Etude et application de systèmes hybrides neurosymboliques

Bruno Orsier

THÈSE

présentée à

L'Université Joseph Fourier
Grenoble I

pour obtenir le titre de
Docteur de l'Université Joseph Fourier
(Arrêtés ministériels du 5 juillet 1984 et du 30 mars 1992)

Spécialité
Informatique

par
Bruno Orsier

Étude et application de systèmes hybrides neurosymboliques

Soutenue le 23 mars 1995 devant le jury composé de :

MM.	Alain	Grumbach	Rapporteurs
	Christian	Pellegrini	
	Philippe	Cinquin	Examinateurs
	Philippe	Liebaert	
	Bernard	Amy	
	François	Rechenmann	Directeur

Thèse préparée au sein du laboratoire LIFIA/IMAG

2

Remerciements

Je tiens à remercier Monsieur Alain GRUMBACH, professeur à l'ENST, et Monsieur Christian PELLEGRINI, professeur à l'université de Genève, d'avoir accepté d'être rapporteurs sur cette thèse, malgré leurs emplois du temps chargés. Mes remerciements vont également à Monsieur Philippe CINQUIN, professeur à l'université Joseph Fourier, et à Monsieur Philippe LIEBAERT, adjoint au chef du groupe « Informatique et Robotique » du service de recherche de la DRET, qui ont accepté d'être examinateurs.

Je dois beaucoup à mes deux directeurs de thèse, Messieurs François RECHENMANN et Bernard AMY, qui m'ont permis de réaliser ce travail dans le cadre d'une coopération entre leurs équipes de recherches respectives. Tous deux m'ont laissé une grande liberté tout en me faisant profiter de leur expérience, et leurs conseils ont notamment permis d'améliorer de nombreux points du manuscrit. Enfin, ils ont toujours mis à ma disposition tous les moyens nécessaires à l'accomplissement de mon travail.

Je voudrais également remercier Monsieur Philippe JORRAND, directeur du LIFIA, pour m'avoir accueilli dans ce laboratoire. Les membres du personnel administratif du LIFIA (Laurence, Danièle, Mirella, Claudia, Claude, Lionel, Henri et Françoise) méritent d'être particulièrement remerciés pour leur gentillesse et leur disponibilité.

Étant donné que j'ai travaillé sur plusieurs applications des systèmes hybrides, de nombreux experts m'ont apporté leur concours :

- Régine BASSET, Patrick GENDRE, Aude VERNAT, du CNET, ont consacré beaucoup de leur temps à participer à une étude de faisabilité d'un système hybride neurosymbolique pour modéliser l'expertise en test paramétrique ; Arnaud GIACOMETTI a également participé à plusieurs des réunions relatives à ce travail ;
- Didier JOSSELIN et Pierre DUMOLARD, du Laboratoire de la montagne alpine, ainsi que Claude JANIN, de la Chambre d'Agriculture de l'Isère, ont fourni des données, de l'expertise et du temps, pour explorer les possibilités d'application des systèmes hybrides en géographie ; de plus, Monsieur Bruno CRÉMILLEUX, de l'université de Caen, nous a aimablement fourni une version de ARBRE, son logiciel d'induction d'arbres de décision ;
- Patrick REIGNIER et Jim CROWLEY, du LIFIA, m'ont permis d'utiliser et modifier leur simulateur de robot MOLUSC, et m'ont donné de précieux conseils.

M. Éric Gauthier a travaillé sur l'utilisation de réseaux récurrents dans la partie perception de mon application NESSY3L/ROB, dans le cadre d'un stage de Magistère.

Je voudrais remercier tout particulièrement un expert en systèmes hybrides : Arnaud Giacometti. Arnaud s'est prêté au jeu difficile de l'étude critique de son système SYNHESYS, et a accepté de corriger des versions très préliminaires des chapitres 3 et 4. De nombreuses discussions avec lui m'ont permis d'approfondir ma connaissance de SYNHESYS. Iréna Iordanova, Vincent Rialle et Arno Jacobsen m'ont également fait profiter de leurs réflexions sur SYNHESYS et les systèmes hybrides.

Abderrahim Labbi m'a souvent expliqué des aspects mathématiques des réseaux de neurones, et les trois années fort sympathiques durant lesquelles j'ai partagé son bureau ne sont certainement pas étrangères à l'emploi de ses réseaux versatiles dans le chapitre 5.

Dans le cadre du projet européen MIX, Mélanie Hilario, du Centre Universitaire d'Informatique de l'université de Genève, et Yannick Lallement, du CRIN-INRIA-CNRS, m'ont été d'une aide précieuse, grâce à nos nombreuses discussions électroniques, qui représentent sûrement plusieurs millions d'octets. Ces discussions ont notamment permis de clarifier plusieurs points du chapitre 2.

Outre les membres du jury, plusieurs personnes m'ont fait l'amitié de lire, commenter, critiquer tout ou partie du manuscrit : Elsa Collin, Alex Friedmann, Arnaud Giacometti, Didier Josselin, Nicolas Szilas, Danielle Ziebelin. Je comptais remercier ici Patrick Reignier, mais il a réussi à partir à l'étranger avant que je ne puisse lui donner de la lecture.

J'ai trouvé une excellente ambiance dans les deux équipes de recherche auxquelles j'ai appartenu, « Réseaux d'Automates » et « SHERPA », et je remercie leurs membres respectifs pour les bons moments passés ensemble, en particulier autour d'un café (plutôt avec certains membres de SHERPA : Nathalie, Cécile, Isabelle, Jean-Marc, Alain, Jérôme, Pierre, Florence, Jean-Yves, Olivier, Nina, Jutta) ou d'une bière (plutôt certains membres de Réseaux d'Automates : Nicolas, Maria, Abdel, Fernando, Sophie, Arnaud, et d'autres assoiffé(e)s).

Enfin, je voudrais remercier ma femme Elisabeth pour son constant soutien durant ces trois années d'un travail pas toujours compatible avec une vie de couple.

Table des matières

Introduction générale

En résolution de problèmes, comme dans d'autres secteurs de l'informatique, et des sciences en général, réaliser des systèmes hybrides est une démarche très courante. Lorsque deux techniques, ou deux méthodes, ou deux approches entrent quelque peu en compétition, il n'est pas rare que quelqu'un propose de combiner les points forts de chacune d'entre elles. Cela permet d'obtenir, à peu de frais, des performances plus élevées ou un champ d'application plus large.

