet de lire les valeurs des capteurs et d'envoyer des commandes au robot en temps réel. On peut aussi télécharger un programme de contrôle dans sa mémoire (Khepera est doté d'un processeur Motorola 68331) par la liaison série, puis de le déconnecter du câble série afin de pouvoir rester complètement autonome.

5.4.1.2 Le Simulateur de Khepera

Le *Khepera Simulator 2.0* a été développé par Olivier Michel [MIC 96] à l'université de Nice Sophia-Antipolis sous l'environnement UNIX avec le langage C. Le simulateur est doté d'une interface graphique conviviale qui permet de créer des environnements personnalisés et d'y déplacer le robot très facilement. L'interface graphique est composée de deux parties : une partie correspondant à l'environnement simulé et une partie permettant à l'utilisateur de commander et d'observer le comportement du robot (contrôle des capteurs IR et des moteurs). La Figure 67 présente l'interface graphique du simulateur de Khepera.

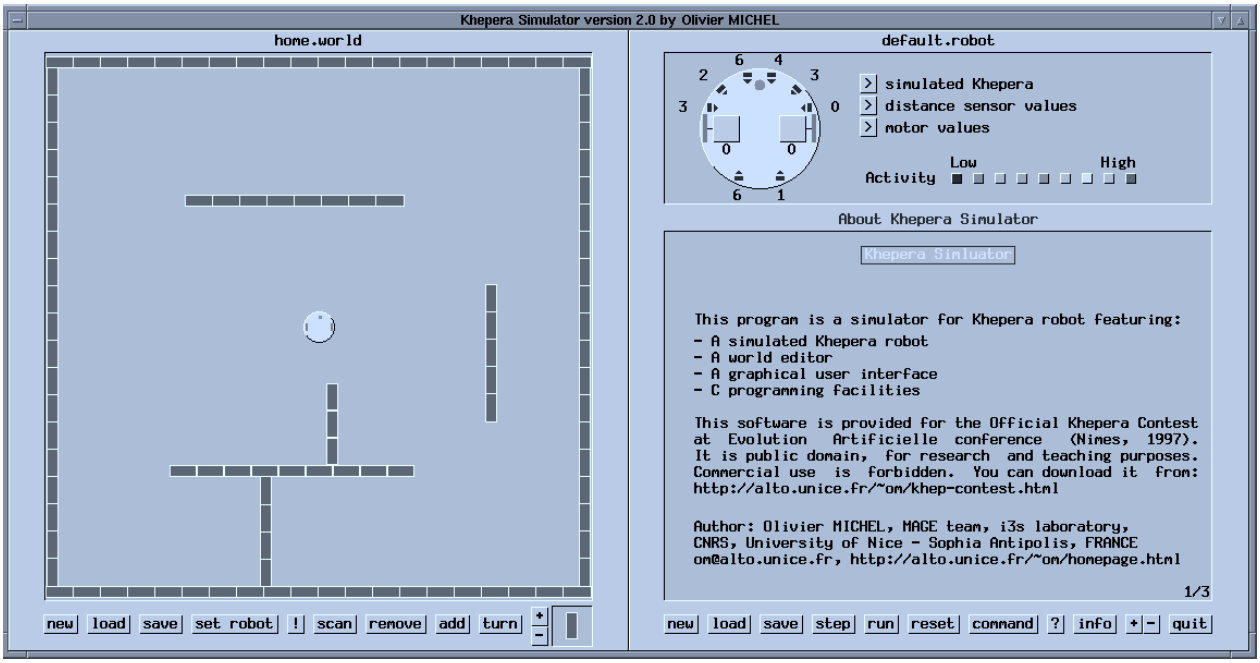


Figure 67 - Interface graphique du simulateur de Khepera

L'environnement simulé est équivalent à un espace plan de 1m sur 1m. Il est composé de trois types d'objets différents : des briques à 24 orientation possibles, des barreaux et des sources lumineuses. Ces éléments peuvent être placés à un endroit quelconque de l'environnement. Les briques permettent de créer des murs, et les barreaux représentent des obstacles ponctuels qui n'activent qu'un seul capteur à la fois. Quant aux sources de lumière, ces sont des obstacles

équivalents aux barreaux mais qui activent les capteurs de luminosité. De nombreux "mondes" (configurations de l'environnement) sont déjà définis dans le simulateur.

La commande du robot, l'affichage graphique de textes et de dessins sont entièrement paramètrables et programmables en langage C. C'est grâce à ces fonctions programmables que l'on peut développer son propre contrôleur du robot mobile Khepera. Toutes les informations sur l'état du robot sont aussi facilement accessibles par ces fonctions (e.g. valeurs des capteurs de distance, valeurs de la commande passée aux moteurs). Le manuel du simulateur possède une description détaillé des fonctions disponibles et de son mode d'emploi [MIC 96].

Le modèle choisi pour simuler les moteurs du robot est très simple. Le robot se déplace suivant la commande envoyée par l'utilisateur. Un bruit aléatoire de plus ou moins 10% est ajouté à la vitesse du moteur et un bruit de plus ou moins 5% est ajouté au calcul de la position résultante du robot obtenue en fonction du déplacement et de la direction indiquée par la commande précédente. Pour simuler un capteur, on considère un ensemble de 15 points disposés dans un triangle situé devant le capteur. La distance à l'obstacle est obtenue en fonction de la présence ou non d'obstacles en chacun des points considérés. A cela est rajouté un bruit de plus ou moins 10% de l'amplitude de la distance précédemment calculée.

Le modèle de simulation utilisé donne des résultats suffisamment proches de la réalité pour qu'on puisse ensuite passer au robot réel. Nous n'avons pas besoin de réaliser des changements significatifs sur les algorithmes de contrôle développés sur le simulateur afin de les utiliser sur le robot réel. De plus, nous pouvons utiliser la même interface du simulateur lorsque nous avons à choisir entre deux modes d'opération : mode de simulation ou mode de contrôle en temps réel du robot (les commandes et les valeurs des capteurs sont obtenues à partir de la liaison série).

Nous avons introduit quelques modifications sur le simulateur de Khepera afin de pouvoir mieux exploiter ses fonctionnalités :

1. Nous avons introduit la possibilité de commander le robot simulé par le clavier. L'utilisateur peut remplacer l'algorithme de contrôle et générer lui-même quelques commandes à envoyer au robot (e.g. avancer, tourner à gauche, tourner à droite).

2. Nous avons créé des fonctions qui permettent d'enregistrer sur des fichiers toutes les commandes envoyées au robot, ainsi que sont état à chaque pas (capteurs, position et orientation). De cette façon, nous pouvons créer des fichiers d'apprentissage supervisé (voir Figure 68) en réalisant le contrôle du robot par le clavier. L'utilisateur peut accompagner l'état des capteurs du robot, qui sont affichés à l'écran, et ensuite indiquer la commande la plus appropriée à réaliser.

3. Nous avons introduit une version simplifiée du simulateur NeuSim dans le simulateur de Khepera. Ainsi, nous avons la possibilité d'utiliser les réseaux créés par INSS pour contrôler le petit robot. Cette version simplifiée de NeuSim permet de charger un réseau et de l'activer, mais elle ne permet pas de réaliser l'apprentissage. Il faut utiliser la version complète de NeuSim pour pouvoir obtenir un réseau prêt à être employé avec le simulateur du robot.

4. Nous pouvons aussi enregistrer le comportement du robot quand il est contrôlé par le simulateur des réseaux de neurones. Cela nous permet se suivre précisément les réponses données par le système et, si l'on veut, de les valider postérieurement. De plus, nous pouvons utiliser ce fichier afin d'afficher la trajectoire complète parcourue par le robot pendant une expérience.

Fichier : robot.learn (base d'apprentissage - 8 entrées, 3 sorties)

```
8   3   1028   0
4   1    3   4    6    3   0   4        0 1 0
1   3    6   3    6    6   6   6        0 1 0
6   3   104  234  772  96  3   6        1 0 0
2   6   104  229  724  107 3   4        1 0 0
563 1023 57  6    6    0   3   1        0 0 1
544 1023 1   3    4    0   5   1        0 0 1
...
C0  C1   C2  C3   C4   C5  C6  C7       G A D
```

C0 - C7 : Capteurs 0 à 5 (devant) + 6 et 7 (arrière)
G: Tourner à gauche
A: Avancer
L: Tourner à droite

Figure 68 - Exemple d'un fichier d'apprentissage utilisé par NeuSim

Les expériences que nous avons développées avec le simulateur de Khepera, ainsi que les modifications apportées au simulateur, ont suscité d'autres travaux postérieurs développés au sein de notre équipe [PLE 97, MEK 97, PAR 96].

5.4.2 Les Résultats Expérimentaux Obtenus avec INSS

Nous avons commencé nos expériences par l'apprentissage d'un comportement d'évitement d'obstacles. Cette expérience a consisté à créer un fichier d'apprentissage contenant des situations caractéristiques d'évitement d'obstacle, indiquées dans la Figure 69. Nous avons créé un fichier contenant une base d'apprentissage supervisé (robot télécommandé) et ensuite nous avons fait apprendre au réseau cette base. Le comportement que nous avons voulu apprendre consistait à s'éloigner des obstacles (murs) à chaque fois que le robot les rencontrait, c'est-à-dire à tourner vers un côté jusqu'à ne plus avoir un obstacle en face.

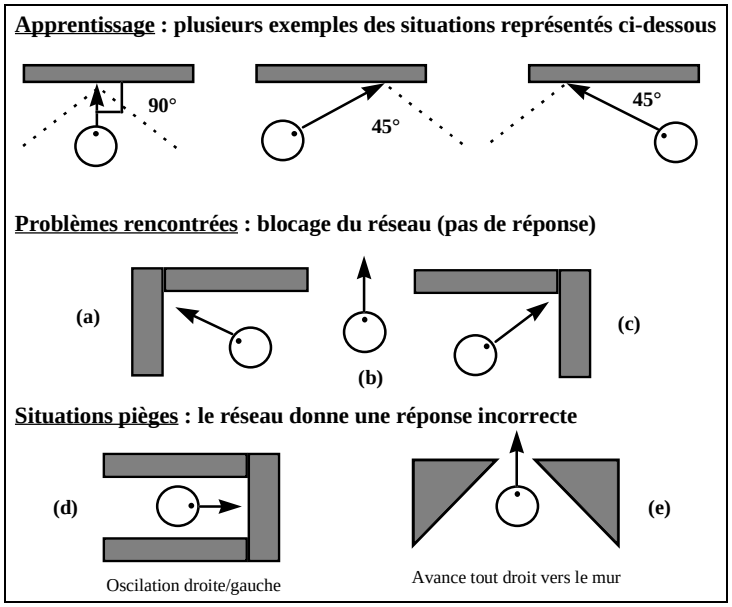
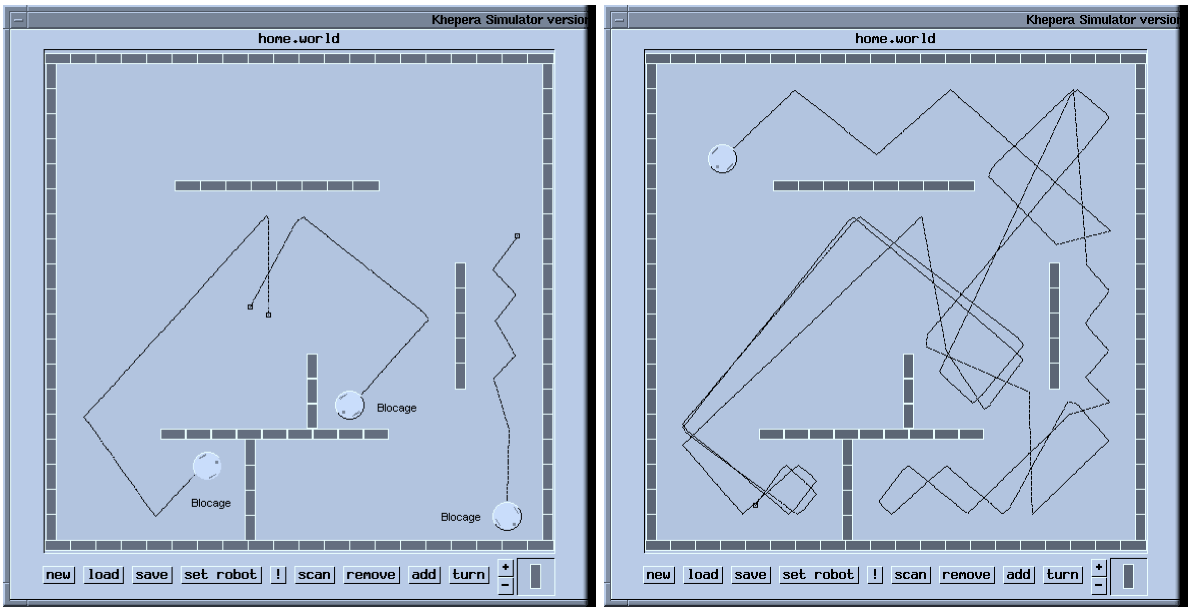


Figure 69 - Situations typiques d'évitement d'obstacle

Dans nos expériences, nous avons obtenu un taux proche de 95% de bonnes réponses sur la base d'apprentissage utilisée. Nous avons constaté que le réseau a bien appris à éviter les murs qui

se présentaient en face du robot (comportement basé sur les 3 types de cas d'apprentissage - Figure 69). Nous avons obtenu une bonne généralisation pour des situations différentes de celles apprises, où le contact avec le mur s'est produit avec un angle différent de ceux utilisés dans la base d'apprentissage.



(a) Apprentissage de situations simples *(b) Après l'ajout des nouveaux exemples*

Figure 70 - Evitement d'obstacles : trajectoire du robot après l'apprentissage

Cependant, le robot qui a fait preuve d'un comportement parfait en présence d'un mur unique et sans coins, n'avait pas un bon comportement en face de situations comme celles que nous décrivons dans la Figure 69 (problèmes et pièges). Dans des situation de type 'a', 'b' ou 'c' le réseau donne une réponse avec toutes les sorties proches de zéro, c'est-à-dire il ne prend aucune décision et s'arrête. Cela se produit car le réseau est face à une situation (configuration des capteurs) complètement inconnue. Nous avons ajouté des exemples des situations 'a' et 'c' dans le fichier d'apprentissage et le réseau ainsi obtenu n'a plus présenté de problèmes par rapport à ce type de cas (voir Figure 70). Par contre, nous avons aussi ajouté des exemples de cas de la situation de type 'b' (passage d'une porte) qui n'ont pas beaucoup contribué à améliorer le comportement du robot contrôlé par notre réseau. Avec ces nouveaux exemples appris, un nouveau problème est apparu : le robot, qui est maintenant devenu capable de passer par une configuration de type 'b', va tout droit dans une situation de type 'e', jusqu'au point de heurter le

mur. Nos essais ont montré que les capteurs du robot n'ont pas une précision suffisante pour distinguer entre une situation 'b' ou 'e' (capteurs latéraux saturés et capteurs frontaux avec une faible activation).

Un dernier problème constaté lors de nos essais sur l'évitement d'obstacles, est le cas du blocage du robot dans une situation de type 'd' (couloir sans issue). Le robot rencontre le fond du couloir. Il commence à tourner vers l'un des deux côtés. Il rencontre un mur et commence à tourner vers l'autre côté où il rencontre de nouveau un mur. Il se met alors à osciller entre un côté et l'autre indéfiniment. Ce type de blocage peut aussi arriver dans certains cas bien précis de situations de type 'a' ou 'c'. Nous avons essayé d'ajouter des exemples sur les situations de blocage, mais cela n'a pas marché. L'utilisation de notre système a permis d'identifier la cause exacte du problème : le robot se met à osciller entre tourner à gauche et tourner à droite, parce que chaque commande provoque un changement de l'état des capteurs qui amène à la commande inverse dans le pas de temps suivant.

Nous avons conclu qu'il faut prendre en compte le temps (contexte de la situation) afin de pouvoir sortir de cette situation de blocage. Une fois que l'on a commencé à tourner à droite, il faut poursuivre cette opération jusqu'à ce que le chemin soit libre. Afin de résoudre ce problème, nous avons envisagé d'utiliser une fenêtre temporelle : utiliser l'état des capteurs dans les pas de temps antérieurs afin de fournir un contexte au réseau, et ainsi pouvoir prendre une décision plus juste.

Nous avons employé ce type d'approche dans une autre expérience qui nécessite aussi une information contextuelle (le suivi de contour de murs). Toutefois, pour éviter de tout recommencer à partir de zéro, nous avons adopté une solution plus simple : nous faisons un post-traitement des réponses du réseau. Si l'on trouve des séquences d'alternance entre les commandes tourner à gauche et tourner à droite, nous faisons ensuite un grand tour vers l'un des côtés.

Ce premier groupe d'expériences nous a permis de constater encore une fois l'importance d'avoir la capacité de faire évoluer les connaissances acquises sur le problème en question. L'utilisation de notre système a beaucoup aidé dans le traitement de cette tâche. Nous avons pu

raffiner les connaissances du réseau en ajoutant de nouveaux exemples. Le système nous a aidés à trouver les 'points faibles' du comportement appris par le réseau et les problèmes de codage des informations d'entrée (nécessité d'informations temporelles). Le choix des exemples à ajouter dans la base d'apprentissage a été possible grâce à l'identification précise des problèmes rencontrés dans le comportement du robot. Nous n'avons pas fait de tests sur toutes les possibilités de problèmes que nous pourrons rencontrer dans une application comme celle-ci, et nous sommes certains qu'il existe encore d'autres problèmes différents de ceux que nous avons présentés ici. L'important pour nous n'est pas d'arriver à un système parfait (on peut se poser la question de savoir si un tel système existe bien) mais d'être capables de faire évoluer constamment les connaissances sur le problème étudié.

Notre deuxième série d'expériences a été composée par des tests sur la création et l'insertion de règles d'ordre 0^+ dans les réseaux. Avec l'expérience acquise dans notre première expérience, nous avons créé une base de règles symboliques qui puisse coder les connaissances sur le problème d'évitement d'obstacles. Nous avons obtenu l'ensemble de règles suivant :

```
#define DETECTWALL 900
#define SENSIB     100

% VERY SIMPLE Autonomous Robot Controller - by Osorio, Lab. Leibniz

$Features:
SSL : range : [0.0 , 1024.0];  % Sensor 0             2   3
SL  : range : [0.0 , 1024.0];  % Sensor 1       1    SFL SFR    4
SFL : range : [0.0 , 1024.0];  % Sensor 2         SL          SR
SFR : range : [0.0 , 1024.0];  % Sensor 3     SSL                SSR
SR  : range : [0.0 , 1024.0];  % Sensor 4      0                  5
SSR : range : [0.0 , 1024.0];  % Sensor 5
SBR : range : [0.0 , 1024.0];  % Sensor 6            SBL SBR
SBL : range : [0.0 , 1024.0].  % Sensor 7             7   6
$End_Feat.

$Rules:

right <= GT (SSL, DETECTWALL, SENSIB);
right <= GT (SL,  DETECTWALL, SENSIB);
right <= GT (SFL, DETECTWALL, SENSIB);

left  <= Not(right), GT (SFR, DETECTWALL, SENSIB);
left  <= Not(right), GT (SR,  DETECTWALL, SENSIB);
left  <= Not(right), GT (SSR, DETECTWALL, SENSIB);

forward <=  Not(right), Not(left).

$End_rules.

$End.
```

Figure 71 - Ensemble de règles symboliques pour l'évitement d'obstacles

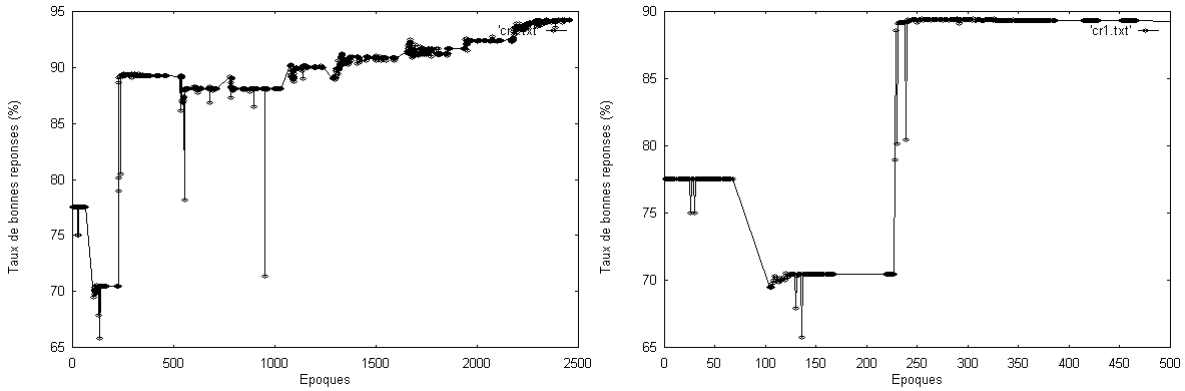
Cette base de règles a été compilée sous la forme d'un réseau. Ensuite, nous avons testé ce réseau dans le simulateur du Khepera. Le comportement du robot obtenu avec cette base de règles est un peu différent de celui obtenu par l'apprentissage de la base d'exemples utilisée dans notre première série d'expériences. Le robot arrive à éviter les murs parfaitement, mais au lieu de s'éloigner toujours des murs, il tourne parfois de façon à pouvoir tout juste éviter les murs en continuant d'avancer. La Figure 73(a) montre des exemples du comportement du robot obtenu en utilisant la base de règles qui se trouve ci-dessus. Cette figure montre que, comme nous l'avons dit, le robot évite de trop s'approcher des murs, mais sans forcement s'en éloigner.

Il faut cependant remarquer que la façon dont nous avons avons conçu la base de règles symboliques a fini par introduire un biais dans le comportement du robot. Le robot va avoir en effet une tendance à tourner plutôt vers la droite. Ce type de comportement finit parfois par restreindre la capacité d'exploration de l'environnement du robot, puisqu'il il va toujours dans la même direction.

Enfin nous avons essayé d'améliorer les connaissances de départ du robot codées par les règles, à l'aide de l'apprentissage de la base d'exemples que nous avons utilisé précédemment avec succès (compilation de règles suivi de l'apprentissage). Notre première réaction a été de considérer cette expérience comme un échec complet. Tout de suite après le démarrage du processus d'apprentissage, nous nous sommes aperçus qu'au lieu d'améliorer la qualité des réponses du réseau, le processus d'apprentissage provoquait une décroissance de la quantité de bonnes réponses obtenues (voir la Figure 72 - les 200 premières époques).

Nous avons arrêté la simulation, et nous sommes d'abord demandé si le problème ne résultait pas d'une mauvaise implémentation de nos algorithmes d'apprentissage. Nous avons refait la simulation de façon à mieux comprendre ce qui s'était passé. Dans cette deuxième simulation, nous avons laissé le processus d'apprentissage poursuivre son exécution pendant un plus grand nombre d'époques. Nous avons été surpris par les résultats obtenus, et surtout par la façon dont s'est développé l'apprentissage. Le réseau a finalement réussi à apprendre la base avec un taux proche de 95% de bonnes réponses. Nous avons répété cette expérience plusieurs fois et

le résultat a toujours été le même : une chute dans le taux de réussite dans les premières époques (autour de 300 époques), suivi d'une augmentation de ce taux jusqu'aux alentours de 95% de réussites. La Figure 72 montre un exemple typique de l'évolution du processus d'apprentissage dans une des expériences que nous avons réalisées.



(a) Apprentissage complet *(b) Vue en détail des premières 500 époques*

Figure 72 - Evolution du taux de réussite pendant l'apprentissage (règles + exemples)

Nous avons fait l'hypothèse que ce comportement tout à fait singulier du réseau est dû à l'introduction de règles symboliques avant la réalisation du processus d'apprentissage. Cette hypothèse a été renforcée par les faits suivants :

1. Les règles symboliques ont donné au robot un comportement (voir Figure 73(a)) différent de celui obtenu par l'apprentissage uniquement des exemples (voir Figure 70). Les connaissances théoriques divergent des connaissances pratiques disponibles sur cette application.