Dans plusieurs domaines (Sciences Cognitives, informatique), deux grandes approches sont plus ou moins en compétition :

- l'approche analytique et formelle, disons symbolique pour simplifier, qui cherche à modéliser explicitement des processus de résolution, et emploie pour cela divers types de systèmes à base de connaissances, souvent appelés « systèmes symboliques » ;
- l'approche connexionniste, qui met plutôt l'accent sur l'utilisation d'exemples de résolution, et qui utilise des réseaux d'automates simples, appelés neurones artificiels, d'inspiration neurobiologique. Cette approche a connu une éclipse durant les années 70, mais suscite un vif intérêt depuis le début des années 80.

Trois visions différentes des réseaux de neurones artificiels coexistent et coopèrent :

- les réseaux de neurones artificiels sont des outils de l'ingénieur, permettant d'effectuer des tâches de approximation, classification, regroupement, optimisation, etc.,
- les réseaux sont des outils de modélisation neurobiologique,
- les réseaux sont des outils de modélisation cognitive.

Les approches symbolique et connexionniste sont en compétition du point de vue informatique, d'une part parce que dans certains cas elles visent à accomplir le même type de tâches (notamment la classification), et d'autre part, parce que leurs domaines d'application sont souvent les mêmes : vision, robotique, traitement des langues, diagnostic. Mais elles sont surtout en compétition comme meilleur candidat à la modélisation des processus cognitifs.

Bien qu'il existe des inconditionnels de chaque approche, certains chercheurs ont proposé de les combiner pour réaliser des systèmes hybrides neurosymboliques (ou encore symboli-connexionnistes), que nous noterons SHNS.

Depuis quelques années, les réalisations de SHNS foisonnent, comme en témoignent

l'organisation de plusieurs « workshops » [Sun et Bookman 1992, Hilario 1994], la publication de numéros spécialisés de revues [AISB 1992, Bookman et Sun 1993], et de livres [Kandel et Langholz 1992, Medsker 1994]. Un projet Esprit est actuellement consacré à l'étude et aux applications des SHNS : le projet MIX (*Modular Integration of Symbolic and Connectionnist Processing in Knowledge-Based Systems*), qui se terminera en 1997. Ce projet regroupe notre laboratoire d'accueil, le LIFIA/IMAG, le CRIN/CNRS/Inria Lorraine, le CUI (Université de Genève), l'Université Polytechnique de Madrid, l'Université Technique de Munich, et un partenaire industriel allemand, Kratzer.

Ce foisonnement justifie notre première contribution, qui est de tenter de présenter une vision structurée de ce vaste domaine des SHNS.

Toujours en raison du grand nombre de systèmes existants, nous avons choisi d'appliquer, dans un souci d'évaluation et de validation, un SHNS existant, SYNHESYS, et cela dans deux domaines : la micro-électronique et la géographie.

Enfin, nous avons proposé une nouvelle architecture de SHNS ainsi qu'un domaine d'application approprié, la robotique mobile. Nous avons réalisé une première version de cette architecture.

Le mémoire est organisé de la manière suivante :

- le chapitre 1 est consacré à une description générale des deux composants d'un système neurosymbolique, mais insiste plus particulièrement sur les réseaux de neurones artificiels dont les classes d'application restent parfois mystérieuses pour les non-spécialistes. Dans le même ordre d'idées, ce chapitre tente de clarifier les relations entre réseaux de neurones artificiels et d'autres méthodes non-neuronales. De plus, par souci pédagogique, nous développons une rapide analogie avec un système hybride pour lequel existent des moyens concrets d'évaluation : la voiture à double moteur électrique et essence ;
- le chapitre 2 est consacré à un état de l'art critique permettant de situer les systèmes hybrides neurosymboliques : nous montrons que ces systèmes hybrides sont un cas particulier de recherches plus générales visant à intégrer les meilleures caractéristiques des paradigmes symboliques et connexionnistes. À l'issue de ce chapitre, nous posons deux problèmes importants que nous résolvons dans les chapitres suivants :
  1. appliquer un des systèmes hybrides existants à des applications réelles, de manière à dépasser le stade de prototype auquel la plupart des systèmes hybrides significatifs sont restés,
  2. développer un prototype de système hybride permettant d'explorer de nouvelles directions de recherche.
- les chapitres 3 et 4 présentent notre étude du premier problème. Le chapitre 3 consiste en une description concise du système SYNHESYS, tandis que le chapitre 4 détaille le travail effectué sur une application en micro-électronique et une en géographie alpine.
- le chapitre 5 concerne le développement du nouveau prototype. Il contient une

réflexion sur une architecture multi-niveaux et sur une application appropriée à ce type de couplage. Puis nous décrivons la réalisation de notre prototype qui permet à un robot mobile d'éviter des obstacles.

Chapitre 1

Systèmes Hybrides NeuroSymboliques (SHNS) : généralités

Les systèmes hybrides tentent de tirer parti des points forts de différents paradigmes pour résoudre des problèmes jusqu'alors insolubles, pour couvrir un champ d'application plus large, ou pour obtenir des performances plus élevées. Dans ce mémoire nous nous consacrons aux systèmes neurosymboliques (SHNS), un domaine de recherche très récent (environ cinq ans) qui est en cours d'évolution et de maturation. Dans ce chapitre, nous définissons les deux composants d'un SHNS, système symbolique et réseau de neurones artificiels, et nous montrons leur complémentarité. Nous insistons surtout sur les réseaux de neurones artificiels, car ils sont beaucoup moins connus. En particulier, nous présentons les divers types de tâches que peuvent accomplir les réseaux : approximation de fonctions, compression de données, regroupement, optimisation par exemple. Étant donné que des méthodes mathématiques non-neuronales, souvent beaucoup plus anciennes, permettent aussi d'effectuer ces tâches, nous donnons des justifications de l'intérêt actuellement accordé aux méthodes neuronales.

L'idée de réaliser des systèmes hybrides dépasse le cadre des systèmes neurosymboliques et même celui de l'informatique. Aussi nous commençons ce chapitre par une série d'exemples d'applications ayant tiré parti de la synergie entre plusieurs disciplines ou méthodes pour apporter des solutions novatrices à des problèmes concrets. Nous faisons également une analogie avec une tentative d'hybridation dans un domaine très différent afin d'illustrer les problèmes auxquels sont confrontés les concepteurs de systèmes hybrides, informatiques ou non :

- définir la répartition du travail entre les divers composants,
- définir les coopérations possibles entre ces composants, et trouver les mécanismes qui permettent d'obtenir cette coopération,
- évaluer si on garde bien les avantages de chaque composant sans prendre ses faiblesses,

- évaluer les apports du système hybride par rapport aux systèmes non hybrides,
- étudier la rentabilité du système hybride par rapport aux systèmes non hybrides.