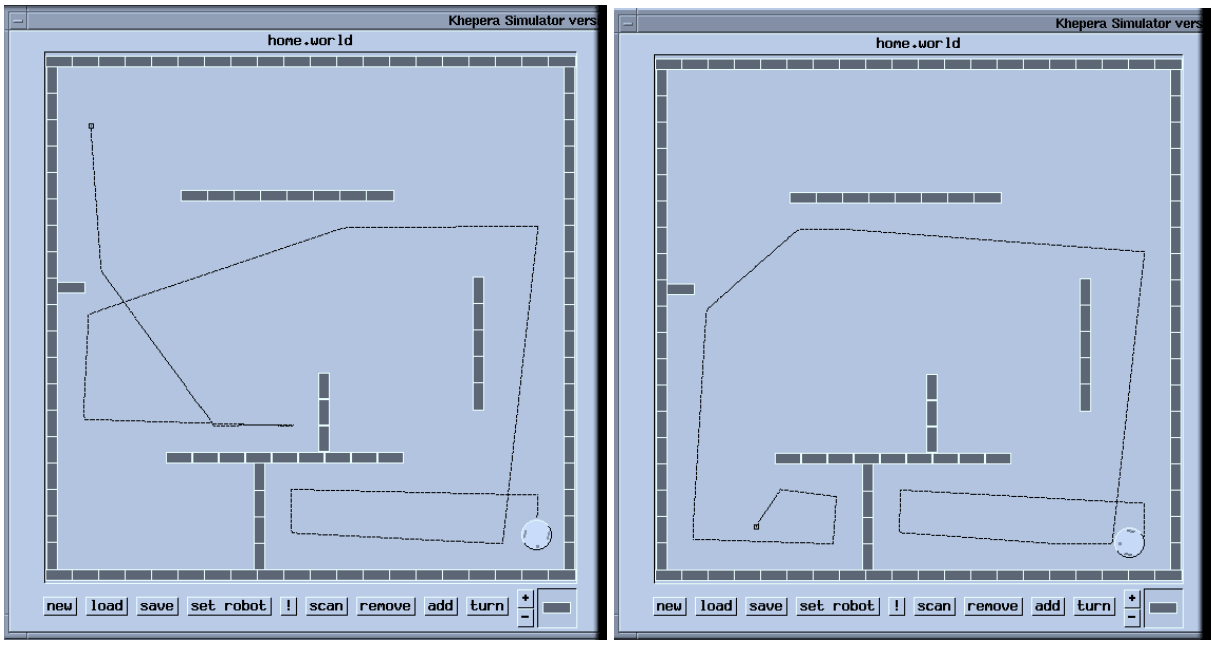
2. Le réseau avait besoin d'effacer (ou modifier fortement) ses connaissances de départ pour pouvoir arriver à bien apprendre la base d'exemples. Cela peut expliquer la décroissance initiale du taux de bonnes réponses, suivi d'une amélioration.

3. Le taux final de bonnes réponses (95%) est pratiquement le même que celui obtenu par l'apprentissage sans l'introduction des règles. Les règles n'ont pas aidé à améliorer la performance du réseau. De plus, il nous semble que les règles ont fini par être totalement remplacées par les nouvelles connaissances acquises.

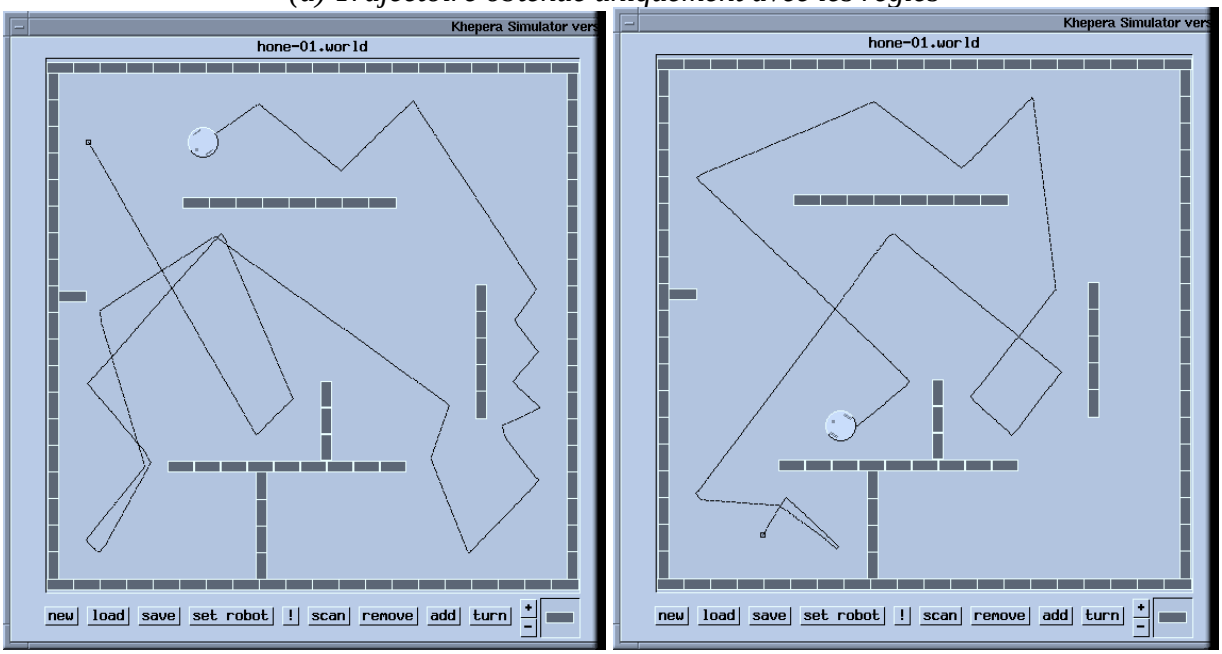
4. Le type de comportement initial du robot avant l'apprentissage (il utilise alors seulement les règles - voir Figure 73(a)) est différent de son comportement après l'apprentissage (voir Figure 73(b)). Le comportement initial est d'éviter les murs, sans avoir besoin de s'en éloigner. Le comportement final obtenu après l'apprentissage (règles + exemples) est très proche, sinon identique, de celui obtenu dans notre première expérience basée uniquement sur des exemples (voir la Figure 70 et Figure 73(b)).

Nous pensons donc que le comportement anormal du réseau est dû au fait que nous avons introduit des règles que ne sont pas en accord avec le type de comportement reproduit dans la base d'apprentissage. (Ce comportement du réseau peut être mis en relation avec le comportement du réseau étudié par D.E. Rumelhart et J.L. McLelland pour l'apprentissage du prétérit des verbes anglais ; voir le chapitre 18 du "PDP" [RUM 86a]). Nous avons réussi à apprendre les exemples, grâce à la capacité de nos réseaux d'ajouter de nouvelles connaissances (et de nouvelles unités), car un réseau statique ne serait pas capable de changer à ce point ses connaissances de départ. Cette expérience a prouvé que nos réseaux sont capables de manipuler des connaissances d'origines différentes (théoriques et pratiques), et ils fonctionnent bien même si les connaissances fournies sur le problème sont très divergentes.

Dernier point : le fait d'avoir réussi à imposer le comportement décrit par les exemples ne veut pas dire que l'utilisateur va être obligé de le considérer comme la meilleure solution. L'un des points forts du système INSS est cette possibilité de mettre face à face les différentes connaissances disponibles sur le problème, et ainsi de pouvoir mieux les intégrer, les exploiter et les valider.



(a) Trajectoire obtenue uniquement avec les règles



(b) Trajectoire obtenue après l'apprentissage (règles + exemples)

Figure 73 - Evitement d'obstacles : utilisation d'une base de règles

Le dernier groupe d'expériences que nous avons réalisées est composé par des tests sur l'apprentissage d'un nouveau type de comportement : le suivi de contour d'un mur. Ce type de tâche est plus complexe qu'un simple évitement d'obstacles, car le contexte de la situation dans

laquelle nous nous trouvons joue un rôle essentiel. De plus, nous avons choisi de tester cette application sur le robot réel afin de valider les expériences réalisées au niveau du simulateur.

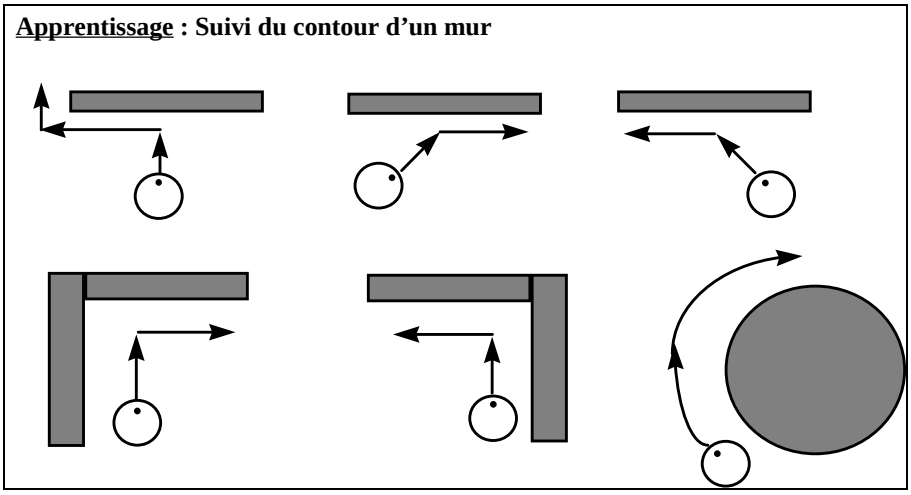


Figure 74 - Apprentissage d'un comportement de suivi du contour d'un mur

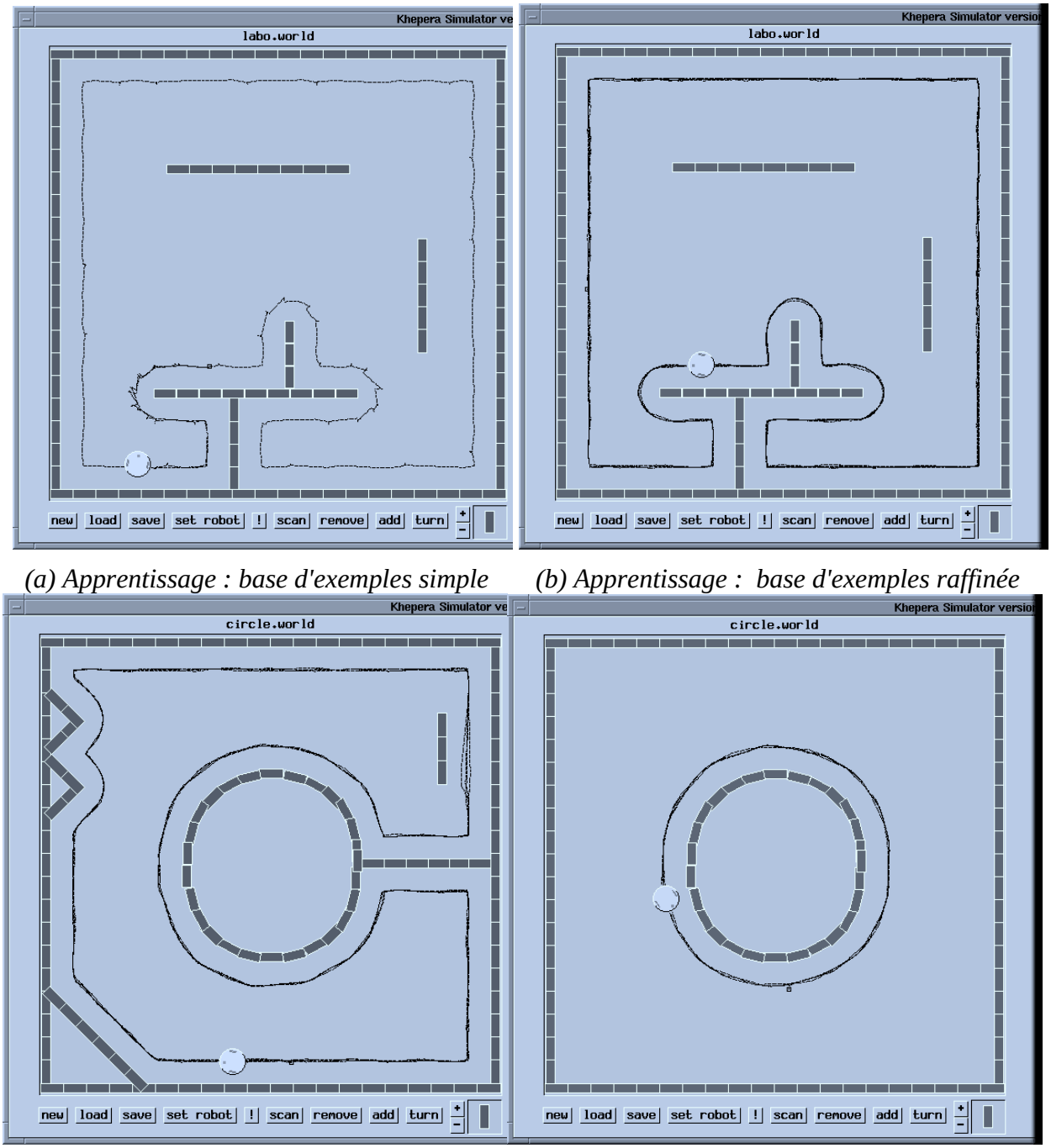
Nous avons commencé cette étude par l'apprentissage d'une base d'exemples. Cette base était composée de situations simples, comme celles représentées dans la Figure 74, sauf que nous avons obligé le robot à toujours tourner dans un seul sens, de façon à garder le mur toujours sur sa gauche (ceci évite les conflits et les exemples contradictoires). Chaque exemple de la base comporte les valeurs des capteurs dans le temps *'t'* (pas de temps présent) et dans le temps *'t-1'* (pas de temps précédent), plus la valeur de la sortie désirée (tourner à gauche, avancer ou tourner à droite). L'utilisation de l'état précédent des capteurs permet d'avoir une information contextuelle et ainsi d'éviter de décrocher du mur. Nos expériences pratiques ont montré que l'utilisation d'un seul pas de temps précédent était suffisante pour éviter le décrochage du mur.

Pour réaliser l'apprentissage du réseau, nous avons procédé de la façon suivante :

- Nous avons commencé l'apprentissage avec une base d'exemples simple, suivi d'un test du comportement du robot obtenu sur le simulateur;

- Ensuite, pour chaque situation de blocage (aucune réponse du réseau) ou de décrochage du mur, nous avons identifié les cas problèmes et nous avons ajouté des exemples complémentaires dans la base d'apprentissage afin de mieux apprendre à traiter ces cas;

- Nous avons répété ce processus jusqu'à ne plus avoir de problèmes de blocage ou de décrochage sur certains environnements de test préalablement choisis.



(a) Apprentissage : base d'exemples simple *(b) Apprentissage : base d'exemples raffinée*

(c) Trajectoire réalisée sans blocage, ni décrochage *(d) Trajectoire réalisée sans blocage, ni décrochage*

Figure 75 - Trajectoires du robot dans les environnements de test (suivi du contour)

La Figure 75 montre les trajectoires du robot dans les environnements de test du suivi de contour de murs. Dans la figure (a), on peut remarquer le comportement irrégulier du robot dans une phase initiale du processus d'apprentissage. Dans les figures (b), (c) et (d), on peut voir le comportement du robot après la fin du processus d'apprentissage (base d'exemples raffinée plusieurs fois). Nos tests ont montré qu'apparemment le robot simulé n'avait plus de problèmes de blocage ou de décrochage. L'étape suivante a été la réalisation d'un test de suivi de contour d'un mur avec le robot réel, en utilisant le même réseau testé sur le simulateur.

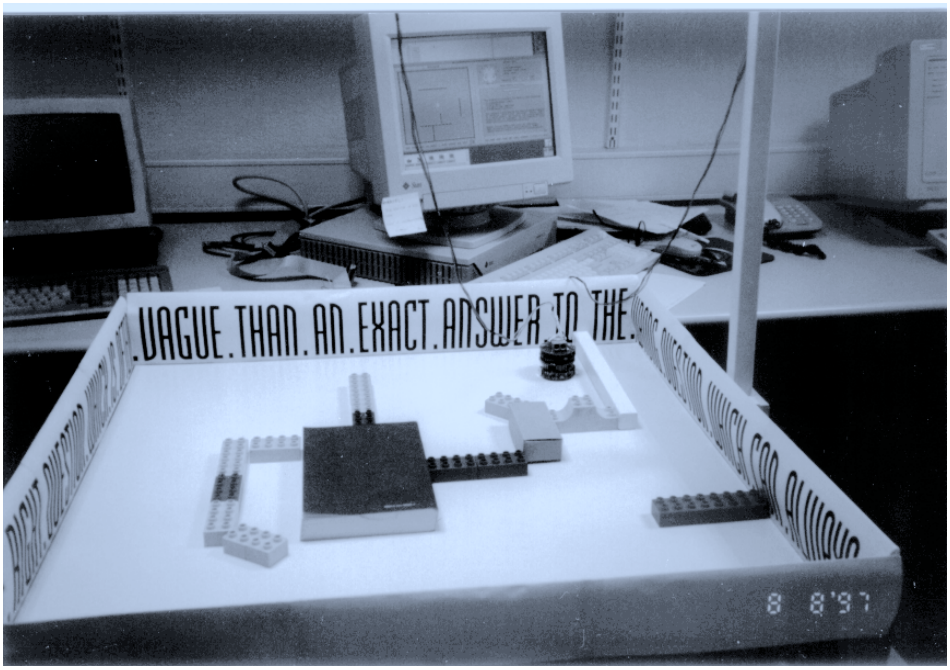


Figure 76 - Environnement de test avec le robot réel

Nous avons construit un environnement de test pour le robot, présenté dans la Figure 76. Comme le simulateur permet de commander le robot en utilisant la liaison série, nous n'avons rien eu à changer sur notre implémentation. La seule opération nécessaire à été de reconfigurer le simulateur de façon à passer du mode de contrôle du robot simulé au mode de contrôle du robot réel. La Figure 77 présente la trajectoire (elle est dessinée sur la photo) du robot réel obtenue en utilisant notre réseau pour la génération des commandes de contrôle. Nous avons répété cette

expérience plusieurs fois. Dans aucune de ces expériences le robot n'a décroché, ni touché aux murs, ni s'est bloqué. Le comportement obtenu avec le robot réel a été parfait.

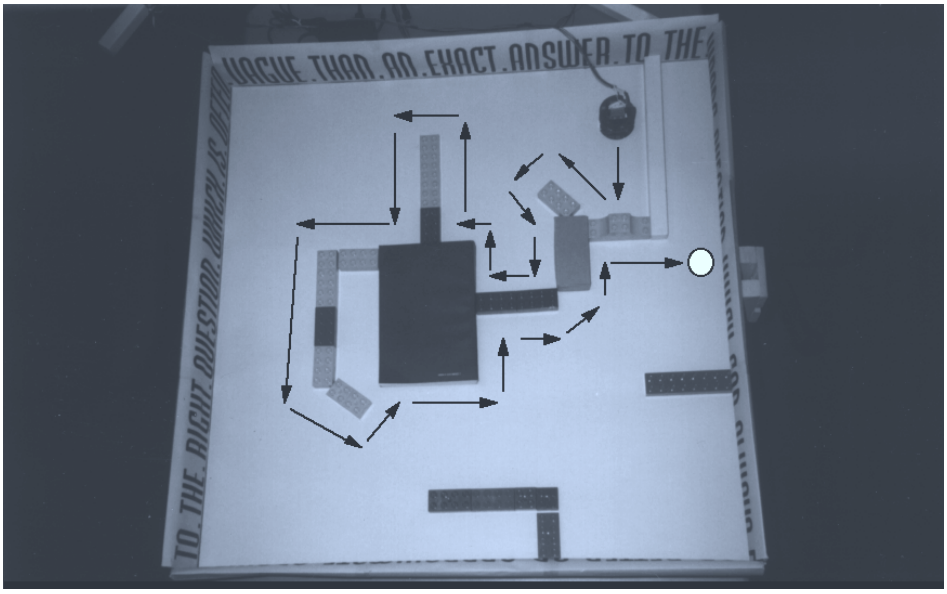


Figure 77 - Trajectoire du robot dans une tâche de suivi de contour des murs

Cette dernière expérience nous a permis de montrer que toutes les expériences réalisées au niveau du simulateur ne sont pas très différentes de ce qu'on peut faire avec le vrai robot. Le simulateur de Khepera implémente un modèle assez proche du comportement du vrai robot. Il faut reconnaître que cette facilité de passage du simulateur au robot réel est propre au Khepera et au type d'expérience que nous avons réalisées.

De plus, les réseaux que nous avons obtenus sont très robustes et généralisent bien dans des situations inconnues. Ainsi, notre système nous a-t-il permis de réussir l'implémentation du comportement de suivi de contour de murs d'une façon simple et rapide

5.4.3 Discussion Finale sur l'Application en Robotique Autonome

L'utilisation de notre système dans une application en robotique autonome nous a permis : de tester l'insertion de règles d'ordre 0^+, d'appliquer les techniques de raffinement de connaissances, et de montrer encore une fois que notre système peut être tout aussi performant dans des applications académiques que dans des applications réelles. Ce type d'application reste l'un des domaines de test des systèmes hybrides le plus intéressant que nous ayons eu l'occasion d'étudier. La possibilité de définir des bases de règles pour le contrôle du robot, ainsi que le fait de pouvoir générer des exemples (en quantité presque illimité), nous a permis d'approfondir nos études pratiques sur notre système.

Le système INSS a également été utilisé dans l'étude d'une autre application en robotique autonome à l'IFREMER. Il s'agit de mettre au point un système hybride de détection de pannes d'un robot sous-marin (Vortex). Des études préliminaires ont été développées par Christophe Perez dans le cadre d'un stage de D.E.S.S. en Informatique et Intelligence Artificielle [PER 96a, PER 96b] sous la supervision de M. Perrier. Ces travaux sont à l'origine des travaux menés actuellement par Benjamin Deuker dans le cadre de sa thèse, et aussi, d'un nouveau projet de coopération entre le laboratoire Leibniz et l'IFREMER (en cours d'élaboration).

Nous pensons que les études que nous avons développées jusqu'à présent sur les applications en robotique autonome doivent être poursuivies afin d'une part de mieux connaître toutes les potentialités et limites de notre système, d'autre part de confronter directement les résultats que nous avons obtenus à des résultats obtenus par d'autres chercheurs en robotique. Les systèmes de robotique autonome restent difficiles à évaluer avec précision. Nous n'avons pas les moyens d'obtenir une mesure absolue des performance du système, par opposition à certaines autres applications où cela est possible (e.g. diagnostic médical).