1.1 Résolution de problèmes et hybridation

Les SHNS font partie du domaine de **la résolution de problèmes**. Au moins trois grandes approches peuvent être distinguées dans ce domaine :

- la seule recherche d'une solution : la qualité, l'efficacité de la méthode employée sont prépondérantes ; des méthodes mathématiques « classiques[1] » (approximation, contrôle, optimisation, statistique, etc.) sont souvent utilisées ;
- la recherche d'une solution **explicable** : il est indispensable d'expliquer à l'utilisateur comment et pourquoi tel résultat a été obtenu, et pour cela il est souvent utile de disposer de connaissances explicites sur le processus de résolution de problèmes ; ces connaissances explicites, généralement fournies par un expert, sont modélisées à l'aide de systèmes à base de connaissances [Chaillot 1993, Willamovski 1994]. Dans cette approche, l'efficacité de la méthode employée est parfois moins importante ;
- la recherche de la compréhension des mécanismes cognitifs qui permettent à un expert, ou plus généralement à un être vivant, de résoudre des problèmes ; cette approche relève des Sciences Cognitives [Varela 1989].

Des liens existent entre ces approches, notamment entre la deuxième et la troisième. En effet, la compréhension de certains mécanismes cognitifs peut aider à la représentation de connaissances, tandis que certains modèles de représentation de connaissances peuvent être étudiés comme modèles cognitifs.

Nous présentons ci-dessous les motivations qui conduisent à réaliser des SHNS, puis nous montrons que la volonté de réaliser des systèmes hybrides existe dans d'autres domaines que celui des SHNS.

1.1.1 Motivations de la recherche d'une synergie neurosymbolique

On retrouve dans les SHNS l'ambivalence des approches possibles en résolution de problèmes : certains veulent construire de meilleurs outils informatiques pour résoudre des problèmes [Becraft, Lee, et Newell 1991], d'autres veulent construire de meilleurs modèles cognitifs [Harnad 1990, Dinsmore 1992b], et beaucoup ont des démarches intermédiaires [Amy 1991, Giacometti 1992] : utiliser des idées provenant des Sciences Cognitives pour obtenir de meilleurs outils informatiques. Toutefois,

1. Dans le reste du mémoire, nous qualifierons souvent de telles méthodes de « non-neuronales » pour les distinguer de méthodes accomplissant des tâches équivalentes, mais réalisées sous forme de réseaux de neurones artificiels.

que ces recherches soient plutôt informatiques ou plutôt cognitives, elles ont un point commun important : elles sont toutes motivées par l'incapacité actuelle d'un paradigme donné (symbolique, connexionniste ou autre) à résoudre à lui seul des problèmes difficiles. Par exemple la reconnaissance d'un visage est difficile pour un système symbolique, ou alors que l'homme (voire certains animaux) accomplissent facilement ce type de tâches. D'autres problèmes sont difficiles à traiter avec des réseaux de neurones artificiels, comme la planification.

Étant donné qu'il est très difficile d'inventer de toutes pièces un nouveau paradigme plus satisfaisant, la « démarche du moindre effort » consiste à tirer parti des points forts de plusieurs paradigmes et à réaliser des systèmes ou modèles hybrides. Dans le cas des systèmes neurosymboliques, on constate que les points forts des systèmes symboliques compensent à peu près les points faibles des systèmes connexionnistes et vice-versa. Nous développons cette idée plus loin. Dans ce type de démarche, les systèmes hybrides constituent une voie de recherche intéressante, « *the path of least resistance in the short term* » [Hendler 1989].

Les chercheurs vont plus loin que ce que nous avons appelé la « démarche du moindre effort » en donnant des justifications cognitives à la réalisation de modèles hybrides. Pour Giacometti [1992], dans un système expert, il est indispensable de disposer de plusieurs modes complémentaires d'expression des connaissances, de manière à pouvoir représenter le « savoir-que » et le « savoir-faire » d'un problème : un SHNS est un des moyens d'offrir différents modes d'expression. Pour Dinsmore [1992b], les systèmes hybrides sont une nécessité, en raison de la complexité des systèmes cognitifs : il pense que quand un chercheur examine les différents aspects d'un problème cognitif et sélectionne les niveaux de description les plus utiles, le modèle qui émergera sera un patchwork de composants symboliques et connexionnistes, donc un modèle hybride.

Mais ces arguments restent controversés, car dans les sciences cognitives il y a d'autres manières d'envisager les relations entre les mondes symboliques et connexionnistes [Dinsmore 1992b] :

- la philosophie de l'élimination (*eliminative connectionism*) : il s'agit de supprimer les modèles symboliques qui ne seraient pas du tout nécessaires à la description complète de la cognition ;
- la philosophie de l'implantation (*implementational connectionism*) : les systèmes connexionnistes sont seulement une autre manière de réaliser les structures et processus symboliques.

Quoi qu'il en soit, en attendant que triomphe éventuellement l'une de ces approches, les systèmes hybrides neurosymboliques constituent une direction de recherche intéressante, comme en témoignent l'organisation de plusieurs « workshops » [Sun et Bookman 1992, Hilario 1994], la publication de numéros spécialisés de revues [AISB 1992, Bookman et Sun 1993], et de livres [Kandel et Langholz 1992, Medsker 1994].

1.1.2 Des exemples de systèmes hybrides pour la résolution de problèmes

Il existe de nombreux exemples de systèmes hybrides, non nécessairement neuro-symboliques, qui donnent des résultats très intéressants.

Par exemple, CABOT [Callan 1991] est un système de Raisonnement à Base de Cas (RBC) qui joue au jeu de plateau Othello. CABOT possède l'architecture classique d'un système RBC (recherche de plusieurs cas similaires au problème à traiter, sélection de l'un d'eux, adaptation de la solution, stockage du nouveau cas résolu) mais il a la particularité de pouvoir ajuster ses mécanismes de recherche et d'adaptation, en fonction d'un retour d'information (*feedback*) sur la qualité du coup qu'il a choisi de jouer. Ce retour est limité car on dit au système quel aurait été le meilleur coup, mais on ne lui dit pas pourquoi ce coup aurait été meilleur. Les deux mécanismes adaptables sont réalisés à l'aide de LTUs (*Linear Threshold Units*) ; unités qui calculent une fonction linéaire de leurs entrées puis appliquent une fonction seuil. L'apprentissage est effectué sur les paramètres de ces fonctions. Grâce à cette capacité d'apprentissage, le système est capable de battre plusieurs autres programmes, dont un système de RBC classique, ce qui montre l'intérêt de ce couplage RBC/apprentissage automatique.

Il existe aussi un système de contrôle réactif d'un robot autonome [Ram et Santamaria 1993], qui doit trouver un chemin dans un terrain encombré d'obstacles. Les systèmes réactifs sont une alternative aux méthodes de planification, et se basent sur une simple représentation de leur environnement pour choisir la prochaine action à exécuter. Ces méthodes sont utiles dans des environnement inconnus et variables, mais ne peuvent pas en général modifier ou améliorer leur comportement avec leur expérience. Pour cette raison, les auteurs ont ajouté un module d'apprentissage à un module de navigation réactive. Ce module d'apprentissage règle en permanence certains paramètres du module de navigation et combine lui-même deux méthodes, le RBC et l'apprentissage renforcé. Le composant RBC perçoit et caractérise l'environnement, recherche un cas approprié, et utilise les recommandations de ce cas pour régler les paramètres ; le composant d'apprentissage renforcé, de son côté, affine le contenu des cas en fonction de l'expérience courante du système.

Le système MORPH [Gould et Levinson 1991] combine plusieurs méthodes d'apprentissage et parvient à jouer à un niveau moyen aux échecs (il obtient des parties nulles contre GnuChess Level I, réputé plus fort que 60% des joueurs de tournoi), mais avec peu de connaissances sur le domaine, une faible profondeur de recherche (1 coup de profondeur) et aucun professeur indiquant la qualité d'un coup. Les méthodes étudiées sont l'apprentissage par la méthode des différences temporelles, le recuit simulé, les algorithmes génétiques, la généralisation à base d'explication, l'induction de concepts structurés, le tout combiné avec deux fonctions d'évaluation heuristique. On change donc d'échelle par rapport aux systèmes précédents, du point de vue de la complexité du système. Les auteurs soulignent qu'ils ont plutôt cherché à utiliser l'essence de chacune des méthodes, plutôt que la méthode elle-même, et que la nécessité de les combiner a parfois conduit à relâcher certaines contraintes associées aux méthodes. Une description détaillée de ce système très complexe sor-

tirait du cadre de cette introduction, mais ce qui nous semble important à ce stade est que les auteurs soulignent que cette intégration n'aurait pas été possible sans une représentation des connaissances commune à toutes ces méthodes. On voit donc là une première contrainte pouvant apparaître dans la conception d'un système hybride. On devine également que la vérification, la validation, l'évaluation d'un tel logiciel peuvent être fort complexes, d'autant plus qu'il est nécessaire d'avoir des compétences dans plusieurs domaines.

Enfin, le besoin de structuration des connaissances dans les systèmes experts a conduit à réaliser des systèmes experts hybrides, comprenant par exemple des règles de production et des objets. C'est le cas de la plupart des générateurs de systèmes experts commerciaux, comme SMECI [ILOG 1992].

1.1.3 Une analogie : la voiture hybride

L'hybridation de techniques ou d'idées n'est pas propre à l'informatique. C'est même une démarche technique ou scientifique plutôt courante. Un exemple récent est la **voiture hybride** : les voitures électriques, silencieuses, apparemment peu polluantes, utilisant une énergie peu coûteuse (car peu taxée), pourraient bientôt connaître un succès important auprès du public. Toutefois, leur autonomie réduite (une centaine de kilomètres) limite leur utilisation à la ville. Les voitures thermiques, par contre, sont bruyantes, assez polluantes, utilisent une énergie plus coûteuse, mais ont une autonomie importante. Le seul moyen, actuellement, de profiter des avantages des deux types de voiture est de réaliser une voiture hybride, et cela est sérieusement étudié par certains constructeurs. Dans ce cas le partage des tâches entre les deux moteurs est simple : la voiture est électrique en ville, thermique sur les grands trajets, éventuellement les deux en même temps pour profiter momentanément d'un surcroît de puissance.

Une telle solution aurait de plus pour avantage de rationaliser l'utilisation des moteurs thermiques, qui fonctionnent de manière optimale quand ils sont chauds et en régime stationnaire. Par contre, il est probable que les moteurs thermiques utilisés ne seront pas du même type que ceux que nous utilisons actuellement (études sur les turbines à gaz).

Une telle voiture pourrait être bientôt commercialisée en France. Cette idée de voiture hybride est très séduisante sur le papier, mais il reste à voir si ses avantages (qui sont en principe ceux des deux types de propulsion) l'emporteront sur ses inconvénients (plus chère, plus lourde par exemple).

On pourrait objecter que cette analogie est plutôt limitée, car les deux moteurs semblent couplés de manière assez faible, et fonctionnent séparément. Ce n'est pas le cas car dans les projets les plus avancés, ces deux moteurs peuvent « coopérer » : lorsque la voiture est en phase thermique, on peut recharger les batteries en vue de la prochaine phase électrique. L'intégration des deux moteurs peut aussi se faire de plusieurs manières : soit ils sont complètement séparés, soit ils partagent un arbre de transmission commun, par exemple. Dans ce dernier cas, la course de la pédale d'accélération peut éventuellement déterminer quel moteur doit être employé.

Cet exemple de la voiture hybride est très intéressant car il illustre bien les problèmes auxquels sont confrontés les concepteurs de systèmes hybrides, informatiques ou non :

- définir la répartition du travail entre les divers composants,
- définir les coopérations possibles entre ces composants, et trouver les mécanismes qui permettent d'obtenir cette coopération,
- évaluer si on garde bien les avantages de chaque composant sans prendre ses faiblesses (le danger est en effet que le système hybride combine les inconvénients de ses composantes au lieu des avantages),
- évaluer les apports du système hybride par rapport à ses composants,
- étudier la rentabilité du système hybride par rapport aux systèmes non hybrides.

Tout au long de ce mémoire nous verrons que ces mêmes questions doivent être étudiées dans le cas des systèmes hybrides neurosymboliques.