5.5 CONSIDERATIONS FINALES

Dans ce chapitre, nous avons présenté l'utilisation de notre système dans des applications académiques et dans des applications réelles. Les résultats obtenus montrent d'une façon pratique les divers points forts du système INSS :

- Le système a la capacité d'intégrer des connaissances théoriques (règles) et des connaissances empiriques (exemples) avec l'utilisation d'algorithmes très performantes. (Application utilisée : les problèmes du Moine);

- Le système a la capacité d'extraire seulement les nouvelles connaissances acquises par le réseau, sans avoir besoin d'extraire toutes les connaissances qui ont été introduites par compilation de règles. (Application utilisée : les problèmes du Moine);

- Le système est capable d'acquérir, d'extraire et de valider les connaissances disponibles sur un problème réel très complexe, avec une très bonne performance. (Application utilisée : aide au diagnostic des comas toxiques);

- Le système est capable de coder des règles de haut niveau (règles d'ordre 0+) dans les réseaux. Il peut traiter parfaitement les valeurs d'entrée discrètes ainsi que les valeurs continues. De plus, son module de validation permet d'identifier et de traiter des problèmes liés à l'étude des tâches sur lesquelles il fonctionne. (Applications utilisées : le problème de la balance et le contrôle en robotique autonome);

Nous pensons avoir réussi à montrer les différentes capacités du système INSS à travers ces expériences. Nous savons qu'il existe encore beaucoup d'études à réaliser sur notre système. Cependant, les expériences que nous avons réalisées montrent que ce nouveau système est très performant et qu'il peut être facilement appliqué à différents problèmes.

6. CONCLUSIONS ET PERSPECTIVES

Dans ce mémoire, nous avons présenté nos recherches sur les système hybrides neuro-symboliques (SHNS), et en particulier sur l'acquisition incrémentale de connaissances à partir de connaissances théoriques (règles symboliques) et de connaissances empiriques (exemples). Nous avons décrit un nouveau système hybride, nommé système INSS - *Incremental Neuro-Symbolic System*, que nous avons étudié et réalisé. Ce système permet le transfert de connaissances déclaratives (règles symboliques) d'un module symbolique vers un module connexionniste (réseau de neurones artificiel - RNA) à travers un convertisseur de règles en réseaux. Les connaissances du réseau ainsi obtenu sont affinées par un processus d'apprentissage à partir d'exemples. Ce raffinement se fait soit par ajout de nouvelles connaissances, soit par correction des inconsistances, grâce à l'utilisation d'un réseau constructif.

Nous avons aussi présenté notre méthode d'extraction incrémentale de règles qui a été intégrée au système INSS, ainsi que les algorithmes de validation des connaissances qui ont permis de mieux coupler les modules connexionniste et symbolique. Ainsi, le système que nous avons développé est-il caractérisé comme un outil d'apprentissage automatique qui possède la capacité de réaliser une acquisition constructive de connaissances.

Notre système a été testé sur plusieurs applications, en utilisant des problèmes académiques et des problèmes réels (diagnostic médical et contrôle d'un robot autonome). Les résultats obtenus montrent que le système INSS à des propriétés intéressantes et de nombreux avantages par rapport aux autres systèmes hybrides du même type :

- Il permet l'insertion et la représentation de connaissances de haut niveau (règles de haut niveau) dans les réseaux connexionnistes;

- Il est incrémental à plusieurs niveaux : les réseaux connexionnistes sont de type constructif, l'extraction de règles peut être réalisée de manière incrémentale, et les connaissances

peuvent évoluer à travers un processus cyclique, centré sur les réseaux neuronaux, "d'insertion-raffinement-extraction-validation";

- Il offre la possibilité de travailler avec des données qualitatives aussi bien qu'avec des données quantitatives;

- Il est capable de mieux exploiter les connaissances sur un problème donné, car il peut intégrer les deux types de connaissances usuellement disponibles : les connaissances théoriques et les connaissances empiriques;

- Il est très performant et sa réalisation modulaire et parametrable lui permet d'être facilement adaptable à différents types d'applications.

Notre principal apport au domaine de recherche en I.A. est donc la proposition d'un nouveau système intelligent hybride : *le système hybride neuro-symbolique INSS*, capable d'acquérir des connaissances théoriques et empiriques d'une façon constructive. Nous avons développé une nouvelle architecture de SHNS et de nouvelles méthodes d'insertion et d'extraction de connaissances à partir des réseaux de neurones. Cette architecture s'est révélée très utile dans les domaines d'application où le système INSS a été utilisé.

Ce travail a ouvert de nouvelles perspectives dans le domaine de l'apprentissage automatique et des systèmes hybrides neuro-symboliques. Cependant, il restent encore des points à étudier et à développer plus profondément :

- L'étude des "règles de haut niveau", leur représentation et leur apprentissage par les réseaux de neurones ; leur application à des problèmes réels, et le développement de méthodes permettant de savoir quand il convient de les utiliser ;

- L'extraction de règles d'ordre 0^+ ou de règles de haut niveau. Actuellement, le système ne permet que l'extraction de règles d'ordre 0 (prédicats du type si-alors avec des opérateurs logiques simples);

- La possibilité de l'étude d'une nouvelle méthode d'extraction de règles à partir des réseaux basée sur des algorithmes génétiques, telle qu'elle a été proposée par G. Reyes dans son rapport d'études du D.E.A.;

- L'étude de la prise en compte du temps dans les réseaux, avec la possibilité d'utiliser des systèmes hybrides neuro-symboliques basés sur des réseaux récurrents;

- Le développement de nouvelles méthodes de validation des connaissances acquises de façon incrémentale. Le processus de validation de connaissances est très lourd du point de vue computationel, car il cherche à établir des relations entre *toutes* les connaissances disponibles. Il va falloir trouver de nouvelles méthodes optimisées pour la réalisation de cette tâche;

- La poursuite des expériences sur le problème de la balance, afin de mieux étudier l'évolution des connaissances et le passage d'un niveau cognitif à l'autre;

- La poursuite des expériences en robotique autonome, qui ont déjà donné des résultats très intéressants avec l'utilisation d'INSS. Le prochain pas serait de tenter d'intégrer différents comportements dans le système, comme par exemple : aller d'un point A à un point B, tout en évitant des obstacles;

- Le développement d'une interface graphique plus conviviale pour le système INSS.

7. REFERENCES BIBLIOGRAPHIQUES

** Liste des documents cités dans le corps du mémoire :*

[ABD 94] Abdi, Hervé. *Les Reseaux de Neurones.* Editions PUG (Presses Universitaires de Grenoble), Collection 'Sciences et Technologies de la Connaissance'. 1994.

[AHA 97] Aha, David W. (Ed.) *Lazy Learning*. Reprinted from: Artificial Intelligence Review, Vol.11:1-5, 432p., Kluwer Academic Publishers, Dordrecht. June 1997.

[AJJ 91] Ajjanagadde, V. & Shastri, L. *Rules and Variables in Neural Nets*. Neural Computation, 3, pp.121-134. 1991.

[ALC 93] d'Alché-Buc, Florence. *Modèles Neuronaux et Algorithmes Constructifs pour l'Apprentissage de Règles de Décision.* Thèse de Doctorat en Sciences. Loratoire LEP, Université de Paris-Sud XI - Centre d'Orsay, France. Décembre 1993.

[ALT 97] Althoff, K.-D.; Bergmann, R.; Wess, S.; Manago. M.; Auriol, E.; Larichev, O. I.; Bolotov, A.; Zhuravlev, Y. I.; Gurov, S. I. *Integration of Induction and Case-Based Reasoning for Decision Support Tasks in Medical Domains: The Inreca Approach.* AI in Medicine Journal, December 1997. Web: http://wwwagr.informatik.uni-kl.de/~althoff/althoff-pub.html

[AMA 95] Amat, J.L.; Yahiaoui, G. *Techniques Avancées pour le Traitement de l'Information : Réseaux de Neurones, Logique Floue, Algorithmes Génétiques*. Editions CEPADUES, France, 1995.

[AMY 96] Amy, Bernard. *Recherches et Perspectives dans le Domaine des Réseaux Connexionnistes.* Rapport de Recherche - DRET, Octobre 1996. Web: http://www-leibniz.imag.fr/RESEAUX/public.html . Ftp: ftp://ftp.imag.fr/pub/LEIBNIZ/RESEAUX-D-AUTOMATES/

[AMY 97] Amy, B.; Danel, V.; Ertel, W.; Gonzalez, J.; Hilario, M.; Malek, M.; Nerot, O.; Osório, F.; Rialle, V.; Rida, A.; Schramm, M; Shultz, S.; Velazco, J. *MIX Medical Application (Final Report - Deliverable D16).* Technical Report, Projet Esprit-BRA 'MIX' - European Community. May 1997. Web: http://www-leibniz.imag.fr/RESEAUX/public.html . Ftp: ftp://ftp.imag.fr/pub/LEIBNIZ/RESEAUX-D-AUTOMATES/MIX-FMR.ps.gz

[AND 95] Andrews, R.; Diederich, J.; Tickle, A.B. *A Survey And Critique of Techniques For Extracting Rules From Trained ANN*. Technical Report - Neurocomputing Research Centre, Queensland University of Technology - Brisbane, Australia. January 1995 (To appear in : Knowledge-Based Systems). Web: http://www.fit.qut.edu.au/NRC/ http:// Ftp: ftp://ftp.fit.qut.edu.au/pub/NRC/tr/ps/QUTNRC-95-01-02.ps.Z

[AND 96] Andrews, Robert & Diederich, Joachim. (Eds.) *Rules and Networks* - Proceedings of the Rule Extraction From Trained Artificial Neural Networks Workshop. SSAISB-AISB'96, University of Sussex, Brighton, UK. 2nd April. Published by QUT - Queensland Univ. of Technology, Australia. 1996.

[AYE 90] Ayel, Marc & Rousset, Marie-Christine. *La Cohérence dans les Bases de Connaissances*. Collection Intelligence Artificielle, Cepadues Editions, France. Août 1990.

[BAL 94] Baluja, Shumeet & Fahlman, Scott E. *Reducing Network Depth in the Cascade-Correlation Learning Architecture.* Carnegie Mellon University. Technical Report -CMU-CS-94-209. October 1994. Web: http://www.cs.cmu.edu/Reports/index.html

[BAL 95] Balakrishnan, Karthik & Honavar, Vasant. *Evolutionary Design of Neural Architectures - A Preliminary Taxonomy and Guide to Literature*. Technical Report: ISU CS-TR-95-01, AI group - Dept. Computer Science, Iowa State University. 1995. Web: http://www.cs.iastate.edu/~honavar/publist.html

[BEC 91] Becraft, W.R.; Lee, P.L.; Newell, R.B. *Integration of Neural Networks and Expert Systems*. In: Proceedings of the International Joint Conference on Artificial Intelligence - IJCAI, Sydney - Autralia, pp.832-837. Morgan-Kaufmann, 1991.

[BIS 97] Bishop, Christopher. *Classification and Regression*. In: Handbook of Neural Computation (section B6.2). E. Fiesler and R. Beale (Eds.) Institute of Physics and Oxford University Press. New York, NY - U.S.A., 1997. Web: http://www.idiap.ch/publications/fiesler-96.1.bib.abs.html or http://www.oup-usa.org/acadref/nc_accs.html

[BLO 96] Blond, P.-Y. *Validation de Connaissances dans un Système Hybride Neuro-Symbolique à Apprentissage Incrémental.* Rapport du D.E.A. en Sciences Cognitives, Laboratoire LEIBNIZ - IMAG / INPG, Grenoble - France. Juin 1996.

[BLU 87] Blumer, A.; Ehrenfeucht, A.; Haussler, D.; Warmuth, M. *Occan's Razor*. Information Processing Letters, 24, pp.377-380. 1987.

[BOU 91] Bourret, P; Reggia, J.; Samuelides, M. *Réseaux Neuronaux : une Approche Connexionniste de l'Intelligence Artificielle*. Editions Teknea, France, 1991.

[BOZ 95] Boz, Olcay. *Knowledge Integration and Rule Extraction in Neural Networks*. Technical Report, EECS, Lehigh University. October 1995. Web: http://www.lehigh.edu/~ob00/integrated.html

[BOZ 97b] Boz, Olcay. *Converting a Trained Neural Net to a Decision Tree* - DECEXT (Decision Tree Extractor). Technical Report - University of Lehigh, U.S.A. Web: http://www.lehigh.edu/~ob00/integrated.html

[BRA 90] Bratko, Ivan. *Prolog Programming for Artificial Intelligence*. Addison-Wesley, 1990.

[BRE 84] Breiman, L.; Friedman, J. H.; Olshen, R. A.; Stone, C. J. *Classification and Regression Trees*. Belmont, CA: Wadsworth. 1984.

[BUC 94] Buckingham, David & Schultz, Thomas R. *A Connectionist Model of the Development of Velocity, Time and Distance Concepts.* Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society, pp. 72-77. Hillsdale, NJ: Erlbaum. 1994. Web: http://www.psych.mcgill.ca/labs/lnsc/html/Lab-Home.html (or Pub-cog-dev.html). Ftp: ftp://ego.psych.mcgill.ca/pub/shultz/vtd.ps.gz

[CAR 92] Carpenter, G. et al. *Fuzzy ARTMAP: A Neural Network Architecture for Incremental Supervised Learning of Analog Multidimensional Maps*. IEEE Transactions on Neural Networks 3(5), pp.698-713. September 1992. Web: http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/fuzzy/systems/artmap/

[CAU 90] Caudill, Maureen. *Driving Solo*. AI Expert, September 1991.

[CAU 92] Caudill, Maureen & Butler, Charles. *Understanding Neural Networks - Vol.1: Basic Networks, Vol.2: Advanced Networks*. Bradford Books, MIT Press, 1992.

[CEN 87] Cendrowska, Jadzia. *PRISM: An Algorithm for Inducing Modular Rules.* International Journal of Man-Machine Studies, Vol.26, N.1,2,4; Vol.27,N.2,3,4. 1987.

[CES 87] Cestnik, B.; Kononenko, I.; Bratko, I. *ASSISTANT 86: A Knowledge-Elicitation Tool for Sophisticated Users*. In : Progress in Machine Learning, I. Bratko and N. Lavrac, Sigma Press, Wilmslow. 1987.

[CHE 97] Chentouf, Rachida. *Construction de Réseaux de Neurones Multicouches pour l'Approximation.* Thèse de Doctorat en Sciences Cognitives, Laboratoire TIRF - INPG, Grenoble - France, Mars 1997.

[COL 85] Colmerauer, A. *Prolog in 10 Figures*. Communications of the ACM, 28:12, pp.1296-1310. 1985.

[COX 92] Cox, Earl. *Integrating Fuzzy Logic into Neural Nets*. AI Expert, June 1992.

[CRA 96a] Craven, M. & Shavlik, J. *Extracting Tree-Structured Representations of Trained Networks*. Advances in Neural Information Processing Systems, MIT Press, Denver. pp.24-30. 1996.

[CRA 96b] Craven, Mark W. *Extracting Comprehensible Models from Trained Neural Networks*. Ph.D. Thesis - Dept. of Computer Science, Univ. of Wisconsin-Madison, U.S.A. 1996. Web: http://www.cs.wisc.edu/~shavlik/mlrg/publications.html

[CRA 97] Crampe, Isabelle. *Révision Interactive dans les Bases de Connaissances par Objets.* Thèse en Doctorat en Informatique, INRIA Rhône-Alpes, France. Octobre 1997.

[CUL 93] Culbert, C.; Riley, G. Donnel, B. *CLIPS Reference Manual, Basic Programming Guide - Version 6.0*. Lyndon B. Johnson Space Center, Software Technology Branch, NASA - U.S.A. 1993. Web: http://www.jsc.nasa.gov/~clips/CLIPS.html or http://home.haley.com/clips.html . Ftp: ftp://ftp.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/expert/systems/clips/0.html

[DAN 97] Danel, Vincent. *Des bases de cas vers l'aide à la connaissance : A propos des comas toxiques*. Thèse de Doctorat, spécialité : Génie Biologique et Médical. Laboratoire TIMC - IMAG, UJF. Grenoble - France. Octobre 1997.

[DAV 91] Davis, Lawrence. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold. 1991.

[DAV 93] David, J.-M.; Krivine, J.-P.; Simmons, R. (Eds.) *Second Generation Expert Systems*. Springer-Verlag, 1993.

[DEC 95] Decloedt, Loïc. *Explicitation de Connaissances dans un Système Hybride d'Intelligence Artificielle*. Rapport du D.E.A. en Informatique, Laboratoire LIFIA - IMAG / INPG, Grenoble - France, Juin 1995. Web: http://www-leibniz.imag.fr/RESEAUX/public.html . Ftp: ftp://ftp.imag.fr/pub/LEIBNIZ/RESEAUX-D-AUTOMATES/DEA/decloedt.dea.ps.gz

[DEC 96] Decloedt, L.; Osório, F.; Amy, B. *RULE_OUT Method: A New Approach for Knowledge Explicitation from Trained Artificial Neural Networks.* In: Rules and Networks

(Proceedings of the AISB'96 - Workshop on Rule Extraction from Trained Neural Nets), Andrews and Diederich Eds., Queensland University of Technology - QUT. pp.34-42, 1996. Web: http://www-leibniz.imag.fr/RESEAUX/public.html . Ftp: ftp://ftp.imag.fr/pub/LEIBNIZ/RESEAUX-D-AUTOMATES/osorio.aisb96.ps.gz

[DEJ 88] De Jong, Kenneth A. *Learning with Genetic Algorithms: An Overview*. Machine Learning, 13, pp.121-138. Kluwer Academic Publishers, Boston, MA - U.S.A. 1988.

[DEJ 90] De Jong, Kenneth A. *Genetic Algorithm-Based Learning*. In: Machine Learning - Vol.3. Y. Kodratoff et al. (Eds.), Chapter 21. Morgan Kaufmann Publishers, San Mateo. 1990.

[DEJ 93a] De Jong, Kenneth. *Apprentissage à partir d'Algorithmes Génétiques*. In: Apprentissage Symbolique - Une Approche de l'Intelligence Artificielle, Tome II. Y. Kodratoff, R. Michalski, J. Carbonell, T. Mitchell (Eds.), Editions Cépaduès, Toulouse - France. 1993.

[DEJ 93b] De Jong, K.; Spears, W.; Gordon, D. *Usign Genetic Algorithms for Concept Learning*. Machine Learning, 13, pp.161-187. Kluwer Academic Publishers, Boston, MA - U.S.A. 1993.

[DIN 92] Dinsmore, John (Ed.) *The Symbolic and Connectionist Paradigms: Closing the Gap*. Lawrence Erlbaum Associates, Hillsdale, New Jersey. 1992.

[DUR 93] Durkin, J. *Expert Systems : Catalog of Applications.* Intelligent Computer Systems, Inc., Akron, Ohio, 1993.

[EIC 96] Eick, Christoph F. ; Kim, Yeong-Joon ; Secomandi, Nicola ; Toto, Ema**.** *DELVAUX - An Environment that Learns Bayesian Rule-Sets with Genetic Algorithms.* In : The Third World Congress on Expert Systems. Seoul, Korea, February, 1996. Web: http://www.cs.uh.edu/~yjkim/ or http://www.cs.uh.edu/~ceick/ceick.html

[ELM 93] Elman, Jeffrey L. *Learning and Development in Neural Networks: The Importance of Starting Small*. Cognition, 48(1993), pp.71-99. 1993. Web: http://crl.ucsd.edu/~elman/ . Ftp: ftp://crl.ucsd.edu/pub/neuralnets/cognition.ps.Z

[FAH 88] Fahlman, Scott E. *An Empirical Study of Learning Speed in Back-Propagation Networks.* Carnegie Mellon University - CMU. Computer Science Technical Report CMU-CS-88-162. September 1988. Web: http://www.cs.cmu.edu/Reports/index.html

[FAH 90] Fahlman, S. E.; Lebiere, C. *The Cascade-Correlation Learning Architecture*. Carnegie Mellon University - CMU, Computer Science Technical Report - CMU-CS-90-100. February

1990. Web: http://www.cs.cmu.edu/Reports/index.html . Ftp: ftp://archive.cis.ohio-state.edu/pub/neuroprose/fahlman.cascor-tr.ps.Z

[FAH 91] Fahlman, Scott E.*The Recurrent Cascade-Correlation Architecture.* Carnegie Mellon University - CMU. Computer Science Technical Report CMU-CS-91-100. May 1991. Web: http://www.cs.cmu.edu/Reports/index.html

[FAH 95] Fahlman, Scott. *Self-Configuring Network Architectures*. (Présentation orale) Workshop on Neural Network Design and Analysis - WNNDA. CUI, Université de Genève, Suisse. Jan. 1995.

[FEI 79] Feigenbaum, E.A., *Themes and Case Studies of Knowledge Engineering.* Expert Systems in the Micro-Eletronic Age, Ed. D. Michie, Edimburg University Press, 1979, pp.3-25.

[FES 96] Fessant, F; Partakelidis, P.; Giacometti, A. *Application des Réseaux de Neurones à L'Intelligence Artificielle : Extraction de connaissances et couplage dynamique dans les systèmes hybrides.* Rapport de fin d'études de la convention DRET n° 94.34.120.00.470.75.01. Laboratoire LEIBNIZ - IMAG / INPG, Grenoble - France. Janvier 1996. Web: http://www-leibniz.imag.fr/RESEAUX/public.html

[FES 97] Fessant, F; Gauthier, E.; Mekrez, I.; Pleynet, B.; Amy, B.; Memmi, D. *Application des Réseaux de Neurones à L'Intelligence Artificielle : Architectures modulaires pour le contôle en robotique.* Rapport de synthèse final, convention DRET n° 96.34.074.00.470.75.65. Laboratoire LEIBNIZ - IMAG / INPG, Grenoble - France. Août 1997. Web: http://www-leibniz.imag.fr/RESEAUX/public.html . Ftp: ftp://ftp.imag.fr/pub/LEIBNIZ/RESEAUX-D-AUTOMATES/

[FIE 94a] Fiesler, E. *Neural Networks Formalization and Classification.* Computer Standard & Interfaces, Special Issue on Neural Networks Standards, John Fulcher (Ed.). V.16, N.3. Elsevier Sciences Publishers, Amsterdam, June, 1994. Web: http://www.idiap.ch/idiap-networks.html . Ftp: ftp://ftp.idiap.ch/pub/papers/neural/fiesler.formalization.ps.Z

[FIE 94b] Fiesler, Emile. *Comparative Bibliography of Ontogenic Neural Networks.* Proceedings of the International Conference on Artificial Neural Nets - ICANN'94. Sorrento, Italy, May 1994. Web: http://www.idiap.ch/idiap-networks.html . Ftp: ftp://ftp.idiap.ch/pub/papers/neural/fiesler.ontogenic-summary.ps.Z

[FIE 97] Fiesler, E. & Beale, R. *Handbook of Neural Computation*. Institute of Physics and Oxford University Press. New York, NY - U.S.A., 1997. Web: http://www.idiap.ch/publications/fiesler-96.1.bib.abs.html or http://www.oup-usa.org/acadref/nc_accs.html

[FIN 79] Findler, N. V. *Associative Networks - Representation and Use of Knowledge by Computers*. Academic Press, New York, 1979.

[FLE 93] Fletcher, J. & Obradovic, Z. *Combining prior Symbolic Knowledge and Constructive Neural Network Learning.* Connection Science, 5(3-4), pp.365-375, 1993.

[FLE 94] Fletcher, Justin Barrows S. *A Constructive Approach to Hybrid Architectures for Machine Learning.* Ph.D. Thesis. Washington State University, August 1994. Web: http://www.eecs.wsu.edu/~zoran/

[FLO 94] Floreano, Dario & Mondada, Francesco. *Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural-Network Driven Robot.* In: From Animals to Animats III - Proceedings of the 3rd. International Conference on Simulation of Adaptive Behaviour. MIT Press - Bradford Books, Cambridge - MA. 1994. Web: http://diwww.epfl.ch/ lami/team/mondada/

[FOR 82] Forgy, Charles L. *Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem.* Artificial Intelligence, 19, pp.17-37. 1982.

[FRE 90] Frean, M. *The Upstart Algorithm: A Method for Constructing and Training Feed-Forward Neural Networks*. Neural Computation, 2, pp.198-209, 1990.

[FU 89] Fu, LiMin. *Integration of Neural Heuristics into Knowledge-Based Inference.* Connection Science, 1, pp.325-340. 1989.