1.2 Les systèmes symboliques

Il s'agit des travaux qui reposent sur la manipulation de symboles. C'est l'approche familièrement appelée la « bonne vieille IA » (*GOFAI, Good Old Fashioned AI*) [Grumbach 1994], et qui repose souvent sur l'hypothèse du système de symboles physiques [Newell et Simon 1976] :

> « A physical symbol system has the necessary and sufficient means for general intelligent action. »

Ces auteurs donnent la définition suivante d'un système de symboles physiques :

> « A physical symbol system consists of a set of entities, called symbols, which are physical patterns that can occur as components of another type of entity called an expression (or symbol structure). ... A physical symbol system is a machine that produces through time an evolving collection of symbol structures. »

On peut aussi qualifier l'Intelligence Artificielle (IA) symbolique d'IA restreinte, par opposition à une IA généralisée qui inclut le connexionnisme [Grumbach 1994].

Nous développons dans cette section les divers systèmes symboliques intervenant dans les SHNS, et avant d'en examiner les points forts et les points faibles, nous précisons la place qu'occupe actuellement le calcul numérique dans de tels systèmes.

1.2.1 Les systèmes symboliques dans les systèmes hybrides

Dans le domaine des SHNS, la littérature ne donne pas réellement de définition précise du terme « système symbolique », et la plupart des auteurs [Kasabov 1990, Gutk-

necht et Pfeifer 1990, Sun 1991, Tirri 1991, Giacometti et al. 1992, Towell 1992] ne considèrent que les systèmes experts à base de règles de production (le plus souvent d'ordre 0 ou 0^+), mais quelques autres utilisent des réseaux sémantiques [Hendler 1989], des systèmes à base d'objets [Brachman et McGuinness 1988] ou encore des systèmes de RBC [Callan 1991]. En fait, la notion de système symbolique correspond assez bien aux « systèmes à base de connaissances » [Rechenmann 1991], qui mettent l'accent sur la séparation des connaissances et de leur contrôle. L'archétype du système symbolique est sans doute le système formel, mais il sert plutôt de support théorique que d'outil pratique.

Nous verrons dans le chapitre 2 que pour rendre compte de l'ensemble des travaux neurosymboliques, il est nécessaire d'accepter une définition plus étendue des systèmes symboliques. En effet, pour certains, une base de règles, un algorithme voire un programme sont des systèmes symboliques. Le mot « symbolique » est alors utilisé comme synonyme de « explicite » ou « verbalisable ».

1.2.2 Aspects numériques des systèmes symboliques

Nos lectures nous conduisent toutefois à souligner que cet adjectif « symbolique » caractérise fort mal les travaux actuels en résolution de problèmes. Certes, les systèmes experts, avec leurs règles de production, sont relativement proches des systèmes formels, symboliques, mais leur application à des problèmes pratiques peut conduire à introduire des coefficients numériques associés aux règles et aux prémisses (comme les coefficients de vraisemblance de MYCIN, cité dans [Laurière 1987]) pour obtenir une plus grande souplesse. D'autre part, les générateurs commerciaux de systèmes experts, comme SMECI [ILOG 1992], deviennent souvent des sortes de langages très puissants permettant à la fois la représentation de connaissances et la programmation (orientée-objet) où le numérique peut jouer un rôle très important. De même, les systèmes de déduction automatique, comme SETHEO [Letz et al. 1992, Goller 1994] utilisent souvent des heuristiques numériques et ne manipulent donc pas uniquement des symboles.

Dans un autre registre, des systèmes de représentation de connaissances comme SHIRKA [Rechenmann et al. 1988] permettaient dès leur création d'appeler des procédures externes, provenant par exemple d'une bibliothèque d'analyse numérique, et l'on cherche d'ailleurs à intégrer ces procédures [Willamovski 1994, Orsier 1990] dans des modèles de représentation de connaissances.

De même, il suffit d'ouvrir un livre récent d'IA, comme celui de Haton, Bouzid, et al. [1991], pour constater qu'à côté de la traditionnelle logique mathématique, du raisonnement qualitatif, du raisonnement hypothétique et du maintien de la vérité, tous grands utilisateurs de symboles, une large place est faite au raisonnement approximatif, essentiellement numérique.

On pourrait également citer le RBC qui utilise des mesures souvent numériques de similarité. De son côté, la logique floue est aussi plutôt d'inspiration symbolique, mais manipule des quantités numériques (degrés d'appartenance à des ensembles).

L'apprentissage automatique compte aussi des systèmes comme KBG [Bisson 1993], qui a une forte composante numérique (résolution d'un système d'équations linéaire).

En conclusion, il nous paraît important de garder à l'esprit que cette IA dite « symbolique » est certes fortement marquée par l'utilisation de symboles, ce qui est sans doute à l'origine de certaines de ses limites, mais que le numérique y prend actuellement une part de plus en plus importante, et permet de mieux prendre en compte le caractère approximatif de beaucoup de connaissances et de raisonnements. Les systèmes hybrides neurosymboliques s'insèrent donc dans un courant plus général de coopération entre les mondes numériques et symboliques. En fait, du point de vue historique, l'IA s'est beaucoup ouverte aux techniques numériques, par nécessité, pendant que les SHNS se développaient (depuis 1989 environ) en restant essentiellement basés sur la *GOFAI*.

1.2.3 Points forts des systèmes symboliques

Du point de vue informatique, les systèmes symboliques ont plusieurs points forts qui expliquent leur succès :

- une capacité d'explication au moins potentielle (même si dans la pratique il reste difficile de dépasser le stade de la trace du raisonnement),
- la séparation des connaissances et du système qui les exploite (moteur d'inférence, etc.), qui permet l'évolution séparée et la réutilisation de ces deux composants de base,
- leur nature déclarative, qui facilite l'expression des connaissances,
- le caractère fortement structuré des connaissances qu'ils permettent de représenter. Ce caractère reste pour le moment indispensable dans la plupart des applications en IA (traitement des langues, etc.).

Du point de vue cognitif, les systèmes symboliques ont l'intérêt de posséder certaines des propriétés importantes de la cognition [Fodor et Pylyshin 1988, Schwartz 1992, Crucianu 1994] :

- la productivité (*productivity*) : la capacité non bornée de représenter des propositions [Schwartz 1992] ; ou encore, d'après Crucianu [1994] : « la propriété hypothétique de la cognition d'avoir une capacité de représentation et de traitement – donc une compétence – potentiellement illimitée ».
- la systématicité (*systematicity*) : la propriété de la cognition de présenter certaines symétries [Crucianu 1994]. Il faut distinguer systématicité d'inférence (ou même de traitement) et systématicité de représentation [Barnden 1992b]. La systématicité d'inférence est la capacité de faire toutes les inférences d'un certain type logique, pas seulement quelques unes d'entre elles. Par exemple, un système pouvant inférer P à partir de $P \wedge Q \wedge R$ mais ne pouvant inférer P à partir de $P \wedge Q$ ne possède pas la propriété de systématicité. La systématicité de représentation suppose qu'un système soit toujours capable de représenter la proposition P quand (1) il est capable de représenter une proposition Q, et

(2) P et Q sont suffisamment similaires, en un sens donné. Par exemple, un système possédant la systématicité de représentation et pouvant représenter *Mary aime John* devrait pouvoir aussi représenter *John aime Mary*. Finalement, d'après Crucianu [1994], « Si la productivité fait l'hypothèse que personne n'a une compétence intellectuelle **finie**, la systématicité affirme que personne n'a une compétence intellectuelle **ponctuelle** ».

- la compositionnalité (*compositionality*)[2] : ce n'est pas une propriété observable de la cognition, mais plutôt un élément d'explication des deux propriétés ci-dessus [Crucianu 1994]. Pour avoir les propriétés en question, un système cognitif doit posséder la compositionnalité : les représentations utilisées par le système cognitif doivent avoir une structure reposant sur la composition, ce qui veut dire que les constituants d'une représentation doivent être explicitement concaténés selon des règles de combinaison strictes, et que les processus cognitifs doivent être systématiquement sensibles à la structure [Memmi 1993]. Par exemple, la transformation de la représentation d'une phrase active en représentation d'une phrase passive doit être sensible à la structure de la phrase active [Barnden 1992b].

C'est pourquoi les systèmes symboliques rencontrent un grand succès parmi les chercheurs qui étudient la cognition.

1.2.4 Points faibles des systèmes symboliques

Nous avons regroupé dans ce paragraphe les principaux points faibles que mettent en avant les concepteurs de systèmes hybrides. Toutefois, ces diverses limites des systèmes symboliques sont bien connues, et des solutions sont en cours d'étude (par exemple, logique floue, raisonnement à base de cas, apprentissage automatique).

Quelques points faibles assez généraux

Divers points faibles sont généralement avancés, outre le problème classique de l'acquisition des connaissances [Beaubouche r 1994]. Dans le cas des systèmes experts, on constate notamment :

- la non-amélioration des performances avec l'expérience [Gutknecht et Pfeifer 1990]. Si un système expert doit résoudre un problème déjà rencontré, il va refaire exactement les mêmes étapes de raisonnement. Alors qu'un expert humain mémorise et réutilise la connaissance précédemment acquise.
- l'incapacité de s'adapter [Gutknecht et Pfeifer 1990]. En cas de changement dans l'environnement (par exemple l'amélioration d'une partie d'un appareil sujette à des pannes fréquentes) le comportement d'un système expert ne change pas (à moins de modifier la base de connaissances), alors qu'un expert changerait immédiatement sa stratégie de diagnostic.

2. Le lecteur pourra aussi se reporter à [Grumbach 1994] pour une présentation plus rigoureuse de cette notion

- la faible (ou inexistante) « dégradation progressive » (*graceful degradation*) des performances [Gutknecht et Pfeifer 1990]. Lorsqu'une situation n'a pas été prédéfinie, ou lorsque les données sont incomplètes ou bruitées, un système expert échoue complètement, alors qu'un expert humain donnera quand même une décision, éventuellement moins bonne.
- la difficulté de prendre en compte dans un système symbolique des données provenant de capteurs [Tirri 1991]. De telles données se caractérisent par leur appartenance à un espace de dimension élevée, et par le fait qu'elles peuvent être bruitées ou incomplètes.
- la difficulté à généraliser. Lorsque la solution d'un problème A, exprimé sous la forme de structures de symboles, est connue, il n'est pas simple de donner une solution pour un problème B proche. En effet, la notion même de proximité est délicate à définir dans un système symbolique.

D'autres points faibles sont la difficulté de la construction d'applications de grande taille [Medsker 1994] et de leur maintenance [David, Krivine, et Simmons 1993]. Mais ces points faibles sont très généraux et sont propres à tous les systèmes informatiques de grande taille. D'autre part, le coût des opérations effectuées dans les systèmes symboliques peut être élevé, étant donné que la plupart des ordinateurs actuels sont plus conçus pour le traitement des nombres que pour celui des symboles [Cousin et al. 1988]. Mais les machines dédiées à la manipulation de symboles n'ont pas eu le succès escompté.

D'un point de vue plus cognitif, les systèmes symboliques ne sont pas satisfaisants car ils ne permettent pas de réaliser certaines tâches que font facilement les humains. Ces tâches (appellées parfois processus mystérieux [Dinsmore 1992b]) sont des opérations synthétiques difficiles ou impossibles à verbaliser [Cornu 1992, Giacometti 1992], comme la reconnaissance d'un visage par exemple.

La fragilité des systèmes symboliques

La fragilité des systèmes symboliques est l'une de leurs plus importantes et incontestables limitations, éventuellement acceptable quand on utilise les outils symboliques cités ci-dessus comme des langages de programmation puissants mais pas quand on veut construire un système « intelligent ».

Malheureusement la fragilité est une notion assez floue, de type « fourre-tout ».

Pour R. Sun [Sun 1991], qui s'intéresse au raisonnement de sens commun, la fragilité contient notamment l'un des points que nous avons mentionné ci-dessus, la dégradation brutale des performances des systèmes symboliques face à une situation inconnue. Par exemple, imaginons une base de règles contenant uniquement la règle $A \longrightarrow B$; si le fait C apparaît, le système ne peut donner aucune réponse (aucune règle ne s'applique), même si le fait C est proche, en un sens à préciser, du fait A. Mais Sun va plus loin en considérant que **la fragilité c'est l'incapacité à traiter, d'une manière systématique et dans un cadre unifié, les aspects**

suivants du raisonnement :

- information partielle,
- information incertaine ou floue,
- absence de règles applicables,
- les interactions entre règles (c'est-à-dire le manque de consistance et de complétude dans une base de règles partielle)[3],
- l'héritage de propriétés,
- l'apprentissage de nouvelles règles et la modification des règles existantes.