[FU 90] Fu, L.M. & Fu, L.C. *Mapping Rule-Based Systems into Neural Architecture.* Knowledge-Based Systems, 3(1), pp.48-56, March 1990.

[FU 91] Fu, LiMin. *Rule Learning by Searching on Adapted Nets.* In: Proceedings of the Ninth National Conference on Artificial Intelligence, pp.590-595. Anaheim - CA. 1991.

[FU 92] Fu, LiMin. *Knowledge Base Refinement by Backpropagation.* Data and Knowledge Engineering, 7, pp.35-46. 1992.

[FU 93] Fu, LiMin. *Knowledge-Based Connectionism for Revising Domain Theories*. IEEE Transactions on Systems, Man and Cybernetics, Vol.23, N.1, January/February 1993.

[FU 94] Fu. LiMin. *Neural Networks in Computer Intelligence*. McGraw-Hill Publishing Inc., 1994.

[FU 94a] Fu, LiMin. *Rule Creation from Neural Networks*. IEEE Transactions on Systems, Man and Cybernetics, Vol.24, N.8. August 1994.

[GAL 88] Gallant, S. I. *Connectionist Expert Systems*. Communications of the ACM, 31(2), pp.152-169. 1988.

[GAL 88] Gallant, S. I. *Connectionist Expert Systems*. Communications of the ACM, 31(2), pp.152-169. February 1988.

[GAL 93] Gallant, S. I. *Neural Network Learning and Expert Systems*. MIT Press, Cambridge, MA. 1993.

[GAL 93] Gallant, S. I. *Neural Network Learning and Expert Systems*. Bradford Books / MIT Press, Cambridge - MA, U.S.A. 1993.

[GAL 95] Gallant, S. I. *Expert Systems and Decision Systems Using Neural Networks.* In: The Handbook of Brain Theory and Neural Networks. M. Arbib (Ed.), MIT Press, pp.377-380.1995.

[GAL 95] Gallant, Stephen I. *Expert Systems and Decision Systems Using Neural Networks.* In: The Handbook of Brain Theory and Neural Networks. M. Arbib (Ed.), MIT Press, 1995.

[GIA 92] Giacometti, Arnaud. *Modèles Hybrides de l'Expertise*. Thèse de Doctorat en Informatique et Réseaux, Lab. LIFIA - IMAG, Grenoble / ENST Paris - France, Novembre 1992. Web: http://www-leibniz.imag.fr/RESEAUX/public.html . Ftp: ftp://ftp.imag.fr/pub/LEIBNIZ/ESEAUX-D-AUTOMATES/giacometti.these.ps.tar.gz

[GIA 93] Giarratano, Joseph C. *CLIPS User's Guide - Version 6.0*. Lyndon B. Johnson Space Center, Software Technology Branch, NASA - U.S.A. 1993. Web: http://www.jsc.nasa.gov/~clips/CLIPS.html or http://home.haley.com/clips.html . Ftp: ftp://ftp.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/expert/systems/clips/0.html

[GIA 94] Giarratano, Joseph & Riley Gary. *Expert Systems: Principles and Programming.* PWS Publising, 1994.

[GOL 89] Goldberg, David E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA - U.S.A. 1989.

[GOL 94] Goldberg, D. & Corruble, V. *Algorithmes Génétiques : Exploration, Optimisation et Apprentissage Automatique*. Addison-Wesley, Paris - France. 1994.

[GON 96] González, J.C.; Velasco, J.R.; Iglesias, C.A. *Prototype of a Fuzzy-Neural Hybrid Model*. Evaluation report. Rapport Technique MIX/WP2/UPM/3.0, UPM, MIX Project deliverable D7. 1996.

[GOO 95] Goonatilake, Suran & Khebbal, Sukdev. (Eds.) *Intelligent Hybrid Systems*. John Wiley and Sons, London. 1995.

[GRE 93] Greene, David P. & Smith, Stephen, F. *Competition-Based Induction of Decision Models from Examples*. Machine Learning, 13, pp.229-258. Kluwer Academic Publishers, Boston, MA - U.S.A. 1993.

[GUT 91] Gutknecht, M.; Pfeifer, R.; Stolze, M. *Cooperative Hybrid Systems*. In: Proceedings of the International Joint Conference on Artificial Intelligence - IJCAI, pp.824-829. Morgan-Kaufmann Publisher, 1991.

[HAL 94] Halgamuge, S.K. *FuNeGen - Fuzzy Neural System*. Publié dans le WEB-Internet. Web: http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/fuzzy/systems/funegen/

[HAL 95] Hallam, John (Ed.).*Hybrid Problems, Hybrid Solutions*. Proceedings of the AISB'95. IOS Press, 1995.

[HAL 97] Haley, Paul. *ECLIPSE* - WWW Internet site, Halley Enterprise. 1997. Web: http://home.halley.com/eclipse.html

[HAT 89] Haton, J. P. & Haton, M. C. *L'Intelligence Artificielle.* Collection 'Que sais-je', No. 2444, PUF, Paris - France. 1989.

[HAT 90] Haton, Jean-Paul. *Les Systèmes à Bases de Connaissances.* In: Artificial Intelligence IV: Methodology, Systems, Applications - Proceedings of AIMSA'90 - Albena, Bulgaria. Jorrand, Ph. and Sgurev, V. (Eds.). Amsterdam, APIA - Elsevier Science Publishers. Sept. 1990.

[HAY 83] Hayes-Roth, F.; Waterman, D.; Lenat, D. (Eds.). *Building Expert Systems,* V.1. Addison-Wesley Publishing Company Inc., Reading, Massachusetts, U.S.A., 1983.

[HAY 94] Hayashi, Y. & Buckley, J. J. *Approximations Between Fuzzy Expert Systems and Neural Networks*. International Journal of Approximate Reasoning, Vol.10, pp.63-73. 1994

[HAY 96] Hayward, Ross & Diederich, Joachim. *Implementation of a SHRUTI Knowledge Representation and Reasoning System.* In: Proceedings of the ECAI'96 Workshop on Neural Networks and Structured Knowledge. Budapest, Hungary. August 1996.

[HEN 89] Hendler, J. *Marker-Passing over Microfeatures: Towards a Hybrud Symbolic/Connectionist Model*. Connection Science, 13, pp.79-106. 1989.

[HER 94] Herault, J. & Jutten, C. *Réseaux Neuronaux et Traitement du Signal.* Editions Hermès, Paris, 1994.

[HIL 93] Hilario, Melanie. *MIX: Modular Integration of Connectionist and Symbolic Processing in Knowledge-Based Systems.* Proposal for Basic Research Project - Esprit BRA, EEC, n° 09119.

[HIL 94a] Hilario, M.; Pellegrini, C.; Alexandre, F. *Modular Integration of Connectionist and Symbolic Processing.* In : Knowledge-Based Systems - Proceedings of the ISIKNH'94: International Symposium on Integrating Knowledge and Neural Heuristics, pp. 123-132. Pensacola, Florida: May 1994. Ftp: ftp://cui.unige.ch/AI/mix/isiknh-94.ps

[HIL 94b] Hilario, Melanie et al. (Eds.) *Combining Symbolic and Connectionist Procesing*. Proceedings of the ECAI'94 Workshop. Amsterdan, The Netherlands, August 1994.

[HIL 95c] Hilario, M.; Rida, Ahmed; Orsier, Bruno; Stückelberg, M. *Model-Based Knowledge Engineering for MLP Configuration*. Technical Report - Implementation Report, Projet Esprit-BRA 'MIX' - European Community. Deliverable S4.1. December 1995. Web: http://cuiwww.unige.ch/AI-group/home.html (also in MIX ftp-loria repository)

[HIL 96b] Hilario, M. *An Overview of Strategies for Neurosymbolic Integration.* In: Connectionist-Symbolic Integration: From Unified to Hybrid Approaches. Ron Sun (Ed.) - Chapter 2. Kluwer Academic Publishers, 1996. Ftp: ftp://cui.unige.ch/AI/

[HIN 89] Hinton, G. E. *Connectionist Learning Procedures*. Artificial Intelligence Journal, 40, pp.185-234. Elsevier Publishing, 1989.

[HOL 75] Holland, John H. *Adaptation in Natural and Artificial Systems*. University of Michigan Press. 1975.

[HOL 86] Holland, J. *Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems.* In: Machine Learning - An Artificial Intelligence Approach, Volume II. R. Michalski et al. (Eds.), Morgan Kaufmann. 1986.

[HOL 87] Holland, J.; Burks, A. *Adaptive Computing System Capable of Learning and Discovery*. U.S. Patent 4,697,242. Issued September 29, 1987.

[HOM 95] Homaifar, A. & McCormick, E. *Simultaneous Design of Membership Functions and Rule Sets for Fuzzy Controllers Using Genetic Algorithms.* IEEE Transactions on Fuzzy Systems, Vol.3, Issue 2. May 1995.

[HON 94] Honavar, V. & Uhr, L. (Eds.) *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration*. Academic Press, Boston - MA. 1994.

[HON 95] Honavar, V., and Uhr, L. *Integrating Symbol Processing and Connectionist Networks.* In: Intelligent Hybrid Systems. S. Goonatilake and S. Khebbal (Eds.), London, John Wiley and Sons. 1995. Web: http://www.cs.iastate.edu/~honavar/Papers/ (hybrid94.ps)

[IIT 96] ITI - Institute for Information Technology. *Fuzzy CLIPS*. Publié dans le WEB-Internet. IIT - NRC, Canada. Web: http://ai.iit.nrc.ca/fuzzy/fuzzy.html

[JAN 93] Jang, Jyh-Shing. *ANFIS: Adaptive-Network-Based Fuzzy Inference Systems*. IEEE Transactions on Systems, Man and Cybernetics. May 1993. Web: http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/fuzzy/systems/anfis/

[JAN 96] Jang, J.-S. R.; Sun, C.-T.; Mizutani, E. *Neuro-Fuzzy and Soft Computing*. Prentice Hall, 1997.

[JAN 96] Jang, J.-S. R.; Sun, C.-T.; Mizutani, E. *Neuro-Fuzzy and Soft Computing*. Prentice Hall Publishing Co., September 1996. Web: http://www.cs.nthu.edu.tw/~jang/book/

[JOD 94a] Jodouin, Jean-François. *Les Réseaux de neurones : Principes et définitions.* Editions Hermès, Paris, 1994.

[JOD 94b] Jodouin, Jean-François. *Les Réseaux Neuromimétiques : Modèles et applications.* Editions Hermès, Paris, 1994.

[KAN 92] Kandel, Abraham & Langholz, Gideon. *Hybrid Architectures for Intelligent Systems.* CRC Press, Boca Raton - Florida, 1992.

[KAS 90] Kasabov, Nikola K. *Hybrid Connectionist Rule-Based Systems.* In: Artificial Intelligence IV: Methodology, Systems, Applications. Proceedings of AIMSA'90 - Albena, Bulgaria. P. Jorrand and V. Sgurev (Eds.). Amsterdam, APIA - Elsevier Science Publishers. September 1990.

[KAS 95] Kasabov, N. *Learning Fuzzy Rules and Approximate Reasoning in Fuzzy Neural Networks and Hybrid Systems.* Fuzzy Sets and Systems. Special Issue, pp.1-19. 1995. Web: http://divcom.otago.ac.nz:800/COM/INFOSCI/KEL/

[KAS 96] Kasabov, N.K. *Foundations of Neural Networks, Fuzzy Systems and Knowledge Engeneering*. Cambridge, Massachussets, MIT Press. 1996. Web: http://divcom.otago.ac.nz:800/COM/INFOSCI/KEL/

[KIL 97] Kilgour, Richard. *FuzzyCOPE 1, FuzzyCOPE2 and FuNN*. Publié dans le WEB-Internet. Web: http://divcom.otago.ac.nz:800/COM/INFOSCI/KEL/software.htm

[KLE 56] Kleene, S. C. *Representation of Events in Nerve Nets and Finite Automata*. In: Automata Studies. Shannon, C. and McCarthy, J. (Eds.). pp.3-41, Princeton University Press, Princeton - NJ. 1956.

[KOL 93] Kolodner, J. *Case-Based Reasoning*. Morgan Kaufmann Publishers Inc. 1993.

[KON 96] König, Thomas. *Comp.graphics.apps.gnuplot FAQ* (Usenet Frequently Asked Questions - Gnuplot). Web: http://wwwfg.rz.uni-karlsruhe.de/~ig25/gnuplot-faq.html

[KOZ 92] Koza, John R. *Genetic Programming : On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. MIT Press, 1992.

[KRO 92] Krovvidy, S. & Wee, W.G. *An Intelligent Hybrid System for Wastewater Treatment*. In: Hybrid Architectures for Intelligent Systems (Chapter 17), A. Kandel and G. Langholz (Eds.) - CRC Press, Boca Raton, Florida. 1992.

[KRZ 97] Krzysztof, J. C. & Pedrycz, W. *Neuro-Fuzzy Systems*. In: Handbook of Neural Computation (section D1). E. Fiesler and R. Beale (Eds.) Institute of Physics and Oxford University Press. New York, NY - U.S.A., 1997. Web: http://www.idiap.ch/publications/fiesler-96.1.bib.abs.html or http://www.oup-usa.org/acadref/nc_accs.html

[LAB 93] Labbi, Abderrahim. *Sur l'Approximation et les Systèmes Dynamiques dans les Réseaux Neuronaux*. Thèse de Doctorat en Mathématiques Appliquées, Laboratoire LIFIA - IMAG, Grenoble - France, Décembre 1993.

[LAL 96] Lallement, Yannick. *Intégration Neuro-Symbolique et Intelligence Artificielle: Applications et Implantation Parallèle*. Thèse de Doctorat en Informatique, CRIN-INRIA Lorraine, Université Henri Poincarré - Nancy I, France, Juin 1996. Web: http://www.loria.fr/exterieur/equipe/rfia/cortex/ . Ftp: ftp://ftp.loria.fr/pub/loria/rfia/publications/cortex

[LEC 88] LeCun, Y. *Learning Process in an Asymetric Threshold Network*. Disordered Systems and Biological Organisation. E Beinenstock, F. Fogelman-Soulié and G. Weisbuch (Eds.), Springer-Verlag, Berlin, pp.33-48. 1988.

[LEC 90] LeCun, Y.; Denker, J. S.; Solla, S.A. *Optimal Brain Damage* ; Advances in Neural Information Processing Systems 2, D.S. Touretzky (Ed.), Morgan Kauffmann publishers, 1990.

[LEE 95] Lee, S.-J. & Jone, M.-T. *A Symbolic Logic Approach of Deriving Initial Neural Network Configurations for Supervised Classification*. Computers and Artificial Intelligence, vol.14(4), pp.317-337. 1995.

[LET 92] Letz, R.; Bayerl, S.; Bibel, W. *SETHEO: A High-Performance Theorem Prover*. Journal of Automated Reasoning, 8(2), pp.183-212. 1992.

[LIA 94] Liaw, Andy & Crawford, Dick. *Gnuplot 3.5 - User's Guide*. Software introduction guide (available through the WWW-Internet). Web: http://www.cs.dartmouth.edu/gnuplot_info.html . Ftp: ftp://ftp.dartmouth.edu/pub/gnuplot/

[LIN 80] Lindsay, R.; Buchanan, B.; Feigenbaum, E.; Lederberg, J. *Applications of Artificial Intelligence for Chemical Inference: The Dendral Project.* McGraw-Hill, NY, 1980.

[MAH 93] Mahoney, J. & Mooney, R. *Combining Connectionist and Symbolic Learning to Refine Certainty-Factor Rule-Bases*. Connection Science, 5(1993), pp.339-364. 1993. Web: http://net.cs.utexas.edu/users/ml/ . Ftp: ftp://ftp.cs.utexas.edu/pub/mooney/papers

[MAH 96] Mahoney, J. Jeffrey. *Combining Symbolic and Connectionist Learning Methods to Refine Certainty-Factor Rule Bases*. Ph.D. Thesis, Dept. of Computer Science, Univ. of Texas at Austin, May 1996. Web: http://net.cs.utexas.edu/users/ml/ . Ftp: ftp://ftp.cs.utexas.edu/pub/mooney/papers (rapture-dissertation-96.ps.Z)

[MAL 95] Malek, M. & Labbi, A. *Integration of Case-Based Reasoning and Neural Approaches for Classification*. Rapport Technique - RT 131, LIFIA, IMAG. Grenoble, France. 1995. Web: http://www-leibniz.imag.fr/RESEAUX/public.html

[MAL 96a] Malek M., Amy B. *A Preprocessing Model for Integrating Case-Based Reasoning and Prototype-Based Neural Network.* Connectionist Symbolic Integration, Lawrence, Erlbaum Associates. 1996. Web: http://www-leibniz.imag.fr/RESEAUX/public.html

[MAL 96b] Malek, Maria. *Un modèle hybride de mémoire pour le raisonnement à partir de cas.* Thèse de Doctorat en Informatique, Lab. LEIBNIZ - Université Joseph Fourier, IMAG, Grenoble - France, Octobre 1996. Web: http://www-leibniz.imag.fr/RESEAUX/public.html . Ftp: ftp://ftp.imag.fr/pub/LEIBNIZ/RESEAUX-D-AUTOMATES/malek.these.ps.gz

[MAR 93] Mareschal, D. & Schultz, T. R. *A Connectionnist Model of the Development of Seriation.* Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society, pp. 676-681. Hillsdale, NJ: Erlbaum. 1991. Web: http://www.psych.mcgill.ca/labs/lnsc/html/Lab-Home.html (or Pub-cog-dev.html). Ftp: ftp://ego.psych.mcgill.ca/pub/shultz/seriat.ps.gz

[MCC 43] McCulloch, W. S. & Pitts, W. A. *A Logical Calculus of the Ideas Immanent in Nervous Activity*. Bulletin of Mathematical Biophysics, 5:115-133. 1943.

[MCC 89] McClelland, J.L. *Parallel Distributed Processing : Implications for Cognition and Development.* In : Morris, R. G. M. (Ed.). Parallel Distributed Processing : Implications for Psychology and Neurobiology. Oxford University Press. 1989.

[MED 92] Medsker, L. & Bailey, D. *Models and Guidelines for Integrating Expert Systems and Neural Networks*. In: Hybrid Architectures for Intelligent Systems (Chapter 8), A. Kandel and G. Langholz (Eds.) - CRC Press, Boca Raton, Florida. 1992.

[MED 94] Medsker, L.R. *Hybrid Neural Network and Expert Systems*. Kluwer Academic Publishers, Boston. 1994.

[MED 94] Medsker, L.R. *Hybrid Neural Network and Expert Systems*. Kluwer Academic Publishers, Boston. 1994.

[MED 95] Medina, C. & Pratt, L.Y. *NNES: A Neural Network Explanation System for Transforming Trained Neural Networks into Decision Trees.* International Joint Conference on Artificial Intelligence - IJCNN'95 (Submitted), 1995. Web: http://vita.mines.colorado.edu:3857/0/lpratt . Ftp: ftp://franklinite.mines.edu/pub/pratt-papers/nnes-ijcai.ps.Z

[MED 95] Medsker, L.R. *Hybrid Intelligent Systems*. Kluwer Academic Publishers, Boston. 1995.

[MEK 97] Mekrez, Idriss. *Application de Réseaux de Neurones Récurrents au Contrôle d'un Robot Autonome.* Rapport du D.E.A. en Sciences Cognitives, Laboratoire LEIBNIZ - IMAG / INPG, Grenoble - France. Juin 1997. Web: http://www-leibniz.imag.fr/RESEAUX/public.html . Ftp: ftp://ftp.imag.fr/pub/LEIBNIZ/RESEAUX-D-AUTOMATES/

[MEM 97] Memmi, Daniel. *The Implicit/Explicit Distinction in Connectionism and Symbolic Artificial Intelligence*. Cognitive Systems, 5(1), pp.17-35. October 1997.

[MIC 96] Michalewicz, Zbigniew. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag. 1996.

[MIC 96] Michel, Olivier. *Khepera Simulator version 2.0 - User Manual*. Université de Nice - Sophia Antipolis, Laboratoire I3S, CNRS. France. March 1996. Web: http://wwwi3s.unice.fr/~om/khep-sim.html ou http://diwww.epfl.ch/lami/team/michel/khep-sim/

[MIN 75] Minsky, Marvin. *A framework for Representing Knowledge*. In: The Psychology of Computer Vision, Wiston (Ed.), MacGraw-Hill, New York. 1975.

[MIN 90] Minsky, Marvin. *Logical vs. Analogical or Symbolic vs. Connectionist or Neat vs. Scruffy*. In: Artificial Intelligence at MIT - Expanding Frontiers, P. Winston (Ed.), vol.1, MIT Press (reprinted in AI Magazine). 1990. Web: http://www.ai.mit.edu/people/minsky/minsky.html

[MIT 93] Mitchell, T. & Thrun, S. *Explanation-Based Learning: A Comparison of Symbolic and Neural Network Approaches*. In: Proceedings of the Tenth International Conference on Machine Learning, P. Utgoff (Ed.), Morgan Kaufmann, San Mateo, CA - U.S.A. 1993. Web: http://www.informatik.uni-bonn.de/~thrun/mitchell.EBL-ml93.ps.Z

[MIT 93] Mitchell, Tom M. & Thrun, Sebastian B. *Explanation Based Learning: A Comparison of Symbolic and Neural Network Approaches*. In: Proceedings of Tenth International Conference on Machine Learning, Amherst, MA. 1993.

[MIT 97] Mitchell, Tom M. *Machine Learning*. McGraw-Hill Publishing Company, McGraw-Hill Series in Computer Science (Artificial Intelligence). 1997.

[MOL 90] Moller, Martin F. *A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning*. Technical Report PB-339 - Compter Science Dept., University of Aarhus, Denmark. November 1990. Ftp: ftp://archive.cis.ohio-state.edu/pub/neuroprose/moller.conjugate-gradient.ps.Z

[MON 93] Mondada, F.; Franzi, E.; Ienne, P. *Mobile Robot Miniaturization: a Tool for Investigation in Control Algorithms*. ISER'93, Kyoto, Japan, October 1993. Web: http://diwww.epfl.ch/lami/robots/K-family/ . Ftp: ftp://lamiftp.epfl.ch/khepera/papers/ mondada.ISER93.ps.Z

[MOO 94] Moody, J. *Prediction Risk and Architecture Selection for Neural Networks* ; From Statistics to Neural Networks: Theory and Pattern Recognition Applications, NATO ASI Series F. Springer-Verlag, 1994. Web: http://www.cse.ogi.edu/~moody/pubs.html . Ftp: ftp://neural.cse.ogi.edu/pub/neural/papers/moody94.predictionrisk.ps.Z

[NAD 93] Nadal, Jean-Pierre *Réseaux de neurones : de la physique a la psychologie*. Editeur Armand Colin, 1993.

[NAU 95b] Nauck, Detlef. *Beyond Neuro-Fuzzy: Perspectives and Directions*. In: Proceedings of the Third European Congress on Intelligent Techniques and Soft Computing - EUFIT'95, Aachen, pp.1159-1164. August 1995. Web: http://www.cs.tu-bs.de/~nauk

[NAU 97] Nauck, D.; Nürnberger, A.; Kruse, R. *NEFCON, NEFCLASS and NEFPROX*. Publié dans le WEB-Internet. Web: http://fuzzy.cs.uni-magdeburg.de/nnfuz.html or http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/fuzzy/systems/nefcon

[NAZ 97] Nazareth, Derek. *Knowledge-Based Systems Validation, Verification & Testing*. Ressource Guide - Available through Internet - University of Wisconsin-Milwaukee (UWM). 1997. Web: http://www.uwm.edu/~derek/kbsvvt/

[NIK 97] Nikolopoulos, Chris. *Expert Systems - Introduction to First and Second Generation and Hybrid Knowledge Based Systems.* Marcel Dekker Inc. Press, 1997.

[NIL 80] Nilsson, N. J. *Principles of Artificial Intelligence*. Tioga Press, Palo Alto, California - U.S.A. 1980.

[NIL 97] Nilsson, Nils. *Introduction to Machine Learning* (Draft of a proposed new textbook). Robotics Lab., Dept. of Computer Science, Stanford University - CA, U.S.A. 1997. Web: http://robotics.stanford.edu/people/nilsson/mlbook.html

[NOL 94] Nolfi, Stefano; Parisi, Domenico; Elman, Jeffrey L. *Learning and Evolution in Neural Networks*. Adaptive Behavior, 3(1), pp.5-28. 1994. Web: http://crl.ucsd.edu/~elman/

[OPI 95a] Opitz, D.W. *An Anytime Approach to Connectionist Theory Refinement: Refining the Topologies of Knowledge-Based Neural Networks*. Ph.D. Thesis, University of Wisconsin-Madison. Web: http://www.cs.wisc.edu/~shavlik/uwml.html . Ftp: ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/opitz.thesis.ps.Z

[OPI 95b] Opitz, D.W. & Shavlik, J. W. *Dynamically Adding Symbolically Meaningful Nodes to Knowledge-Based Neural Networks*. Knowledge-Based Systems, 8(6), pp.301-311. 1995. Web: http://www.cs.wisc.edu/~shavlik/abstracts/opitz.kbs95.ps.abstract.html . Ftp: ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/opitz.kbs95.ps

[OPI 97] Opitz, D.W. & Shavlik, J. W. *Connectionist Theory Refinement: Genetically Searching the Space of Network Topologies*. JAIR - Journal of Artificial Intelligence Research, 6, pp.177-209. Morgan Kaufmann Publishers, 1997. Web: http://www.cs.wisc.edu/~shavlik/uwml.html. Ftp: ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/opitz.jair97.ps

[ORS 94] Orsier, B.; Amy, B.; Rialle, V.; Giacometti, A. *A Study of the Hybrid System SYNHESYS.* In: Proceedings of the ECAI-94 Workshop - Combining Symbolic and

Connectionnist Processing, August 1994 - Amsterdam. 1994. Web: http://cuiwww.unige.ch/AI-group/staff/orsier.html

[ORS 95] Orsier, Bruno. *Etude et Application de Systèmes Hybrides NeuroSymboliques.* Thèse en Informatique, Laboratoire LIFIA-IMAG, UJF - Grenoble, 1995. Web: http://www-leibniz.imag.fr/RESEAUX/public.html . Ftp: ftp://ftp.imag.fr/pub/LEIBNIZ/RESEAUX-D-AUTOMATES/orsier.these.ps.gz

[ORS 97] Orsier, B. & Labbi, A. *NESSY3L: A Step Toward Fully-Integrated Hybrid Systems*. In: Connectionist-Symbolic Integration: From Unified to Hybrid Approaches (Chapter 9), R. Sun and F. Alexandre (Eds.). Lawrence Erlbaum Associates, 1997. Web: http://cuiwww.unige.ch/AI-group/staff/orsier.html

[OSO 95a] Osório, Fernando S. & Amy, Bernard. *INSS: Un Système Hybride avec l'apprentissage à partir de règles et d'exemples*; (Texte en Portugais: INSS: Um Sistema Híbrido Simboli-Connexionista com Aprendizado à partir de Regras e de Exemplos), Panel'95 - XXI Latin-American Conference on Computer Science, Canela, Brazil, August 1995. Web: http://www-leibniz.imag.fr/RESEAUX/osorio/articles/diret.html

[OSO 96] Osório, F. S. *Rule-Extraction from Trained Artificial Neural Networks* (Notes from the AISB'96 Rule-Extraction from ANN Workshop). AISB Quaterly No.95 - SSAISB, Univ. of Sussex, Brighton - UK. Summer 1996. Web: http://www-leibniz.imag.fr/RESEAUX/osorio/articles/diret.html

[OSO 97] Osório, Fernando S. & Amy, Bernard. *Apprentissage Automatique Constructif : Un Nouveau Modèle Neuro-Symbolique.* VII[e] Colloque AIDRI'97 - Apprentissage: Des principes naturels aux méthodes artificielles - Université de Genève, Suisse, Juin 1997. Web: http://www-leibniz.imag.fr/RESEAUX/public.html . Ftp: ftp://ftp.imag.fr/pub/LEIBNIZ/RESEAUX-D-AUTOMATES/osorio.aidri97.ps.gz

[PAR 85] Parker, D. B. *Learning-Logic*. Technical Report TR-47, Center for Computational Research in Economics and Management Science. MIT - Massachusetts Institute of Technology, Cambridge - MA. April 1985.

[PAR 96] Partakelidis, P.; Fessant, F; Amy, B. *Application des Réseaux de Neurones à L'Intelligence Artificielle : Application de l'apprentissage par renforcement à l'évitement d'obstacle et à la recherche d'une source lumineuse (simulateur de robot Khepera)*. Compte rendu d'expériences - Rapport de fin d'étude, DRET - Laboratoire LEIBNIZ - IMAG / INPG,

Grenoble - France. Octobre 1996. Web: http://www-leibniz.imag.fr/RESEAUX/public.html . Ftp: ftp://ftp.imag.fr/pub/LEIBNIZ/RESEAUX-D-AUTOMATES/

[PAT 97] Patel, M. & Honavar, V. (Eds.) *Advances in Evolutionary Synthesis of Neural Systems*. Boston, MA - U.S.A. MIT Press, 1997.

[PER 96a] Perez, Christophe. *Connexion du simulateur au contrôleur temps réel du robot Vortex - Etude et conception d'un prototype de système de détection de pannes*. Rapport de stage, D.E.S.S. Informatique et Intelligence Artificielle, Université AixMarseille II - IFREMER, Centre de Toulon. Juillet 1996.

[PER 96b] Perez, Christophe. *Détection de Pannes : Mise en correspondance en temps-réel de l'état du robot Vortex avec son modèle*. Documentation technique - Diagnostic de Pannes. Rapport technique réf. DITI/SM/RIA/96-497. Laboratoire de Robotique et d'Intelligence Artificielle, IFREMER, Toulon, France. Juillet 1996.

[PIT 90] Pitrat, Jacques. *Métaconnaissances : Futur de l'I.A.* Editions Hermès, Paris - France. 1990.

[PLE 97] Pleynet, Benoit. *Réalisation d'un Système Hybride pour la Navigation en Robotique*. Rapport du D.E.A. en Informatique, Laboratoire LEIBNIZ - IMAG / INPG, Grenoble - France. Juin 1997. Web: http://www-leibniz.imag.fr/RESEAUX/public.html . Ftp: ftp://ftp.imag.fr/pub/LEIBNIZ/RESEAUX-D-AUTOMATES/

[POL 90] Pollack, J.B. *Recursive Distributed Representations*. Artificial Intelligence, 46, pp.77-105. 1990.

[POM 91] Pomerleau, D.A.; Gowdy, J.; Thorpe, C.E. *Combining Artificial Neural Networks and Symbolic Processing for Autonomous Robot Guidance*. Engineering Applications of Artificial Intelligence, 4(4), pp.279-285. 1991.

[POM 92] Pomerleau, Dean A. *Neural Network Perception for Mobile Robot Guidance*. Ph.D. dissertation, Carnegie-Mellon University. (Also edited as a book by Kluwer Academic Publishing). February 1992. Web: http://www.cs.cmu.edu/afs/cs/project/alv/member/www/projects/ALVINN.html

[POR 97] Porto, V. W. *Neural-Evolutionary Systems*. In: Handbook of Neural Computation (section D2). E. Fiesler and R. Beale (Eds.) Institute of Physics and Oxford University Press.

New York, NY - U.S.A., 1997. Web: http://www.idiap.ch/publications/fiesler-96.1.bib.abs.html or http://www.oup-usa.org/acadref/nc_accs.html

[POS 92] Posey, C.; Kandel, A.; Langholz, G. *Fuzzy Hybrid Systems*. In: Hybrid Architectures for Intelligent Systems. A. Kandel and G. Langholz (Eds.). CRC Press, Boca Raton - Florida, 1992.

[POT 92] Potter, Michell. *A Genetic Cascade-Correlation Learning Algorithm*. In: Proceedings of the COGANN-92 Internetional Workshop on Combinations of Genetic Algorithms and Neural Networks, pp.123-133, IEEE Computer Society Press, 1992.

[QUI 79] Quinlan, J. Ross. *Discovering Rules by Induction from Large Collections of Examples*. In: D. Michie (Ed.), Expert Systems in the Micro Electronic Age. Edimburgh, UK: Edimburgh University Press. 1979.

[QUI 86] Quinlan, J. Ross. *Induction of Decision Trees*. Machine Learning, 1, pp.81-106. Reprinted in Readings in Machine Learning, J. Shavlik and T. Dietterich (Eds.), San Mateo, CA: Morgan Kaufmann, 1991. Reprinted in Readings in Knowledge Acquisition and Learning, B. Buchanan and D. Wilkins (Eds.), San Mateo, CA: Morgan Kauffman, 1992.

[QUI 93] Quinlan, John Ross. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA - U.S.A. 1993.

[QUI 96] Quinlan, J. R. *Improved Use of Continuous Attributes in C4.5*. JAIR - Journal of Artificial Intelligence Research, vol. 4, pp. 77-90, 1996. Web: http://www.cs.su.oz.au/People/quinlan.html

[REC 89] Rechenmann, F.; Fontanille, P. Uvietta, P. Shirka *: Manuel d'utilisation*. Doc. Interne, INRIA - Laboratoire ARTEMIS, IMAG, Grenoble - France. 1989.

[REY 97] Reyes Salgado, Gerardo. *Étude des Connaissances dans les Réseaux de Neurones Artificiels: Representation et Explicitation de Règles de Haut Niveau.* Rapport du D.E.A. en Sciences Cognitives. Laboratoire LEIBNIZ - IMAG/INPG. Grenoble, France. Juin 1997. Web: http://www-leibniz.imag.fr/RESEAUX/public.html . Ftp: ftp://ftp.imag.fr/pub/LEIBNIZ/RESEAUX-D-AUTOMATES/DEA/reyes.dea.tar.gz

[REY 97a] Reyes, G.; Osório, F.; Amy, B. *Neural Learning of "High Level Rules": The Balance Scale Problem*. HELNET'97 - International Workshop on Neural Networks. Montreux, Suisse. October 1997.

[RIE 93] Riedmiller, Martin & Braun, Heinrich. *A Direct Adaptative Method for Faster Backpropagation Learning: The RPROP Algorithm.* Proceedings of the IEEE International Conference on Neural Networks. San Francisco - CA - USA. 1993. Web: http://i11www.ira.uka.de/~riedml/ (or ~neuro)

[ROC 92] Rocha, A.F. & Yager, R.R. *Neural Nets and Fuzzy Logic*. In: Hybrid Architectures for Intelligent Systems. A. Kandel and G. Langholz (Eds.). CRC Press, Boca Raton - Florida, 1992.

[ROS 59] Rosenblatt, F. *Principles of Neurodynamics*. New York, Spartan Books. 1962.

[ROU 98] Rouzier, Sophie. *Réseaux Modulaires* (Titre provisoire). Thèse de Doctorat en Sciences Cognitives, Eq. Réseaux d'Automates - Lab. LEIBNIZ - IMAG, Grenoble - France, 1998 (rédaction en cours).

[RUM 86a] Rumelhart, D. & McClelland, J.L. (Eds.) *Parallel Distributed Processing - Explorations in the Microstructure of Cognition - Vol.1: Foundations, Vol.2: Psychological and Biological Models*. Cambridge: MIT Press, 1986.

[RUM 86b] Rumelhart, D.; Hinton, G. & Willians, R. *Learning Internal Representations by Error Propagation.* In : Rumelhart & McClelland: Parallel Distributed Processing - Explorations in the Microstructure of Cognition - Vol.1: Foundations. Cambridge: MIT Press, 1986.

[SAH 95] Sahami, Mehran. *Generating Neural Networks Through the Induction of Threshold Logic Uni Trees*. In: Proceedings of the First International IEEE Symposium on Intelligence in Neural and Biological Systems. Washington, DC. May 1995.

[SAI 88] Saito, K. & Nakano, R. *Medical Diagnostic Expert System based on PDP Model*. In: Proceedings of IEEE International Conference on Neural Networks, Vol.1, pp.255-262. 1988.

[SAL 96] Salomon, Ralf & van Hemmen, Leo. Accelerating Backpropagation through Dynamic Self-Adaptation. Neural Networks, vol.9(4), pp.589-601, Pergamon Press. 1996.

[SAM 92] Samad, T. *Hybrid Distributed/Local Connectionist Architectures*. In: Hybrid Architectures For Intelligent Systems, A. Kandel and G. Langholz (Eds.), Boca Raton - CRC Press. 1992.

[SAN 97] Sandia National Labs. *JESS: Java Expert System Shell* - WWW Internet site, Sandia Labs., Livermore - CA, USA. 1997. Web: http://herzberg.ca.sandia.gov/jess

[SAR 97] Sarle, Warren S. *Comp.ai.neural.nets FAQ* (Usenet Frequently Asked Questions - Neural Nets). Web: http://www.cis.ohio-state.edu/hypertext/faq/usenet/ai-faq/neural-nets/top.html . Ftp: ftp://ftp.sas.com/pub/neural/FAQ.html

[SCH 77] Schank, R. & Abelson, R. *Scripts, Plans, Goals and Understanding*. Lawrence Erlbaum Associates, Hillsdale. 1977.

[SCH 93] Schiffmann, W.; Joost, M. & Werner, R. *Comparison of Optimized Backpropagation Algorithms.* Proceedings of the European Symposium on Artificial Neural Networks, ESANN'93, Brussels, p.97-104, 1993. Web: http://www.uni-koblenz.de/~schiff/publications.html

[SCH 94] Schifmann, W.; Joost, M. & Werner, R. *Optimization of the Backpropagation Algorithm for Training Multilayer Perceptrons.* Technical Report, University of Koblenz, Deutschland. September 1995. Web: http://www.uni-koblenz.de/~schiff/publications.html

[SCH 96] Schmidt, William & Ling, Charles X. *A Decision-Tree Model of Balance Scale Development.* Machine Learning. Kluwer Academic Publishers, Boston, pp.1-30, 1996. Web: http://www.csd.uwo.ca/faculty/ling/

[SHA 90] Shavlik, Jude & Dietterich, Thomas (Eds). *Readings in Machine Learning*. Morgan Kaufmann Publishers, Inc. San Mateo, California, 1990.

[SHA 93] Shastri, L. & Ajjanagadde, V. *From Simple Associations to Systematic Reasoning*. Behavioral and Brain Sciences (BBS), Vol.16, No.3, pp.417-494. 1993. Web: http://http.icsi.berkeley.edu/~shastri/

[SHA 95] Shavlik, Jude W. *Learning by Symbolic and Neural Methods*. In: The Handbook of Brain Theory and Neural Networks. M. Arbib (Ed.), MIT Press, pp.533-537.1995.

[SHO 76] Shortliffe, E., *Computer Based Medical Consultations: MYCIN.* Elsevier Publ., 1976.

[SHU 91] Shultz, Thomas R. & Schmidt, William C. *A Cascade-Correlation Model of Balance Scale Phenomena.* Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society, pp. 635-640. Hillsdale, NJ: Erlbaum. 1991. Web: http://www.psych.mcgill.ca/labs/lnsc/html/Lab-Home.html (or Pub-cog-dev.html). Ftp: ftp://ego.psych.mcgill.ca/pub/shultz/balcog.ps.gz

[SHU 94a] Shultz, Thomas R.; Mareschal, D. & Schimidt, W. *Modeling Cognitive Development on Balance Scale Phenomena.* Machine Learning, Kluwer Academic Publishers, Boston. 16

(1994), pp. 57-86. Web: http://www.psych.mcgill.ca/labs/lnsc/html/Lab-Home.html (or Pub-cog-dev.html). Ftp: ftp://ego.psych.mcgill.ca/pub/shultz/balml.ps.gz

[SHU 94b] Shultz, Thomas R.; Buckingham, David & Oshima-Takane, Yuriko. *A Connectionnist Model of the Learning of Personal Pronouns in English.* In : S.J.Hanson, T.Petsche, M.Kearns & R.L.Rivest (Eds.). Computational Learning Theory and Natural Learning Systems. Vol.2: Intersection between theory and experiment (pp. 347-362). Cambridge, MA: MIT Press, 1994. Web: http://www.psych.mcgill.ca/labs/lnsc/html/Lab-Home.html (or Pub-cog-dev.html). Ftp: ftp://ego.psych.mcgill.ca/pub/shultz/pronouns.ps.gz

[SHU 96] Shultz, Thomas, R. Models of Cognitive Development. In: Penser L'Esprit: Des Sciences de la Cognition à une Philosophie Cognitive. V. Rialle et D. Fisette (Eds.). PUG - Presses Universitaires de Grenoble, 1996.

[SIE 76] Siegler, R.S. *Three Aspects of Cognitive Development.* Cognitive Psychology. No. 8, 1976.

[SIM 83] Simon, Hebert A. *Why should Machines Learn?* In: Machine Learning: An artificial intelligence approach (Vol.1), R. Michalski, J. Carbonell, T. Mitchell (Eds.). Mogan Kaufmann, San Mateo, CA - U.S.A. 1983.

[SIM 90] Simpson, Patrick. *Artificial Neural Systems: Foundations, Paradigms, Applications and Implementations*. Pergamon Press, 1990.

[SIM 93] Simon, Natalio. *Constructive Supervised Learning Algorithms for Artificial Neural Networks*. Master Thesis, Faculty of Electrical Engineering, Delft University of Technology. The Netherlands, June 1993. Ftp: ftp://archive.cis.ohio-state.edu/pub/neuroprose/

[SJO 91] Sjogaard, Steen. *A Conceptual Approach to Generalization in Dinamical Neural Networks*. Ph.D. Thesis, Computer Science Department - Aarhus University, Denmark. October 1991. Ftp: ftp://archive.cis.ohio-state.edu/pub/neuroprose/ (sjogaard.concept.ps.Z)

[STE 83] Stefik, M. et al. *Basic Concepts for Building Expert Systems*. In: Building Expert Systems - V.1. F. Hayes-Roth, D. Waterman and D. Lenat (Eds.). Addison-Wesley Publishing Company Inc., Reading, Massachusetts, U.S.A., 1983.

[STU 96] Stückelberg, M. & Hilario, M. *Declarative Heuristics for Neural Network Design.* In:. Proceedings of the ECAI'96 Workshop - Neural Networks & Structured Knowledge. Budapest, Hungary, August 1996.

[SUL 93] Sulzberger, S.; Tschichold-Gurman, N.; Vestli, S. *FUN: Optimization of Fuzzy Rule Based Systems Using Neural Networks*. IEEE Conference on Neural Networks, San Francisco, U.S.A. April 1993. Web: http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/fuzzy/systems/flie/

[SUN 91] Sun, Ron. Integrating Rules and Connectionism for Robust Reasoning: A Connectionist Architecture with Dual Representation. Ph.D. Thesis, Brandeis University, Waltham, MA. Technical Report CS-91-160. 1991.

[SUN 94] Sun, Ron & Bookman, Lawrence. (Eds.) *Computational Architectures Integrating Symbolic and Connectionist Processing*. Kluwer Academic Publishers, Boston. November 1994.

[SUN 95a] Sun, Ron. (Ed.) *Integrating Rules and Connectionism for Robust Commonsense Reasoning*. John Wiley and Sons, 1995.

[SUN 95b] Sun, Ron. *Robust Reasoning: Integrating Rule-Based and Similarity-Based Reasoning*. Artificial Intelligence Journal. June 1995. Web: http://www.cs.ua.edu/sun/

[SUN 97] Sun, Ron & Alexandre, Frederic. (Eds.)*Connectionist-Symbolic Integration*: *From Unified to Hybrid Approaches*. Lawrence Erlbaum Associates, 1997.

[SZI 95] Szilas, Nicolas. *Apprentissage dans les Réseaux Récurrents pour la Modélisation Mécanique et Etude de leurs Interactions avec l'Environnement*. Thèse de Doctorat en Sciences Cognitives, Laboratoire LIFIA/LEIBNIZ - IMAG, Grenoble - France, Décembre 1995.

[THR 91] Thrun, S. B. et al. *The Monk's Problem - A Performance Comparison of Different Learning Algorithm*. Carnegie Mellon University - CMU, Technical Report CMU-CS-91-197. December 1991. Web: http://www.cs.cmu.edu/~thrun/ . Ftp: ftp://archive.cis.ohio-state.edu/pub/neuroprose/

[TOU 89] Touretzky, D.S. *BoltzCONS: Dynamic Symbol Structures in a Connectionist Network*. Technical Report CMU-CS-89-182, Carnegie-Mellon University, U.S.A. August 1989.

[TOW 91] Towell, Geoffrey. *Symbolic Knowledge and Neural Networks: Insertion, Refinement and Extraction*. Ph.D. Thesis, Computer Science Dept., University of Wisconsin-Madison, U.S.A. 1991. Web: http://www.cs.wisc.edu/~shavlik/uwml.html . Ftp: ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/ (towell.thesis.*.ps)

[TOW 93] Towell, G. & Shavlik, J. *Extracting Refined Rules From Knowledge-Based Neural Nets.* Machine Learning, Kluwer Academic Publishers, Boston, pp.71-101, 13 (1993). Web: http://www.cs.wisc.edu/~shavlik/uwml.html . Ftp: ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/towell.mlj93.ps

[TOW 94] Towell, G. & Shavlik, J. *Knowledge-Based Artificial Neural Networks.* Artificial Intelligence Journal, 70, pp.119-165. Elsevier Publishing, 1994. Web: http://www.cs.wisc.edu/~shavlik/uwml.html . Ftp: ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/towell.aij93.ps

[UTG 88] Utgoff, Paul. *Perceptron Trees: A case study in hybrid concept representations*. In: Proceedings of the Seventh National Conference on Artificial Intelligence, St. Paul, MN. Morgan Kaufmann Publishing. 1988.

[UTG 89] Utgoff, Paul E. *Incremental Induction of Decision Trees*. In : Machine Learning, 4, pp.161-186. Kluwer Academic Publishers, Boston, MA - U.S.A. 1989.

[UTG 95] Utgoff, Paul E. *Decision Tree Induction Based on Efficient Tree Restructuring.* Technical Report 95-18, Dept. of Computer Science - Univ. of Massachusetts, USA. March 1995. Web: http://ml-www.cs.umass.edu/~utgoff/

[VAN 90] Van de Velde, Walter. *Incremental Induction of Topologically Minimal Trees.* In: Proceedings of the Seventh International Conference on Machine Learning - B. Porter, R. Mooney (Eds.). Morgan Kaufmann, San Mateo, CA - U.S.A. 1990.

[VES 93] Vestli, Sjur. *FLIE - Fuzzy-Logic Inference Engine*. Publié dans le WEB-Internet. Web: http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/fuzzy/systems/flie/ . Ftp: ftp://ftp.hiof.no/pub/Fuzzy/flie/ (fliefort.docu)

[WAT 97] Watson, Mark. *Intelligent Java Applications for the Internet and Intranets*. Morgan Kaufman Publishing. 1997.

[WER 74] Werbos, Paul J. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Ph.D. Thesis, Harvard University, Cambridge - MA. August 1974.

[WID 90] Widrow, Bernard & Lehr, M. *30 Years of Adaptive Neural Networks : Perceptron, Madaline, and Back-Propagation.* Proceedings of the IEEE, New York, Vol.78, N.9, pp.1415-1441. September 1990.

[ZIG 92] Zighed, A.; Auray, J. P.; Duru, G. *SIPINA: Méthode et Logiciel.* Lacassagne, 1992.