D'autres auteurs réduisent la fragilité au problème de l'absence de règles applicables. Une telle définition est donnée par Kwasny et Faisal [1992], dans le domaine du traitement des langues : pour ces auteurs, les systèmes utilisant des règles tendent à être fragiles car il n'y a pas de moyen direct de traiter des formes linguistiques qui n'ont pas été anticipées par les règles. Pour d'autres, les systèmes à base de règles tendent à être fragiles en ce qu'ils ne fonctionnement pas bien en dehors de leur domaine limité d'expertise [David, Krivine, et Simmons 1993]. En raison de leur absence de connaissances sur leurs propres limites, les systèmes experts tendent à donner des réponses absurdes au lieu de répondre simplement « je ne sais pas ». Une autre cause de fragilité, pour ces auteurs, provient des interactions entre règles : il y a souvent des interactions négatives (conflits) entre les règles, ce qui réduit les performances. En particulier l'ajout de nouvelles règles peut rendre le système incapable de résoudre certains problèmes qu'il pouvait résoudre avant.

La fragilité des systèmes symboliques est de plus liée à une certaine **rigidité**. Brachman et McGuinness [1988] expliquent que les systèmes symboliques de représentation des connaissances sont limités à des inférences très rigides, ce qui les rend difficiles à utiliser dans des domaines comme la recherche d'informations où les notions de proximité et de similarité sont à l'ordre du jour.

Le problème de l'ancrage des symboles

Là aussi il s'agit d'une limitation fondamentale des systèmes symboliques, à l'origine mise en lumière par des chercheurs en sciences cognitives mais aujourd'hui largement reprise par des informaticiens qui cherchent soit des solutions concrètes [Lallement et Durand 1994] soit à éviter ce problème [Dedieu et Bessière 1994].

Le problème de l'ancrage des symboles (« symbol grouding problem ») est défini par Harnad [1990] de la manière suivante :

> « How can the semantic interpretation of a formal symbol system be

3. Ce point n'est pas très clair avec la formulation employée par Sun, que nous avons traduite directement. Sun donne en fait l'exemple suivant. On dispose de deux règles « *if carrying cargo, buy a utility vehicle* » et « *if carrying passengers, buy a passenger vehicle* ». Un humain est alors capable de décider que s'il doit transporter à la fois du fret et des passagers, il doit acheter un véhicule qui possède les caractéristiques des deux types de véhicules, par exemple un *van*. L'auteur appelle cela « interaction additive de deux règles ».

> made *intrinsic* to the system, rather than just parasitic on the meanings in our heads? How can the meanings of the meaningless symbol tokens, manipulated solely on the basis of their (arbitrary) shapes, be grounded in anything but other meaningless symbols? »

Selon Harnad, les symboles d'un système purement symbolique ne sont pas ancrés, dans le sens où les symboles apparaissant dans un dictionnaire chinois/chinois ne sont pas ancrés pour un observateur ne connaissant pas le chinois auparavant. Si un tel observateur tente d'apprendre le chinois avec ce dictionnaire, il va passer de symboles en symboles sans jamais arriver à une signification.

Pour Harnad, un système qui n'est pas ancré ne peut pas comprendre. C'est donc un obstacle majeur à la réalisation de systèmes intelligents. Pour lui, une solution possible au problème de l'ancrage consiste à réaliser un système hybride symbolique/non-symbolique, dans lequel les réseaux de neurones artificiels auraient probablement un rôle important à jouer. L'ancrage des symboles peut donc être une motivation supplémentaire pour réaliser des systèmes hybrides [Orsier 1993a].

Beaucoup de recherches visent à relier les symboles d'un système symbolique au monde réel, en dotant l'ordinateur de capteurs et d'effecteurs ; pour cette raison on parle beaucoup d'« ancrage perceptif des symboles ». Mais pour Grumbach [1994], cela ne suffit pas et il faudrait passer à des systèmes d'IA autonomes capables de forger eux-mêmes leurs concepts et leurs symboles.

1.3 Réseaux de neurones artificiels

Dans cette section nous présentons le composant neuronal d'un système hybride. En fait, le terme « réseau de neurones artificiels » regroupe une grande variété de modèles, allant de l'outil de l'ingénieur au modèle cognitif en passant par la simulation neuro-biologique. Le point commun de tous ces travaux est une certaine inspiration neuro-biologique, bien que la plupart des modèles ne miment pas le fonctionnement du cerveau, d'ailleurs encore mal connu. Parler de réseaux de neurones est donc la plupart du temps une métaphore, et comme toute métaphore, il importe de connaître ses limites pour ne pas la pousser trop loin.

Nous nous limitons ici à trois modèles représentatifs de ce que nous considérons plutôt comme outils de l'ingénieur, et dont nous aurons besoin dans les chapitres suivants. Nous précisons tout d'abord le vocabulaire que nous utiliserons dans le reste de ce mémoire, avant d'examiner les grandes classes d'application des réseaux de neurones artificiels. Après la présentation des trois modèles, nous étudierons plusieurs questions qui restent ouvertes, ainsi que les points forts et les points faibles des réseaux de neurones artificiels.

Remarque : par abus de langage, nous parlerons quelquefois de « réseaux de neurones » dans la suite de ce document, alors qu'il serait nécessaire de bien préciser « réseaux de neurones artificiels ». Sauf si cela est clairement précisé, tous les neurones mentionnés à partir d'ici sont **artificiels**.

1.3.1 Vocabulaire

Le **neurone formel** est un modèle d'automate, introduit dans les années 40 pour modéliser le fonctionnement du système nerveux humain [Bourret, Reggia, et Samuelides 1991]. Un neurone formel i est défini au minimum par :

- un **état interne** aussi appelé une **activation**, qui est une valeur réelle ou booléenne $a_i(t)$ transmise à d'autres neurones,
- des **connexions** lui permettant de recevoir des signaux $a_j(t)$ provenant d'autres automates,
- des **poids** de transmissions $w_{ij}(t)$ qui sont des coefficients réels associés respectivement aux entrées $a_j(t)$,
- une **fonction d'activation** (ou encore fonction de transfert, ou de seuil), permettant de calculer l'activation en fonction de l'entrée du neurone qui est la somme pondérée : $\sum_j w_{ij}(t)a_j(t)$.

Les mots **unité** et **nœud** sont souvent employés pour désigner un neurone.

Un **réseau de neurones artificiels** est un ensemble de neurones formels, organisés selon une certaine topologie, et possédant une certaine dynamique, laquelle dépend de l'initialisation des poids et d'une règle de propagation des activations. Chaque neurone ne « connaît » que les neurones auxquels il est connecté (notion de localité des opérations). Certains modèles de réseaux possèdent en plus une **règle d'apprentissage** permettant de faire évoluer les poids et de les adapter au problème à résoudre (classification, regroupement, etc – voir le paragraphe 1.3.2). Dans les modèles les plus sophistiqués, la topologie du réseau peut elle-même évoluer.