[ZIG 96b] Zighed, A. & Rakotomalala, R. *A Method for non Arborescent Induction Graphs.* Technical Report, Equipe ERIC - Université Lumière Lyon 2, 1996.

[ZIG 96c] Zighed, A.; Rakotomalala, R.; Rabaseda, S. *A Discretization Method of Continuous Attributes in Induction Graphs.* Proceedings of the Thirteenth European Meeting on Cybernetics and Systems Research, Vienna, pp.997-1002, Sept. 1996. Web: http://eric.univ-lyon2.fr/

7.1 BIBLIOGRAPHIE

** Liste des documents non cités dans le corps du mémoire - informations bibliographiques supplémentaires :*

[AJJ 91] Ajjanagadde, V. G. & Shastri, L. *Rules and Variables in Neural Nets*. Neural Computation, 3:121-134. 1991.

[ALL 93] Alliot, J. M. & Schiex, T.*Intelligence Artificielle et Informatique Theorique*. Editions CEPADUES, France. 1993.

[AMY 89] Amy, Bernard. *Contextual Cognitive Machine.* In : Advances in Cognitive Sciences. Vol.2. Guy Tiberghien (Ed). Ellis Horwood Publ., 1989.

[AND 95a] Andrews, Robert & Geva, Shlomo. *RULEX & CEBP Networks as the Basis for a Rule Refinement System*. In: Hybrid Problems, Hybrid Solutions. John Hallam (Ed.). 1995. Web: http://www.fit.qut.edu.au/~robert/

[AND 96a] Andrews, R.; Diederich, J.; Pop, E.; Tickle, A. *The Importance and Application of Symbolic Rule Extraction from Trained Artificial Neural Networks* (AI meets Artificial Insemination). Technical report QUTNRC-96-01-01, Neurocomputing Research Center, Queensland University of Technology, Brisbane - Australia. 1996. Web: http://www.fit.qut.edu.au/NRC/ . Ftp: ftp://ftp.fit.qut.edu.au/pub/NRC/tr/ps/ QUTNRC-96-01-01.ps.Z

[AND 96b] Andrews, R.; et all. *An Evaluation and Comparison of Techniques for Extracting and Refining Rules from Artificial Neural Networks*. Technical report QUTNRC-96-01-04, Neurocomputing Research Center, Queensland University of Technology, Brisbane - Australia. 1996. Web: http://www.fit.qut.edu.au/NRC/ . Ftp: ftp://ftp.fit.qut.edu.au/pub/NRC/tr/ps/ QUTNRC-96-01-04.ps.Z

[ARB 95] Arbib, Michael A. (Ed.)*The Handbook of Brain Theory and Neural Networks*. MIT Press, 1995.

[BAL 94] Bala, J.; De Jong, K.; Pachowicz, P. *Multistrategy Learning form Engineering Data by Integrating Inductive Generalization and Genetic Algorithms.* In: Machine Learning - A Multistrategy Approach, Volume IV. Chapter 18, R. Michalski and G. Tecuci (Eds.), Morgan Kaufmann Publishers, San Francisco. 1994.

[BAL 95] Balakrishnan, Karthik & Honavar, Vasant. *Evolutionary Design of Neural Architectures: A Preliminary Taxonomy and Guide to Literature*. Technical Report - Artificial Intelligence Research Group. Iowa State University - USA. January 1995. Web: http://www.cs.iastate.edu/~honavar/ . Ftp: ftp://archive.cis.ohio-state.edu/pub/neuroprose/

[BAR 91] Barnden, J. & Pollack, J. (Eds.) *High-Level Connectionist Models* - Advances in Connectionist and Neural Computation Theory. Ablex Publishing, Norwood, New Jersey. 1991.

[BOO 89] Booker, L.; Goldberg, D.; Holland, J. *Classifier Systems and Genetic Algorithms*. Artificial Intelligence 40(1-3). pp.235-282. September 1989.

[BOO 93] Bookman, L. & Sun, R. (Eds.) *Integrating Neural and Symbolic Processes.* Special Issue of Connection Science - n.5, 1993.

[BOZ 97a] Boz, Olcay. *Integrating Connectionism with Symbolism & Rule Integration and Extraction in Neural Networks*. (Ressource guide - Available through Internet - University of Lehigh). Web: http://www.lehigh.edu/~ob00/integrated.html and http://www.lehigh.edu/~ob00/integrated/references-new.html

[CLA 89] Clark, P. & Niblett, T. *The CN2 induction Algorithm*. Machine Learning, 3, pp.261-283, Kluwer Academic Publishers, Boston, MA - U.S.A. 1989.

[CLA 91] Clark, P. & Boswell, R. *Rule Induction with CN2: Some Recent Improvements*. In: Machine Learning - Proceedings of the Fifth European Conference (EWSL-91), Y. Kodratoff (Ed.), pp.151-163. Springer-Verlag, Berlin, 1991. Web: http://www.cs.utexas.edu/users/pclark/(papers/newcn.ps)

[CMU 95a] CMU AI Repository. *Fuzzy Logic & Fuzzy Systems*. Publié dans le WEB-Internet. Web: http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/fuzzy/0.html

[CMU 95b] CMU AI Repository. *Genetic Algorithms & Genetic Programming*. Publié dans le WEB-Internet. Web: http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/genetic/0.html

[CON 94] Condamin, Laurent. *Extraction et Intégration de Règles sur un Réseau Neuro-Mimétique: Application à l'Harmonisation de la Cotation à la Banque de France.* Thèse de Doctorat en Génie de Systèmes. Laboratoire MAS - ECP, Ecole Centrale de Paris, France. 1994.

[COX 94] Cox, Earl. *The Fuzzy Systems Handbook*. The Metus Systems Group - AP Professional. March 1994.

[CRA 94] Craven, M. W. *Using Sampling and Queries to Extract Rules from Trained Neural Network.* Machine Learning: Proc. of the 11th International Conf. of San Francisco. 1994. Web: http://www.cs.wisc.edu/~shavlik/uwml.html . Ftp: ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/craven.mlc94.ps

[CRA 96] Craven, Mark W. *Extracting Comprehensible Models from Trained Neural Networks.* Ph.D. Thesis, Computer Science Dept., University of Wisconsin-Madison, U.S.A. 1996. Web: http://www.cs.wisc.edu/~shavlik/uwml.html . Ftp: ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/craven.thesis.ps

[CUN 90] LeCun, Y.; Denker, J. S.; Solla, S.A. *Optimal Brain Damage* ; Advances in Neural Information Processing Systems 2, D.S. Touretzky ed., Morgan Kauffmann publishers, 1990.

[DIE 95] Dietterich, T. G.; Hild, H.; Bakiri, G. *A Comparison of ID3 and BackPropagation for English Text-to-Speech Mapping*. Machine Learning, 18(1), pp.51-80. Kluwer Academic Publishers, Boston, U.S.A. 1995.

[DOU 95] Dougherty, James; Kohavi, Ron; Sahami, Mehran. *Supervised and Unsupervised Discretization of Continuous Features.* In: Machine Learning - Proceedings of the Twelfth International Conference, A. Prieditis and S. Russell (Eds.), Morgan Kaufmann Publishers, San Francisco, CA. 1995.

[DUA 92] Duane, Darrel; Hintz, Kenneth; Spofford, Jason. *GANNET: Genetic Algorithm / Neural Network* (evolve ANNs using GA). Publié dans le WEB-Internet. Web: http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/genetic/ga/systems/gannet/

[DUR 89] Durbin, R. & Rumelhart, D. *Product Units : A Computationally Powerful and Biologically Plausiblle Extension to Backpropagation Networks.* Neural Computation, Vol. 1, 1989.

[FAY 92] Fayyad, Usama M. & Irani, Keki B. *On the Handling of Continuous-Valued Attributes in Decision Tree Generation*. Machine Learning, 8, pp.87-102. Kluwer Academic Publishers, Boston - U.S.A. 1992. Web: http://www-aig.jpl.nasa.gov/home/fayyad/home.html

[FAY 93] Fayyad, Usama M. & Irani, Keki B. *Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning.* Proceedings of the IJCAI'93. pp.1022-1027. 1993. Web: http://www-aig.jpl.nasa.gov/home/fayyad/home.html

[FIS 87] Fisher, Douglas H. *Knowledge Acquisition via Incremental Conceptual Clustering*. Machine Learning, 2(7), pp.139-172. Kluwer Academic Publishers, Boston, MA - U.S.A. 1987.

[FRA 93] Frasconi, P.; Maggini M.; Gori, M. & Soda, G. *Unified Integration of Explicit Knowledge and Learning by Examples in Recurrent Networks.* IEEE Transactions on Knowledge and Data Enginneering. 1993. Web: http://dsi.ing.unifi.it/neural/ . Ftp: ftp://ftp-dsi.ing.unifi.it/pub/tech-reports/kl-nets.ieee.kde.ps.Z

[FRA 95] Francesco, M.; Hilario, M.; Vargas, R.; Pellegrini, C. *Machine Learning at the Crossroads of Symbolic and Connectionist Research.* Technical Report UNIGE-AI-95-01, Genève, 1995. Ftp: ftp://cui.unige.ch/AI/

[FU 89] Fu, L. *Integration of Neural Heuristics into Knowledge-Based inference.* Connection Science 1(3): 325-339. 1989.

[GEN 90] Gendert, Dedre. *The Mechanisms of Analogical Learning*. In: Readings in Machine Learning, J. Shavlik and T. Dietterich (Eds). Morgan Kaufmann Publishers, Inc. San Mateo, California, 1990.

[GRU 94] Grumbach, Alain. *Cognition artificielle: Du reflexe ... `a la reflexion.* Addison Wesley, France, 1994.

[HAT 91] Haton, Jean-Paul et coll. *Le Raisonnement en Intelligence Artificielle : Techniques, Modèles et Architectures pour les Systèmes à Bases de Connaissances.* InterEditions, Paris, 1991.

[HAY 96a] Hayward, R.; Pop, E.; Diederich, J. *Rule Extraction from Cascade-2 Networks.* Technical report QUTNRC-96-01-02, Neurocomputing Research Center, Queensland University of Technology, Brisbane - Australia. 1995. Web: http://www.fit.qut.edu.au/NRC/ . Ftp: ftp://ftp.fit.qut.edu.au/pub/NRC/tr/ps/ QUTNRC-96-01-02.ps.Z

[HEI 97] Heitkoetter, J. & Beasley, D. (Eds.) *The Hitch-Hiker's Guide to Evolutionary Computation - Comp.ai.genetic FAQ* (Usenet Frequently Asked Questions - Neural Nets). Web: http://www.cis.ohio-state.edu/hypertext/faq/usenet/ai-faq/genetic/top.html or http://alife.santafe.edu/~joke/encore/WWW/

[HER 91] Hertz, J.; Krogh, A.; Palmer, R. *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City - CA. 1991.

[HIL 95a] Hilario, M.; Lallement, Y.; Alexandre, F. *Neurosymbolic Integration: Unified versus Hybrid Approaches*. European Symposium on Artificial Neural Networks - ESANN. Brussels, 1995. Ftp: ftp://ftp.loria.fr/pub/loria/rfia/publications/cortex or ftp://cui.unige.ch/AI/mix/ (esann-95.ps)

[HIL 95b] Hilario, Melanie. *An onverview of Strategies for Neurosymbolic Integration*. Proceedings of the IJCAI Workshop on Connectionnist-Symbolic Integration. Montreal, Canada, 1995. Ftp: ftp://cui.unige.ch/AI/

[HIL 96a] Hilario, M.; Orsier, B.; Rida, A.; Pellegrini, C. *A Knowledge-Based Approach to MLP Configuration.* IEEE International Conference on Neural Networks (ICNN'96), Washington, DC, June 1996. Web: http://cuiwww.unige.ch/AI-group/home.html . Ftp: ftp://cui.unige.ch/AI/

[HOE 91] Hoehfeld, M. & Fahlman, S. *Learning with Limited Numerical Precision Using the Cascade-Correlation Architecture*. Carnegie Mellon University - CMU. Computer Science Technical Report CMU-CS-91-130. May 1991. Web: http://www.cs.cmu.edu/Reports/index.html

[HOL 92] Holland, J. *Les Algorithmes Génétiques.* Pour la Science, Vol.179. Septembre 1992.

[HOL 94] Holsheimer, M. & Siebes, A. *Data Mining: The Search for Knowledge in Databases*. Technical Report CS-R9406, Computer Science / Dept. of Algorithms and Architectures, CWI Amsterdam, The Netherlands. 1994. Web: http://www.cwi.nl/cwi/projects/datamining.html

[JAN 97] Jang, J.-S. R. *Internet's Resources for Neuro-Fuzzy and Soft Computing*. Publié dans le WEB-Internet. Web: http:// www.cs.nthu.edu.tw/~jang/nfsc.html

[JOR 90] Jorrand, Philippe & Sgurev, V. (Eds.). *Artificial Intelligence IV: Methodology, Systems, Applications*. Proceedings of AIMSA'90 - Albena, Bulgaria. Amsterdam, APIA - Elsevier Science Publishers. September 1990.

[KAN 92] Kandel, A. & Langholz, G. (Eds.) *Hybrid Architectures for Intelligent Systems.* CRC Press, 1992.

[KAN 97] Kantrowitz, M.; Horstkotte, E.; Joslyn, C. *Comp.ai.fuzzy FAQ* (Usenet Frequently Asked Questions - Fuzzy Logic and Fuzzy Expert Systems). Web: http://www.cis.ohio-state.edu/hypertext/faq/usenet/fuzzy-logic/part1/faq.html or http://www.cs.cmu.edu/Web/Groups/AI/html/faqs/ai/fuzzy/part1/faq.html

[KAS 96] Kasabov, Nicola K. *Foundations of Neural Networks, Fuzzy Systems and Knowledge Engineering*. The MIT Press books, 1996.

[KLA 95] Klawonnm, Frank; Nauck, Detlef; Kruse, Rudolf. *Generating Rules from Data by Fuzzy and Neuro-Fuzzy Methods*. In: Proceedings of the Third German GI-Workshop "Fuzzy-Neuro Systeme'95". Darmstadt, Germany, November 1995. Web: http://www.cs.tu-bs.de/~nauk

[KOD 93] Kodratoff, Y.; Michalski, R. S.; Carbonell, J. G.; Mitchell, Tom M. *Apprentissage Symbolique : Une Approche de l'Intelligence Artificielle* (Volumes 1 et 2 - Version française du livre "Machine Learning : an artificial intelligence approach"). Editions CEPADUES, France. 1993.

[KOH 96] Kohavi, Ron & Sahami, Mehran. *Error-Based and Entropy-Based Discretization of Continuous Features.* Proceedings of the KDD-96 (Knowledge Discovery and Data Mining Conference), 1996. Web: http://robotics.stanford.edu/users/ronnyk/ronnyk-bib.html

[KOW 86] Kowalik, Janusz (Ed.) *Coupling Symbolic and Numerical Computing in Expert Systems*. Workshop on CSNCES - Bellevue, Washington. Elsevier Science Publishing. 1986.

[KOZ 94] Koza, John R. *Genetic Programming II : Automatic Discovery of Reusable Programs (Complex Adaptive Systems).* Bradford Books. 1994.

[KOZ 96] Koza, J.; Goldberg, D.; Fogel, D. (Eds.) *Genetic Programming 1996 : Proceedings of the First Annual Conference*. Stanford University - July 1996. MIT Press, 1996.

[KUR 96] Kurfeb, F.; Küchler, A.; Memmi, D.; Giacometti, A. (Eds.) *Neural Networks and Structured Knowledge*. Proceedings of the ECAI'96 Workshop. Budapest, Hungary, August 1996.

[LAN 95] Langley, P. & Morgan, M. B. (Eds.) *Elements of Machine Learning*. Morgan Kaufmann Publishing, 1995.

[LIU 95] Liu, Huan & Setiono, Rudy. *Chi2: Feature selection and discretization of numeric attributes*. Proceedings of the 7th IEEE International Conference on Tools with Artificial Intelligence (TAI'95), pp.388-391. Washington D.C., U.S.A. Nov.1995. Web: http://www.iscs.nus.sg/~liuh/

[LIU 96] Liu, Huan & Setiono, Rudy. *Dimensionality Reduction via Discretization.* Knowledge-Based Systems, Vol. 9, No. 1, Feb. 1996, pp. 67-72. Web: http://www.iscs.nus.sg/~liuh/

[MAC 92] Machado, R.J. & Rocha, A.F. *A hybrid Architecture for Fuzzy Connectionist Expert System*. In: Hybrid Architectures for Intelligent Systems. A. Kandel and G. Langholz (Eds.). CRC Press, Boca Raton - Florida, 1992.

[MIC 83] Michalski, R. S. *A Theory and Methodology of Inductive Learning*. In: Machine Learning - An Artificial Intelligence approach, R. Michalski, J. Carbonell, T. Mitchell (Eds.), Morgan Kaufmann Publishing Co., Montain View, CA - U.S.A. 1983.

[MIC 94] Michalski, R. & Tecuci, G. (Eds.) *Machine Learning - A Multistrategy Approach, Volume IV*. Morgan Kaufmann Publishers, San Francisco, CA - U.S.A. 1994.

[NAU 94] Nauck, Detlef. *A Fuzzy Perceptron as a Generic Model for Neuro-Fuzzy Approaches*. In: Proceedings of Fuzzy-Systeme'94, 2nd GI-Workshop, Munich, Siemens Corp., October 1994. Web: http://www.cs.tu-bs.de/~nauk

[NAU 95a] Nauck, Detlef & Kruse, Rudolf. *NEFCLASS: A Neuro-Fuzzy Approach for the Classification of Data*. In: Applied Computing 1995 - Proceedings of the 1995 ACM Symposium on Applied Computing, Nashville, February 1995. Web: http://www.cs.tu-bs.de/~nauk

[NAU 96] Nauck, Detlef ; Nauck, Ulrike ; Kruse, Rudolf. *Generating Classification Rules with the Neuro-Fuzzy System NEFCLASS.* Proceedings of the Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS'96). Berkeley, U.S.A., 1996. Web: http://www.cs.tu-bs.de/~nauck/

[NAU 96a] Nauck, Detlef; Nauck, Ulrike; Kruse, Rudolf. *Generating Classification Rules with the Neuro-Fuzzy System NEFCLASS.* In: Proceedings of Biennal Conf. of the North American Fuzzy Information Processing Society - NAFIPS'96. Berkeley, 1996. Web: http://www.cs.tu-bs.de/~nauk

[NAU 96b] Nauck, Detlef & Kruse, Rudolf. *Neuro-Fuzzy Classification with NEFCLASS*. In: Operational Research Proceedings 1995. Springer, Berlin, pp.294-299. 1996. Web: http://www.cs.tu-bs.de/~nauk

[NOU 96] Nouira, Rym & Fouet, Jean-Marc. *A Knowledge Based Tool for the Incremental Construction, Validation and Refinement of Large Knowledge Bases*. ECAI'96 - Verification & Validation Workshop. Budapest, Hungary. 1996.

[OPI 95] Opitz, D. W.; Shavlik, J. W. *Dynamically Adding Symbolically Meaningful Nodes to Knowledge-Based Neural Networks.* To appear in : Knowledge-Base Systems. Computer Science Dept., University of Wisconsin-Madison. 1995. Web: http://www.cs.wisc.edu/~shavlik/uwml.html . Ftp: ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/opitz.kbs95.ps

[ORS 95] Orsier, B. *Etude et Application de Systèmes Hybrides Neurosymboliques*. Thèse de Doctorat en Informatique, Lab. LIFIA - UJF, Grenoble, France, Mars 1995. Web: http://www-leibniz.imag.fr/RESEAUX/public.html . Ftp: ftp://ftp.imag.fr/pub/LEIBNIZ/RESEAUX-D-AUTOMATES/orsier.these.ps.gz

[OSO 91] Osório, Fernando S. *Un Étude sur la Reconnaissance Automatique d'Images de Caractères avec l'Utilisation des Réseaux Neuronaux,* (Texte en Portugais: Um Estudo sobre Reconhecimento Visual de Caracteres através de Redes Neurais). Memoire: Master of Science, CPGCC, UFRGS, Porto Alegre - Brazil. October 1991.

[OSO 93] Osório, Fernando S. *ADAnLIeNE : Un Modèle de Réseaux de Neurones Artificiels Utilisé pour la Reconnaissance d'Eléments Présents dans des Images ou Sons*, (Texte en Portugais: ADAnLIeNE: Um Modelo de Rede Neural Artificial para o Reconhecimento de Padrões em Imagens e Sons). Panel'93 - XIX Latin-American Conference on Computer Science, Buenos Aires, Argentina, August 1993.

[OSO 95b] Osório, Fernando S. & Amy, Bernard. *Apprentissage Automatique Constructif à Partir de Règles et d'Exemples.* Rapport Interne (Diffusion interne). Équipe Réseaux d'Automates. Lab. Leibniz - IMAG/INPG. Grenoble, France, Novembre, 1995. Web: http://www-leibniz.imag.fr/RESEAUX/osorio/articles/diret.html

[OSO 96a] Osorio, Fernando S. *INSS et la Robotique Autonome : expériences et résultats*. Exposé réalisé dans le cadre d'une réunion de l'équipe Réseaux. Laboratoire LEIBNIZ - IMAG / INPG, Grenoble - France. Octobre 1996.

[OSO 98] Osório, Fernando & Amy, Bernard. *INSS: an Hybrid System for Constructive Machine Learning*. Article soumis à NEURAP'98, special session "Knowledge-Based Applications of Artificial Neural Networks". Cette conférence aura lieu à Marseille en mars 1998.

[OSO 98a] Osório, Fernando & Amy, Bernard. *Constructive Machine Learning: A New Neuro-Symbolic Approach*. Article soumis à la publication dans un livre édité par Gilbert Ritschard - AIDRI. Cet article est une version plus longue de l'article publié dans AIDRI'97 (en anglais). 1998.

[PAR 94] Paradis, François. *Revue de formalismes pour la représentation de la connaissance.* Rapport de Recherche MRIM-SUR94-00, Laboratoire CLIPS (UJF-LGI), IMAG, Grenoble - France. Mars 1994. Web: http://www-clips.imag.fr/mrim/ . Ftp: ftp://ftp.imag.fr/pub/SIRI/publications/1994/paradis94a.ps.gz

[PAZ 92] Pazzani, M. & Kibler, D. *The Utility of Knowledge in Inductive Learning*. Machine Learning, 9(1), pp.57-94. Kluwer Academic Publishers, Boston - U.S.A. 1992.

[PEL 94] Pellegrini,C & Hilario, M. *A Knowledge-Based System For Neural Network Design And Analysis.* Proceedings of the SPP-IF Information Conference on Knowledge-Based Systems, pp. 103-106. Yverdon, Switzerland, December, 1994. Ftp: ftp://cui.unige.ch/AI/harem/spp-if-94.ps

[PFE 89] Pfeifer, R.; Schreter, Z.; Fogelman-Soulié, F.; Steels, L. (Eds.) *Connectionism in Perspective*. Elsevier Science Publishing. 1989.

[PRE 94a] Prechelt, Lutz. *A Study of Experimental Evaluations of Neural Network Learning Algorithms: Current Research Practice.* Technical Report, Universität Karlsruhe, Germany. TR-19/94. 1994. Web: http://wwwipd.ira.uka.de/~prechelt/ . Ftp: ftp://archive.cis.ohio-state.edu/pub/neuroprose/

[PRE 94b] Prechelt, Lutz. *PROBEN1 - A Set of Neural Network Benchmark Problems and Benchmarking Rules*. Technical Report 21/94 - Fac. fur Informatik, Univ. of Karlsruhe - Germany. September 1994. Web: http://wwwipd.ira.uka.de/~prechelt/ . Ftp: ftp://archive.cis.ohio-state.edu/pub/neuroprose/

[QUI 88] Quinlan, J. R. *An Empirical Comparison of Genetic and Decision-Tree Classifiers*. In: Processdings of the Fifth International Conference on Machine Learning, J. Laird (Ed.). Morgan Kaufmann, San Mateo, CA - U.S.A. 1988.

[QUI 90] Quinlan, J. R. *Learning Logical Definitions from Relations.* Machine Learning, 5(3), pp.239-266. Kluwer Academic Publishers, Boston, MA - U.S.A. 1990.

[REY 97b] Reyes, G.; Osório, F.; Amy, B. *Aprendizaje Automático de Reglas de Alto Nivel con Redes Neuronales de Orden Superior*. ENC'97 - Encuentro Nacional en Computation. Queretaro, Mexique. Septembre 1997.

[RUM 95] Rumelhart, D.; Durbin, D.; Golden, R.; Chauvin, Y. *Backpropagation: The Basic Theory*. In: Backpropagation: Theory, Architectures, and Applications. W. Chauvin, D. Rumelhart (Eds.), Hillsdale, NJ. pp.1-34, Lawrence Erlbaum Associates. 1995.

[SES 91] Sestito, S.; Dillon, T. *The Use of Sub-Symbolic Methods for the Automation of Knowledge Acquisition for Expert Systems and Their Applications* ; Proceedings of the 11th International Conference on Expert Systems and their Applications (AVIGNON'91), Avignon. 1991.

[SUN 92] Sun, Ron. *Connectionist Models of Rule-based Reasoning*. In: AISB Quarterly, Special Issue on Hybrid Systems, No. 79, pp. 21-24. 1992

[SUN 95c] Sun, Ron. *A Hybrid Connectionist Model for Integrating Reactions, Rules, and On-Line Learning.* Technical Report, Dept. of Computer Science, University of Alabama, USA. January 1995. Web: http://www.cs.ua.edu/sun/ . Ftp: ftp:// ftp.cs.ua.edu/pub/tech-reports/

[THR 96] Thrun, Sebastian B. *Explanation-Based Neural Network Learning: A Lifelong Learning Approach*. Kluwer Academic Publisher, 1996.

[TIC 95] Tickle, A.; Orlowski, M.; Diederich, J. *DEDEC: Decision Detection by Rule Extraction from Neural Networks*. Technical report QUTNRC-95-01-03, Neurocomputing Research Center, Queensland University of Technology, Brisbane - Australia. 1995. Web: http://www.fit.qut.edu.au/NRC/ . Ftp: ftp://ftp.fit.qut.edu.au/pub/NRC/tr/ps/ QUTNRC-95-01-03.ps.Z

[TIC 96] Tickle, A.; Orlowski, M.; Diederich, J. *DEDEC: A Methodology for Extracting Rules from Trained Artificial Neural Networks*. Technical report QUTNRC-96-01-05, Neurocomputing Research Center, Queensland University of Technology, Brisbane - Australia. 1996. Web: http://www.fit.qut.edu.au/NRC/ . Ftp: ftp://ftp.fit.qut.edu.au/pub/NRC/tr/ps/ QUTNRC-96-01-05.ps.Z

[TIR 95] Tirri, H. *Replacing the Pattern Matcher of an Expert System with a Neural Network.* In: Intelligent Hybrid Systems, S. Goonatilake and S. Khebbal (Eds.), John Wiley and Sons. 1995.

[VAF 94] Vafaie, H. & De Jong, K. *Improving a Rule Induction System using Genetic Algorithms*. In: Machine Learning - A Multistrategy Approach, Volume IV. Chapter 17, R. Michalski and G. Tecuci (Eds.), Morgan Kaufmann Publishers, San Francisco. 1994.

[VAN 97] Vanthienen, Jan & van Harmelen, Frank. *Proceedings of the European Symposium on the Validation and Verification of Knowledge Based Systems*. Leuven, Belgium. June 1997.

[WEI 91] Weiss, S. M. & Kulikowski, C. A. *Computer Systems that Learn : Classification and Prediction Methods form Statistics, Neural Nets, Machine Learning, and Expert Systems*. Morgan Kaufmann Publishing. 1991.

[WNE 94] Wnek, J. & Michalski, R. *Comparing Symbolic and Subsymbolic Learning : Three Studies*. In: Machine Learning - A Multistrategy Approach, Volume IV. Chapter 19, R. Michalski and G. Tecuci (Eds.), Morgan Kaufmann Publishers, San Francisco. 1994.

[ZIG 90] Zighed, A.; Auray, J. P.; Duru, G. *SIPINA: Une Méthode d'Evaliation*. Proceedings of the XXXth International Conference of the Applied Econometrics Association. Ancara, Turkey, 1990.

[ZIG 96a] Zighed, A.; Rakotomalala, R.; Louvet, S.; Schalk, C. *SIPINA For Windows : User Manual*. Technical Report, Equipe ERIC - Université Lumière Lyon 2, 1996.

[ZIG 97] Zighed, A.; Rakotomalala, R.; Feschet, F. *Optimal Multiple Intervals Discretization of Continuous Attributes for Supervised Learning*. In : Proceedings of the Third International Conference on Knowledge Discovery and Data Mining - KDD-97 (to appear), 1997.

8. ANNEXE A

Dans cette annexe, nous présentons un exemple des fichiers utilisés pour représenter les connaissances symboliques sur un problème.

A.1 - Module Symbolique : Fichier Xor.Clp (Clips)

```
(clear)
(reset)

(assert (cas 1 FALSE FALSE Out_False))
(assert (cas 2 TRUE  FALSE Out_True))
(assert (cas 3 FALSE TRUE  Out_True))
(assert (cas 4 TRUE  TRUE  Out_False))

(defrule x1-or-x2
  ?nfct <- (cas ?x ?x1 ?x2 ?Out)
  (test (or (eq ?x1 TRUE) (eq ?x2 TRUE)))
  =>
  (assert (X1_OR_X2 ?x TRUE))
)

(defrule x1-nor-x2
  (cas ?x ?x1 ?x2 ?Out)
  (test (not (or (eq ?x1 TRUE) (eq ?x2 TRUE))))
  =>
  (assert (X1_OR_X2 ?x FALSE))
)

(defrule x1-and-x2
  (cas ?x ?x1 ?x2 ?Out)
  (test (and (eq ?x1 TRUE) (eq ?x2 TRUE)))
  =>
  (assert (X1_AND_X2 ?x TRUE))
)

(defrule x1-nand-x2
  (cas ?x ?x1 ?x2 ?Out)
  (test (not (and (eq ?x1 TRUE) (eq ?x2 TRUE))))
  =>
  (assert (X1_AND_X2 ?x FALSE))
)

(defrule x1-xor-x2
  (X1_OR_X2  ?x ?v1)
  (X1_AND_X2 ?x ?v2)
  (test (and (eq ?v1 TRUE) (not (eq ?v2 TRUE))))
  =>
  (printout t "Cas: " ?x " Sortie: Vraie " crlf)
)
```

```
(defrule x1-nxor-x2
  (X1_OR_X2  ?x ?v1)
  (X1_AND_X2 ?x ?v2)
  (test (not (and (eq ?v1 TRUE) (not (eq ?v2 TRUE)))))
  =>
  (printout t "Cas: " ?x " Sortie: Fausse " crlf)
)
```

A.2 - Module Symbolique : Fichier Xor.Symb (NeuComp)

```
$Features:

X1:binary;
X2:binary.

$End_Feat.

$Rules:

X1_OR_X2  <- X1;
X1_OR_X2  <- X2;
X1_AND_X2 <- X1,X2;
XOR       <- X1_OR_X2,Not(X1_AND_X2).

$End_rules.

$End.

%%
%% Xor d'après la logique de Boole
%% Xor = (A Or B) And Not(A And B)
%%
%% Tableau
%%
%%  A  B   (A or B)  (A and B)  Not(A and B)  XOR
%%  0  0      0          0             1       0
%%  1  0      1          0             1       1
%%  0  1      1          0             1       1
%%  1  1      1          1             0       0
%%
```

9. ANNEXE B

Dans cette annexe, nous présentons les fichiers utilisés pour représenter les connaissances du module connexionnistes NeuSim, et aussi des exemples de chaque type de fichier utilisé par les modules NeuComp et NeuSim.

B.1 - Module NeuComp : Fichier des paramètres de compilation (*.ccfg)

*** Xor.ccfg:**

```
%
% >> Fichier de Configuration de Neucomp <<
%
% Options de Compilation :
%
$Flags [OG]
%
% Options:
%    D - Affiche les tableaux de compilation
%    G - Affiche la topologie du réseau (graphique créé par gnuplot)
%    C - Connecter tous les noeuds
%        (connexions intercouches / utiliser toutes les entrées spécifiées)
%    I - Attribuer des valeurs aléatoires à tous les noeuds
%        (créer seulement la topologie du réseau à partir des règles)
%    O - Créer les entrées dans le même ordre défini par '$Features'
%
$WF 8
%
% WF = Weights_Factor.
% Les valeurs attribuées aux poids sont proportionnelles à "Weights_Factor"
% Voir [Towell 1991] - "W Constant"
%
```

B.2 - Module NeuComp : Fichiers avec des règles symboliques (*.symb)

*** Xor.symb:**

```
    $Features:
     X1:binary;
     X2:binary.
    $End_Feat.
```

```
$Rules:
 X1_OR_X2  <- X1;
 X1_OR_X2  <- X2;
 X1_AND_X2 <- X1,X2;
 XOR       <- X1_OR_X2, Not(X1_AND_X2).
$End_rules.

$End.
```

*** Robot.symb:**

```
#define MAXVAL 1024.0
#define MINVAL 0.0

#define DETECTWALL 800
#define SENSIB     100
#define EXACT      5000

%
% Contrôleur TRES SIMPLE d'un robot autonome Khepera
%

$Features:
SSL : range : [MINVAL , MAXVAL];  % Capteur 0
SL  : range : [MINVAL , MAXVAL];  % Capteur 1
SFL : range : [MINVAL , MAXVAL];  % Capteur 2
SFR : range : [MINVAL , MAXVAL];  % Capteur 3
SR  : range : [MINVAL , MAXVAL];  % Capteur 4
SSR : range : [MINVAL , MAXVAL];  % Capteur 5
SBR : range : [MINVAL , MAXVAL];  % Capteur 6
SBL : range : [MINVAL , MAXVAL].  % Capteur 7
$End_Feat.

$Rules:

Front_Sensor <= GT(SFL, DETECTWALL, SENSIB);
Front_Sensor <= GT(SFR, DETECTWALL, SENSIB);
Front_Sensor <= GT(SL,  DETECTWALL, SENSIB);
Front_Sensor <= GT(SR,  DETECTWALL, SENSIB);

Left_Sensor  <= GT(SL,  DETECTWALL, SENSIB),
                GT(SL, SFL, EXACT),
                GT(SL, SFR, EXACT),
                GT(SL, SR,  EXACT),
                GT(SL, SSR, EXACT);

Left_Sensor  <= GT(SSL, DETECTWALL, SENSIB),
                GT(SSL, SFL, EXACT),
                GT(SSL, SFR, EXACT),
                GT(SSL, SR,  EXACT),
                GT(SSL, SSR, EXACT);

Right_Sensor  <= GT(SR,  DETECTWALL, SENSIB),
                 GT(SR, SFR, EXACT),
                 GT(SR, SFL, EXACT),
                 GT(SR, SL,  EXACT),
                 GT(SR, SSL, EXACT);
```

```
Right_Sensor  <= GT(SSR, DETECTWALL, SENSIB),
                 GT(SSR, SFR, EXACT),
                 GT(SSR, SFL, EXACT),
                 GT(SSR, SL,  EXACT),
                 GT(SSR, SSL, EXACT);

Turn_Left <= Front_Sensor, Not(Left_Sensor);
Forward   <= Not(Front_Sensor);
Turn_Right<= Front_Sensor, Not(Right_Sensor).

$End_rules.

$End.
```

*** Xor.rep: Rapport de la compilation**

```
Neucomp - Rules to Neural Net Compilator
~~~~~~~

Compilation Source: xor.symb
Compilation Flags : [OG]
Random Seed Used  : 881764573

=> Inputs:
   X1
   X2

=> Hiddens:
   X1_OR_X2 (T:1)
   X1_OR_X2#OR (T:1)
   X1_OR_X2#1 (T:0)
   X1_AND_X2 (T:1)

=> Outputs:
   XOR (T:0)

* Inputs not used:

# Statistics :
  - Inputs : 2
  - Hiddens: 4
  - Outputs: 1
  - Layers : 3
  + Total of rules : 4
  + Total of nodes : 7

>>> End of compilation
```

B.3 - Modules NeuComp et NeuSim : Fichiers avec la topologie du réseau (*.top)

*** Xor.top: (Créé par NeuComp)**

```
%
% ANN Created Automatically by NEUCOMP
%
$Network
2 1 3 5
$Connections
I 0 0 0 0
I 0 0 3 3
I 1 1 2 2
I 1 1 3 3
N 0 0 1 1
N 1 1 4 4
N 2 2 1 1
N 3 3 4 4
O 4 4
$Labels
0 X1_OR_X2 1
1 X1_OR_X2#OR 1
2 X1_OR_X2#1 0
3 X1_AND_X2 1
4 XOR 0
$Inputs
0 X1
1 X2
```

*** Xor.top: (Créé par CasCor)**

```
%
% ANN Created Automatically by NEUSIM V2.1R4
%

$Network
2 1 2 2
$Connections
I 0 0 0 1
I 1 1 0 1
N 1 1 0 0
O 0 0
$Layers
L 0 1 1
L 1 0 0
$Labels
0 N0001 99
1 Cand_3_0 2
$Inputs
0 I0001
1 I0002
$End
```

*** Description du format des fichier *.top :**

```
%%
%% Réseau créé par F. Osório - 05/94
%%
```

```
$Network
NEntrées NSorties NCouches NNeurones

$Connections
% Spécification des connexions d'entrée:
% Les entrées de "ex" jusqu'à "ey" sont connectées à "nx" jusqu'à "ny"
% Les entrées et les neurones sont numérotés à partir de zéro!
% Les entrées ne sont pas considérées dans le nombre de couches NCouches
% "ex" peut être égal à "ey", ainsi que, "nx" peut être égal à "ny"

I ex ey nx ny
I ex ey nx ny
...

% Spécifications des connexions des neurones:
% Les neurones de "ni" jusqu'à "nj" sont connectés à "nx" jusqu'à "ny"

N ni nj nx ny
N ni nj nx ny
...

% Spécification des neurones de sortie:
% Les neurones de "nx" jusqu'à "ny" sont des neurones de sortie

O nx ny
O nx ny
...

$Layers
% Couches du réseau
% Couche "cx" inclut les neurones de "nx" jusqu'à "ny"
% Cette partie est facultative

L cx nx ny
...

% Identificateurs des neurones (information symbolique)
% Cette partie est facultative

$Labels

NeuroneX "Nom_de_l'Unité" [Type]
NeuroneY "Nom_de_l'Unité" [Type]
...

% Partie facultative
% Type: 0 = Identificateur d'une règle compilé
%       1 = Identificateur ajouté par le compilateur (règle transformée)
%       2 = Identificateur crée par l'algorithme CasCor (ajout d'unité)
%       3 = Réseau créé automatiquement par NeuSim
%       4 = Réseau créé "à la main"
%      99 = Origine inconnue

$Inputs

EntréeX "Nom_de_l'Entrée"
EntréeY "Nom_de_l'Entrée"
...

$End
```

B.4 - Modules NeuComp et NeuSim : Fichiers avec les poids des connexions du réseau (*.wts)

*** Xor.wts: (Créé par NeuComp)**

```
2 1 3 5 Compiled
I 0 0 8.000000
B 0 0 0 -4.000000
N 0 1 8.000000
N 2 1 8.000000
B 1 0 0 -4.000000
I 1 2 8.000000
B 2 0 0 -4.000000
I 0 3 8.000000
I 1 3 8.000000
B 3 0 0 -12.000000
N 1 4 8.000000
N 3 4 -8.000000
B 4 0 0 -4.000000
End
```

*** Xor.wts: (Créé par CasCor)**

```
2 1 2 2 CasCor
I 0 0 -5.255434
I 0 1 -28.647793
I 1 0 -4.819084
I 1 1 -29.328611
B 0 0 0 7.717860
N 1 0 -24.472671
B 1 1 0 13.135694
End
```

*** Description du format des fichier *.wts :**

```
NEntrées  NSorties  NCouches  NNeurones  [Type_de_Topologie]
B  Neurone  Freeze_Flag  Type_FcAct  Biais
...
N  Départ_Neurone  Arrivé_Neurone  Valeur_du_Poids
...
I Départ_Entrée Arrivé_Neurone Valeur_du_Poids
...
End


"N", "B" et "I" peuvent s'alterner.
Freeze_Flag : Indique s'il est un neurone avec des poids gelés.
Type_FcAct : Indique le type de la fonction d'activation de l'unité. La
  valeur 0 représente une fonction du type sigmoïde asymétrique.
Biais : Valeur du 'biais' (seuil) de l'unité.
```

```
Type_de_Topology : C'est facultatif. Indique le type d'algorithme utilisé
  pour obtenir les poids des connexions. Types acceptés = { Std_Bkp, CasCor,
  QuickProp, Compiled, Unknown }
```

B.5 - Module NeuSim : Fichier avec les données d'apprentissage (*.Learn, *.Test)

*** Xor.learn:**

```
2 1 4 0
0 0 0
1 0 1
0 1 1
1 1 0
```

*** Coma13-Adt.test:**

```
13 1 354 0
0.0 0.720000 0.0 0.0 0.0 0.5 1.0 1.0 0.436782 0.515152 0.500000 0.0 1.0 0
0.0 0.666667 0.0 0.0 0.0 0.0 1.0 1.0 0.632184 0.212121 0.625000 1.0 0.0 1
0.0 0.773333 1.0 0.0 0.0 0.5 0.0 0.0 0.500000 0.454545 0.500000 0.0 0.0 1
0.0 0.766667 0.0 1.0 0.0 0.5 1.0 1.0 0.287356 0.651515 0.500000 1.0 1.0 1
0.0 0.760000 1.0 0.0 0.0 0.0 1.0 1.0 0.540230 0.272727 0.500000 0.0 1.0 0
1.0 0.813333 1.0 0.0 0.0 0.5 0.0 0.0 0.431034 0.424242 0.500000 0.0 0.0 0
0.0 0.733333 1.0 0.0 0.0 0.0 1.0 1.0 0.459770 0.545455 0.500000 1.0 1.0 0
1.0 0.753333 1.0 0.0 0.0 0.5 1.0 1.0 0.172414 0.242424 0.500000 0.0 0.0 1
0.0 0.766667 0.0 0.0 0.0 0.5 1.0 1.0 0.689655 0.545455 0.500000 0.0 1.0 1
0.0 0.786667 0.0 0.0 0.0 0.0 1.0 1.0 0.402299 0.272727 0.500000 0.0 0.0 1
0.0 0.786667 0.0 0.0 0.0 0.0 1.0 1.0 0.229885 0.621212 0.500000 0.0 1.0 0
0.0 0.800000 0.0 0.0 0.0 0.5 1.0 1.0 0.689655 0.469697 0.500000 0.0 1.0 0
0.0 0.720000 0.0 0.0 0.0 0.5 1.0 1.0 0.557471 0.151515 0.500000 1.0 0.0 0
1.0 0.666667 1.0 1.0 0.0 0.5 1.0 1.0 0.344828 0.621212 0.500000 0.0 1.0 1
1.0 0.733333 1.0 0.0 0.0 0.0 1.0 1.0 0.632184 0.348485 0.500000 0.0 0.0 0
...
0.0 0.800000 1.0 0.0 1.0 1.0 1.0 1.0 0.574713 0.621212 0.500000 0.0 0.0 0
1.0 0.800000 0.0 0.0 0.0 0.5 1.0 1.0 0.425287 0.272727 0.500000 0.0 1.0 1
0.0 0.786667 0.0 0.0 0.0 0.0 1.0 1.0 0.402299 0.318182 0.500000 1.0 0.0 1
```

*** Description du format des fichier *.learn/*.test :**

```
Nombre_d'Entrées Nombre_de_Sorties Nombre_d'Exemples Type_des_Données
E0 E1 E2 .... EN S0 S1 S2 ... SN
S0 S1 S2 .... SN S0 S1 S2 ... SN
...

Type_des_Données :
 0 -> Entrées / Sorties = Valeurs réelles (e.g. 0, 1, 15, 5.25, -0.2)
 1 -> Entrées = Valeurs Réelles / Sorties = Valeurs Binaires  (non disp.)
 2 -> Entrées = Valeurs Réelles / Sorties = Valeurs Binaires  (non disp.)
```

B.6 - Module NeuSim : Fichier avec les paramètres d'apprentissage (*.cfg)

*** Xor-bp.cfg: (Méthode Rétro-Propagation)**

```
%%
%% NEUSIM Configuration File - XOR with Back-Propagation learning
%%
Task      LT     % Learning + Final Test
MaxEpochs 2000
NInputs   2
NOutputs  1
BackProp  2      % Learning method: backpropagation with 2 hidden units
%
% Learning parameters
%
RndRange  1.0
Epsilon   0.1
Momentum  0.9
MaxErr    0.4    % Maximum output error
StopCrit  1      % Stop when 100% correct (learning data)
SPOffset  0.1    % Sigmoid-Prime-Offset
DataOrder 1      % Order of data presentation: randomly
%
% Report Flags
%
RepLevel  128    % Report flag
ErrLevel  0      % Error  flag
RepFreq   100    % File Output (# of epochs)
UserFreq  10     % Console Output (# of epochs)
UserRep   1      % Console flag
%%
%% End of CFG
%%
```

*** Xor-qp.cfg: (Méthode QuickProp)**

```
%%
%% NEUSIM Configuration File - XOR with QuickProp learning
%%
Task      LT            % Learning + Final Test
MaxEpochs 1500
NInputs   2
NOutputs  1
BackProp  2             % Learning method:
Learning  2             % Quick-propagation with 2 hidden units
%
% Learning parameters
%
RndRange  1.0
Epsilon   0.05
Momentum  0.0
MaxErr    0.3
StopCrit  1             % Stop when 100% correct (learning data)
```

```
SPOffset  0.1
EpLearn   1             % Quick-prop always use weights adjust by epochs
%
% Report
%
RepLevel  128
ErrLevel  0
RepFreq   50
UserFreq  10
UserRep   1
%%
%% End of CFG
%%
```

*** Xor-cc.cfg: (Méthode CasCor)**

```
%%
%% NEUSIM Configuration File - XOR with Cascade-Correlation learning
%%
Task      LTWS          % Learn + Final Test
MaxEpochs 1500
NInputs   2
NOutputs  1             % Start with no hidden units
Learning  1             % Learning method:
Cascor    8             %     Cascade-Correlation (8 candidate units)
%
% CasCor parameters
%
CCOutPat  16
CCOutECh  0.01
CCHidPat  16
CCHidECh  0.03
EpLearn   1             % CasCor use quick-prop (epoch oriented)
%
% Learning parameters
%
RndRange  0.001
Epsilon   0.0001
Momentum  0.0
MaxErr    0.1
StopCrit  1             % Stop when 100% correct (learning data)
SPOffset  0.1
%
% Report
%
RepLevel  195
ErrLevel  0
RepFreq   10
UserFreq  10
UserRep   1
%%
%% End of CFG
%%
```

*** Xor-inss.cfg: (Méthode 'Raffinement' = Réseau Compilé + Apprentissage avec CasCor)**

```
%%
%% NEUSIM Configuration File - XOR with compiled ANN + CasCor learning
%%
Task      PLT           % Load compiled ANN + Learn + Final Test
MaxEpochs 500
NInputs   2
NOutputs  1             % Start with the pre-compiled ANN hidden units...
Learning  1             % Learning method:
Cascor    8             %     Cascade-Correlation (8 candidate units)
%
% CasCor parameters
%
CCOutPat  8
CCOutECh  0.01
CCHidPat  8
CCHidECh  0.03
EpLearn   1             % CasCor use quick-prop (epoch oriented)
%
% Learning parameters
%
RndRange  0.1
Epsilon   0.35
Momentum  0.0
MaxErr    0.4
StopCrit  1             % Stop when 100% correct (learning data)
SPOffset  0.1
%
% Report
%
RepLevel  128
ErrLevel  0
RepFreq   50
UserFreq  10
UserRep   1
%%
%% End of CFG
%%
```

*** Xor-tst.cfg: (Test d'un réseau créé préalablement - par compilation ou par apprentissage)**

```
%%
%% NEUSIM Configuration File - Test XOR Compiled ANN
%%
Task      PT            % Read Weights + Test
MaxEpochs 1             % Without learning - Just testing
NInputs   2
NOutputs  1
MaxErr    0.4
RepLevel  128           % Report: summary of results
ErrLevel  0
RepFreq   1
UserFreq  1
UserRep   1
%% END of Cfg
```

* Xor.rep: Rapport de la simulation

```
NEUSIM START: 10.Sep.97 - 14:46:55 (Wed)

>> NEUSIM V2.1R4 - By F.Osorio <<
>> Simulation Report File      <<

<CFG > Startup File Prefix: xor-cc
<SEED> Random Seed Used: 881761615
<EXEC> Create Cascade-Correlation initial ANN structure
<EXEC> Initialize ANN topology/default values
<EXEC> Randomize network weights
<EXEC> Read learning data to memory
<EXEC> Read testing data to memory
<EXEC> Simulation of 10 CasCor learn epochs!
<STAT> Correct:   0.00% - Wrong Outs: 00004 - Acc.Error:   2.000000 - Epoch: 00010 (L)
<EXEC> Simulation of 10 CasCor learn epochs!
<EXEC> CasCor - NETWORK EXPANSION!
<EXEC> Simulation of 10 Cascor learn epochs! (Hidden Unit)
<EXEC> Simulation of 10 Cascor learn epochs! (Hidden Unit)
<EXEC> Simulation of 10 Cascor learn epochs! (Hidden Unit)
<EXEC> Simulation of 10 Cascor learn epochs! (Hidden Unit)
<STAT> Correct:   0.00% - Wrong Outs: 00004 - Acc.Error:   1.693244 - Epoch: 00060 (L)
<EXEC> Simulation of 10 CasCor learn epochs!
<STAT> Correct:   0.00% - Wrong Outs: 00004 - Acc.Error:   1.675589 - Epoch: 00070 (L)
<EXEC> Simulation of 10 CasCor learn epochs!
<STAT> Correct:   0.00% - Wrong Outs: 00004 - Acc.Error:   1.463466 - Epoch: 00080 (L)
<EXEC> Simulation of 10 CasCor learn epochs!
<STAT> Correct:  25.00% - Wrong Outs: 00003 - Acc.Error:   1.062275 - Epoch: 00090 (L)
<EXEC> Simulation of 10 CasCor learn epochs!
<STAT> Correct:  25.00% - Wrong Outs: 00003 - Acc.Error:   0.722562 - Epoch: 00100 (L)
<EXEC> Simulation of 10 CasCor learn epochs!
<STAT> Correct:  25.00% - Wrong Outs: 00003 - Acc.Error:   0.509407 - Epoch: 00110 (L)
<EXEC> Simulation of 10 CasCor learn epochs!
<STAT> Correct:  75.00% - Wrong Outs: 00001 - Acc.Error:   0.307144 - Epoch: 00120 (L)
<EXEC> Simulation of 10 CasCor learn epochs!
<STAT> Correct: 100.00% - Wrong Outs: 00000 - Acc.Error:   0.055482 - Epoch: 00125 (L)
<EXEC> Write ANN topology file
<EXEC> Write ANN weights file
<END >
<<<>>> Generalization - BestScore: 100.000000 - BestEpoch: 125 - NewUnits: 1
<<<>>> Test Epochs: 00001 - Learn Epochs: 00125
EXIT: 10.Sep.97 # 14:46:56 (Wed)
```

==> Informations affichées à l'écran pendant la simulation :

```
Epoch:00010 - Correct:   0.00% - AccErr:    2.000000 - MaxErr:   0.175133
Epoch:00017 - CasCor Network Expansion!
Epoch:00020 - Correlation: 0.000139 [5]
Epoch:00030 - Correlation: 0.000142 [3]
Epoch:00040 - Correlation: 1.293679 [5]
Epoch:00050 - Correlation: 1.427921 [0]
Epoch:00060 - Correct:   0.00% - AccErr:    1.693244 - MaxErr:   0.175016
Epoch:00070 - Correct:   0.00% - AccErr:    1.675589 - MaxErr:   0.180441
Epoch:00080 - Correct:   0.00% - AccErr:    1.463466 - MaxErr:   0.147155
Epoch:00090 - Correct:  25.00% - AccErr:    1.062275 - MaxErr:   0.140679
Epoch:00100 - Correct:  25.00% - AccErr:    0.722562 - MaxErr:   0.079148
Epoch:00110 - Correct:  25.00% - AccErr:    0.509407 - MaxErr:   0.048071
Epoch:00120 - Correct:  75.00% - AccErr:    0.307144 - MaxErr:   0.027080
Epoch:00125 - Correct: 100.00% - AccErr:    0.055482 - MaxErr:   0.015177
Stop: 100% correct in Learn data file...

SUCCESS at 125 Epochs with 1 Added units
Random Seed: 881761615

Test Epochs: 00001 - Learn Epochs: 00125
Generalization - BestScore: 100.000000 - BestEpoch: 125 - NewUnits: 1

>>> End of simulation
```

*** Description du format des fichier *.cfg :**

```
Fichier de configuration de Neusim - Options disponibles :

Task LTRWSBOPGIVE
    L=Apprentissage (Learn)
    T=Tester avec le fichier *.test à la fin de l'apprentissage (Test)
    G=Tester avec le fichier *.test après chaque époque d'apprentissage
      (Generalization)
    W=Enregistrer le fichier de topologie *.top (WriteTop)
    S=Enregistrer le fichier des poids des connexions(SaveWt)
    O=Enregistrer les réponses du réseau dans le fichier *.out (SaveOut)
    R=Créer un fichier selon la topologie du fichier *.top (ReadTop)
    P=Lire un réseau préalablement créé, fichiers *.top et *.wts (LoadW)
    I=Passer dans le mode interactif à la fin de la simulation (Interact)
    V=Afficher la topologie finale du réseau (Gnuplot visualiz.)
    E=Créer les fichiers data.ok et data.err (Error File)
    F=FalseAnswers (False True, False Non True - 1 Binary Output)

FilePref  Nom_du_fichier
    Nom des fichiers (nom.cfg, nom.rep, nom.learn, nom.test, etc)

NInputs   Entrées               ('int' x >= 1)
NOutputs  Sorties               ('int' x >= 1)

Learning  Méthode (0 = BackProp, 1 = Cascor, 2=Quick-prop)

BackProp  Unités_Cachées        ('int' x >= 1)
     Création automatique d'un réseau à trois couches

CasCor    Unités_Candidates     ('int' x >= 1)

Epsilon   Vitesse_d'Apprentissage ('float' x >= 0.0)
Momentum  Inertie                 ('float' x >= 0.0)

FunctActv Type_de_la_Fonction_d'Activation
    Valeurs: 0=AsymSigm[0,0.5], 1=SymSigm[-0.5,0.5], 2=SymSigm[-1/+1]
             3=AsymSigm+Temperature, 4=TangHip[-1/+1]

MaxErr    Erreur_acceptée          ('float' x >= 0.0)
    Valeur considérée pour l'estimation de la pourcentage de bonnes
    réponses (sortie_réseau est dans l'intervalle comprise entre
    sortie_désirée +/- MaxErr)

StopCrit  Critère_d'arrêt (0 = StopErr; 1 = 100% Learn; 2 = 100% Test)
StopErr   Arrêt_de_la_simulation  ('float' x >= 0.0)
    Si 'StopCrit' = 0 alors la simulation va s'arrêter
    quand l'erreur des sorties sera inférieure à StopErr

MaxEpochs Nombre_d'Epoques        ('int'   x >= 0)
    La simulation va s'arrêter après 'MaxEpochs' époques de simulation

RndRange  Valeurs_Aléatoires      ('float' x >= 0.0)
    Définition de la valeur maximale des valeurs aléatoires générées
   (poids initiaux du réseau)

RndSeed   Random_Seed             ('int'   x >= 0)
    Valeur de initialisation du générateur de valeurs aléatoires

DataOrder Ordre_des_données       (0=Sequential, 1=Random)
```

```
NetOutput Type_de_Sortie          (0=+/-MaxErr,  1=Binary, 2=Exact)

WtsDecay  Decay             ('float' x <= 0.0)
CrossEnt  Flag              (0=Std error fct., 1=Cross-entropy fct.)
SPOffset  Offset            ('float' x >= 0.0)
EpLearn   Flag              (0=adaptation des poids après chaque exemple,
                             1=adaptation des poids après une époques)
SigmTemp  Température       ('float')

CCOutPat  CasCor_Patience   ('int' x >= 1)
CCOutECh  CasCor_Epochs     ('float')
CCHidPat  CasCor_Patience   ('int' x >= 1)
CCHidECh CasCor_Epochs      ('float')
CCAddOut  Flag    (0=non, il ne modifie pas le réseau initial
                   1=oui, il ajoute une couche de sortie supplémentaire)

RepLevel, ErrLevel, RepFreq, UserFreq, UserRep  :   Spécifient  le  type
    de report de la simulation (console et fichier). Voir documentations
    sur le simulateur à propos des codes acceptées.
```

B.7 - Module NeuSim : Fichier avec les résultats de la simulation (*.out, data.ok, data.err)

*** Xor.out:**

```
0.000000
0.921465
0.947786
0.086538
```

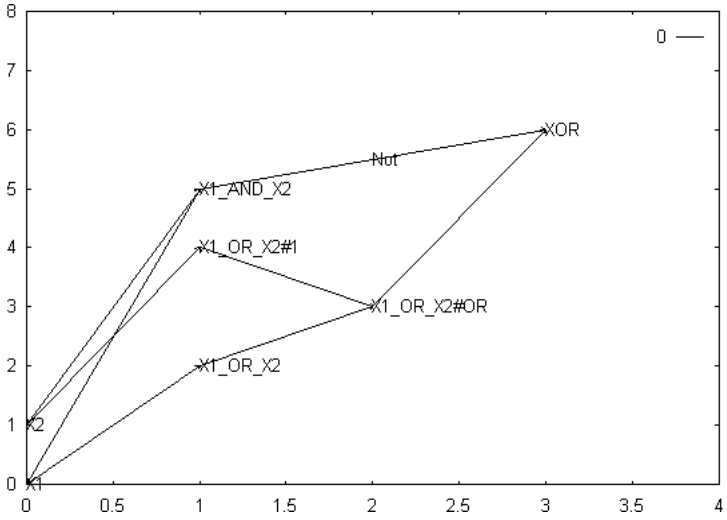
*** Data.ok: (nous avons intentionnellement changé un des 4 exemples)**

```
0.000000 0.000000 0.000000
1.000000 0.000000 1.000000
1.000000 1.000000 0.000000
```

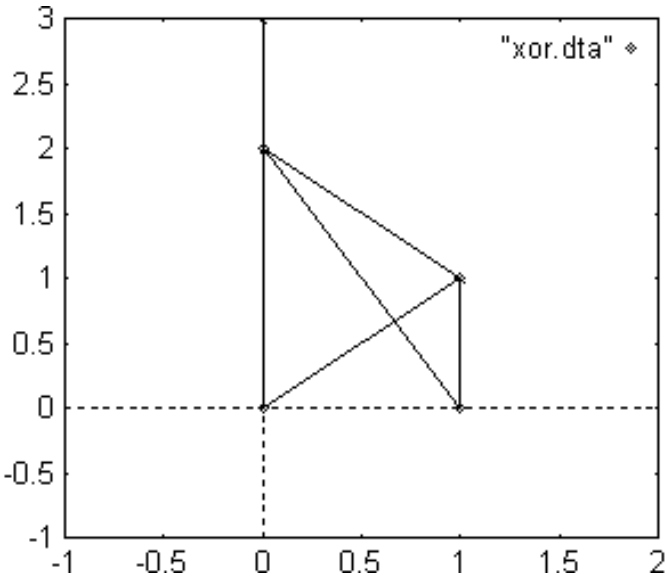
*** Data.err: (nous avons intentionnellement changé un des 4 exemples)**

```
0.000000 1.000000 0.000000 # 0.992506
```

B.8 - Modules NeuComp et NeuSim : Affichage graphique de la topologie des réseaux (*.gnup, *.gnuplot)



XOR - Réseau compilé



XOR - Réseau créé par apprentissage (CasCor)

10. GLOSSAIRE ET ABREVIATIONS

ADALINE, MADALINE - Type d'unités (neurones) qui utilisent une méthode d'apprentissage par minimisation de l'erreur - règle LMS. Principal responsable : B. Widrow. (Anglais : Adaptive Linear Element)

AG/GA - Algorithmes Génétiques (Anglais : *Genetic Algorithms* - G. A.)

ANN - Réseaux de Neurones Artificiels (Anglais : *Artificial Neural Networks* - A.N.N.)

AQ-Learning - Méthodes d'apprentissage par induction. Il existe toute une famille d'algorithmes basés sur l'algorithme AQ développé par Michalski. Les méthodes AQ incluent des versions spécifiques tels que: AQ-14 NT (*Noise Tolerant Learning*), AQ-15 GA (*Genetic Algorithm Attribute Selection*), AQ-15 FCLS (*Flexible Concept Learning*), AQ-17 (*Multistrategy Constructive Learning System*), AQ-17 HCI (*Hypothesis-Driven Constructive Induction*) et AQ-17 DCI (*Data-Driven Constructive Induction*). Principal responsable : R. Michalski.

Arbres de Décision / Arbres d'Induction - Méthode d'apprentissage par induction à partir de connaissances empiriques. L'algorithme le plus connu qu'implemente ce type de méthode est l'ID3. (Anglais : Decision Trees)

ARN2 - Modèle de réseaux connexionnistes à base de prototypes. Devéloppé au Lab. LIFIA, IMAG par l'Equipe Réseaux d'Automates. Principaux responsables : B. Amy, A. Giacometti et A. Azcarraga.

Assitant Professional - Méthode d'apprentissage utilisée pour la construction d'arbres de décision basée sur l'algorithme ID3 de Quinlan. Principaux responsables : B. Cestnik, I. Kononenko et I. Bratko.

Back-Propagation - Méthode d'apprentissage supervisé des réseaux connexionnistes du type PMC basée sur la minimisation de l'erreur entre la sortie du réseau et un résultat désiré. Ceci est obtenu par *descente de gradient* sur une surface d'erreur et par la rétro-propagation de cet erreur aux couches supérieures du réseau.

C4.5 - Système de construction d'arbres de décision. Principal responsable : R. Quinlan.

CART - Méthode d'apprentissage par induction, construction d'arbres de décision. (Anglais : *Classification And Regression Trees* - CART). Principal responsable : L. Breiman.

Cascade-Correlation / CASCOR - Méthode d'apprentissage supervisé de construction incrémentale des réseaux connexionnistes de type PMC, basée sur le calcul de la corrélation. Méthode développée par S. Fahlman et C. Lebière (Cascade-Correlation Learning Algorithm). Principal responsable : S. Fahlman.

CBR - Raisonnement fondé sur des cas (Anglais : *Case-Based Reasoning* - C.B.R.)

CLASSIT - Algorithme d'apprentissage basé sur une méthode de "*conceptual clustering*". Travaille uniquement sur des attributs numériques. Principaux responsables : J. Gennari, P. Langley et D. Fisher.

CLASSWEB - Combination des algorithmes COBWEB et CLASSIT dans un seul système. Principaux responsables : J. Kreuziger, R. Hamann et W. Wenzel.

CLIPS - Langage utilisé pour le dévéloppement de systèmes experts (*C Language Integrated Production System*). Devéloppé par la NASA.

CN2 - Méthode d'induction de règles de classification à partir d'exemples, qui s'utilise de l'entropie comme heuristique de recherche. CN2 combine les méthodes ID3 et AQ-Learning. Principaux responsables : P. Clark et T. Niblett.

COBWEB - Algorithme d'apprentissage non-supervisé basé sur une méthode de "*conceptual clustering*" (création de prototypes). Travaille uniquement sur des attributs nominaux. Principal responsable : D. Fisher.

DENDRAL - Systèmes Experts très connu utilisé pour l’identification de structures moléculaires

ECLIPSE - Langage utilisé pour le développement de systèmes experts. Utilise un moteur d'inférence par chaînage avant et chaînage arrière. Développé par The Haley Enterprise (Paul Haley).

ECOBWEB - Algorithme d'apprentissage non-supervisé basé sur une méthode de "*conceptual clustering*" du type de COBWEB. Principaux responsables : Y. Reich et D. Fisher.

FOCL - Système d'apprentissage automatique basé sur le programme FOIL de Quinlan. Il possède un module d'extension de FOIL avec un apprentissage basé sur l'explication (EBL). Principaux responsables : M. Pazzani et D. Kibler.

FOIL - Méthode d'apprentissage pour induction dont le but est l'obtention de règles du type de la langage Prolog (clauses de Horn). C'est une méthode du type ILP. Principal responsable : J. Quinlan.

GP - Programmation Génétique (Anglais : *Genetic Programming* - G.P.)

GPS - Système de résolution général de problèmes (Anglais : *General Problem Solvers* - G.P.S.)

IA - Intelligence Artificielle (Anglais : *Artificial Intelligence* - A.I.)

IAD - Intelligence Artificielle Distribuée

ID3 - Système de construction d'arbres de décision. Principal responsable : R. Quinlan.

ID5R - Système de construction incrémental d'arbres de décision dérivé d'ID3. Principal responsable : P. Utgoff.

IDL - Méthode incrémentale d'induction d'arbres de décision dérivée d'ID3et d'ID5R. Principal responsable : W. Van de Welde.

IDT - Voir les arbres de décision. (Anglais : Induction Decision Trees)

ILP - Programmation Logique Inductive. Méthode de apprentissage qui a pour but l'obtention d'une base de règles du type de celles utilisées dans le langage Prolog. Exemple de systèmes basés sur l'ILP: ProGol, Prolog-EBG, FOIL, mFOIL, FOCL. (Anglais : *Inductive Logic Programming* - I.L.P.)

INSS - *Incremental Neuro-Symbolic System*. Système hybride neuro-symbolique d'acquisition incrémental de connaissances. Utilise un réseau du type P.M.C. avec l'algorithme *Cascade-Correlation*. Devéloppé au Lab. Leibniz (LIFIA), IMAG par l'Equipe Réseaux d'Automates. Principaux responsables : B. Amy et F. Osório.

ITI - Méthode de construction incrémentale d'arbres de décision dérivée d'ID3 et d'ID5R. Principal responsable : P. Utgoff (Anglais : Incremental Tree Induction - I.T.I.)

KBANN - Knowledge-Based Artificial Neural Networks. Système hybride neuro-symbolique devéloppé à l'University of Wisconsin-Madison, par le équipe de 'machine

learning' de Jude Shavlik. Utilise un réseau du type P.M.C. avec l'algorithme de *Retro-Propagation*. Principaux responsables : J. Shavlik et G. Towell.

KBS - Systèmes à bases de connaissances (Anglais : *Knowledge-Based Systems* - K.B.S.)

KDD - Systèmes intelligents pour l'exploration de données et des connaissances contenues dans ces données. Ils sont usuellement dotés des méthodes d'analyse de données, d'apprentissage automatique et d'interprétation de données. (Anglais : *Knowledge Data Discovery = Data Mining*)

LIFIA - Laboratoire d'Informatique Fondamentale et d'Intelligence Artificielle, IMAG, Grenoble. Le laboratoire LIFIA a disparu en 1995 et certaines de ses équipes de recherche constituent actuellement le Laboratoire Leibniz.

MC - Module Connexionniste. Partie d'un système hybride qui est reponsable pour le traitement connexionniste.

mFOIL - Méthode d'apprentissage automatique du type ILP basée sur l'algorithme FOIL. Principal responsable : S. Dzeroski.

ML - Apprentissage Automatique (Anglais : *Machine Learning* - M.L.)

Monk's Problem - Collection de bases de donnés pour l'évaluation de méthodes d'apprentissage automatique (*benchmark*). Principal responsable : S. Thrun.

MS - Module symbolique. Partie d'un système hybride qui est reponsable pour les manipulations symboliques.

MYCIN - Système Expert très connu utilisé pour le diagnostic de maladies infectieuses sanguines.

NESSY3L - NEuroSymbolic SYstem with 3 Levels. Système hybride neuro-symbolique composé de trois niveaux. Devéloppé au Lab. Leibniz (LIFIA), IMAG par l'Equipe Réseaux d'Automates. Principaux responsables : B. Amy et B. Orsier.

OPS5 - Langage utilisé pour le développement de systèmes experts. Utilise un moteur d'inférence par chaînage avant basé sur l'algorithme Rete. Ce langage est à l'origine des langages comme CLIPS et ECLIPSE. Développé à Carnegie Mellon Univ. (CMU).

PERCEPTRON - Type d'unité des réseaux de neurones artificiels. Les Perceptrons sont une des premières propositions d'un modèle de neurone artificiel. Principal responsable : F. Rosenblatt.

PMC - Perceptron multi-couches (Anglais : *Multi-Layer Perceptron* - M.L.P.)

PRISM - Méthode d'apprentissage utilisée pour la construction d'arbres de décision basée sur l'algorithme ID3 de Quinlan. Il se distingue d'ID3 par le type de méthode employé pour la sélection des attributs discriminants. Principal responsable : J. Cendrowska.

PROBEN1 - Collection de bases de donnés pour l'évaluation de méthodes d'apprentissage automatique (*benchmark*). Principal responsable : L. Prechelt.

ProBis - Système hybride CBR-RNA, avec l'intégration d'un réseau connexionniste et d'un méthode de raisonnement fondé sur des cas. Devéloppé au Lab. Leibniz (LIFIA), IMAG par l'Equipe Réseaux d'Automates. Principaux responsables : B. Amy et M. Malek.

QuickProp - Méthode d'apprentissage supervisée des réseaux connexionnistes du type PMC basée sur la descente de gradient sur une surface d'erreur. C'est une méthode perfectionné de la Rétro-Propagation (plus performante dans la descente de la surface d'erreur) crée par S. Fahlman. Il s'utilise d'une méthode de deuxième ordre basée sur l'approximation de la dérivé seconde.

Rétro-Propagation - Méthode d'apprentissage supervisé des réseaux connexionnistes de type PMC basée sur la rétro-propagation de l'erreur de sortie. Voir : Back-Propagation. (Anglais : *Back-Propagation*)

RNA - Réseaux de Neurones Artificiels (Anglais : *Artificial Neural Networks* - A.N.N.)

SE - Systèmes Experts (Anglais : *Expert Systems*)

SHNS - Systèmes Hybrides Neuro-Symboliques (Anglais : *Hybrid Neuro-Symbolic Systems*)

SIPINA - Système d'apprentissage automatique pour la construction de graphes induction. Principal responsable : A. Zighed.

SMA - Systèmes Multi-Agents

SYNHESYS - Système hybride neuro-symbolique devéloppé au Lab. LIFIA, IMAG par l'Equipe Réseaux d'Automates. Utilise un réseau du type ARN2 à prototypes. Principaux responsables : B. Amy et A. Giacometti.

TDIDT - Induction d'arbres de décision par construction progressive. (Anglais : Top Down Induction of Decision Trees - T.D.I.D.T.)