Le mot **connexionnisme** est souvent employé pour désigner l'ensemble des idées et travaux relatifs aux réseaux de neurones, artificiels ou non, bien qu'il ait un sens plus général car il concerne notamment tous les travaux sur les automates connectés. Le mot **neuro-mimétisme** est plus approprié mais moins courant (il est surtout utilisé en France). On associe souvent connexionnisme et **processus sub-symboliques**, qui sont le sujet d'études cognitives et informatiques inspirées par les réseaux de neurones.

Il est important également de distinguer les réseaux **localistes** des réseaux **distribués**. Dans les réseaux localistes, chaque neurone représente seulement un concept (ou caractéristique sémantique) du domaine d'application, tandis que dans les réseaux distribués, un concept est réparti sur plusieurs neurones (chacun de ces neurones peut représenter une caractéristique pas nécessairement identifiable, souvent appelée micro-caractéristique).

Nous distinguerons également les réseaux **récurrents**, dans lesquels le graphe orienté des connexions comporte des cycles, et les réseaux non récurrents (ou acycliques), dans lesquels ce graphe ne comporte pas de cycles.

Les réseaux capables d'apprentissage apprennent des **exemples**, qui sont en fait :

- des couples de vecteurs $(X, Y) \in \mathbb{R}^n \times \mathbb{R}^p$, dans le cas de l'apprentissage **supervisé** : le réseau doit apprendre à donner la réponse Y quand X lui est présenté en entrée ; cet apprentissage est employé dans des applications de type classification et plus généralement de type approximation de fonctions (on parle aussi souvent d'« association ») ;
- des vecteurs $X \in \mathbb{R}^n$, dans le cas de l'apprentissage **non supervisé** : aucun professeur n'est disponible pour indiquer au réseau quelle réponse il doit fournir quand X lui est présenté en entrée ; cet apprentissage est employé dans des applications de type regroupement.

Notons que les entrées (et les sorties, quand il y a lieu d'en parler) d'un réseau sont nécessairement des vecteurs : en effet elles correspondent aux activations des unités d'entrées et de sortie arbitrairement ordonnées. L'une des difficultés de l'utilisation des réseaux provient de la nécessité de coder toute donnée, même qualitative, sous la forme d'un vecteur de $\mathbb{R}^n$ ou $\{0,1\}^n$.

1.3.2 Que peut-on faire avec des réseaux de neurones artificiels (RNA) ?

L'identification précise des tâches que peuvent accomplir les réseaux de neurones artificiels est très importante pour déterminer la répartition des tâches dans un SHNS. Cette identification nous paraît d'autant plus indispensable que l'habitude a été prise de parler des réseaux de neurones artificiels d'une manière très générale, alors que leurs propriétés et leurs applications possibles peuvent être très différentes selon les modèles.

Cette identification est toutefois rendue difficile par la variété des modèles et du vocabulaire employé. Hrycej [1992] propose néanmoins les **classes d'applications** suivantes :

- **l'approximation de fonctions** : certains réseaux, en particulier ceux du paragraphe 1.3.3, montrent des capacités d'approximation de fonctions très intéressantes. D'autre part, l'approximation de fonctions est un cadre théorique pour des applications concrètes, comme :
  - **la classification** : dans le cas des réseaux, on s'intéresse essentiellement à la classification d'objets pouvant être décrits par des vecteurs de caractéristiques numériques et booléennes (la classification d'objets structurés, au sens des représentations de connaissances par objets [Rechenmann 1991] semble pour le moment être hors de portée). Ce type de classification est aussi fréquemment appelé **reconnaissance des formes**, tandis qu'une sous-classe très importante de la classification est le **diagnostic**, médical ou industriel par exemple.
  - **le contrôle** : il s'agit ici de construire un contrôleur pour un appareil donné. Le contrôleur reçoit des informations de l'appareil, et doit fournir des commandes qui modifient le fonctionnement de l'appareil de façon

à obtenir une performance maximum par rapport à certains objectifs. Traditionnellement ces problèmes de contrôle sont surtout étudiés par les automaticiens.

- **la compression de données** : il s'agit ici d'obtenir une représentation compacte d'un ensemble de vecteurs. Une utilisation particulière des réseaux du paragraphe 1.3.3 permet de compresser des images (c'est la technique de Cottrell/Munro/Zipser [Hecht-Nielsen 1989], dans laquelle le réseau apprend à approximer la fonction identité, et les valeurs d'activités de la couche cachée constituent la représentation compacte cherchée), mais il existe d'autres méthodes [Hrycej 1992]. Cette classe mentionnée par Hrycej est toutefois marginale.
- **le regroupement** (*clustering*) et **la quantification** (*quantization*) : le regroupement consiste à faire une partition d'un ensemble d'objets (bien sûr décrits par des vecteurs) en un certain nombre de groupes (*clusters*). Chaque groupe peut être étiqueté par un symbole ou un nombre. Le fait que des vecteurs de caractéristiques continues peuvent ainsi être étiquetés par une quantité discrète justifie l'autre dénomination de cette tâche : la quantification.
- **l'auto-organisation** des cartes de Kohonen : il est difficile de situer correctement les travaux du finlandais Teuvo Kohonen dans cette petite hiérarchie. Elles sont généralement utilisées pour la classification, mais certains les présentent comme un cas particulier de méthodes de regroupement [Murtagh et Hernández-Pajares 1994]. Tomas Hrycej les laisse au même niveau que les principales classes, sans doute à cause de l'importance exceptionnelle qu'ont prises les méthodes s'inspirant des travaux de Kohonen. Cette importance peut être mesurée par l'existence d'une bibliographie de plus de 1200 références sur ces méthodes [Murtagh et Hernández-Pajares 1994], en une quinzaine d'années. Voici une définition possible de ce que font ces méthodes : « Il s'agit d'associer à un nombre fini de vecteurs d'entrée un nombre fini de vecteurs représentants, appartenant à un espace de dimension inférieure à celle de l'espace d'entrée, de telle sorte que des vecteurs voisins dans l'espace d'entrée se trouvent associés à des vecteurs représentants voisins ». Il y a donc en même temps regroupement, réduction de dimensionnalité, préservation de propriétés topologiques, etc. Une autre caractéristique de ces méthodes est que pour le moment elles résistent étonnamment bien à l'analyse mathématique [Cottrell 1993, Cottrell, Fort, et Pagès 1994] et que l'on connaît encore bien mal leurs propriétés. Malgré (ou à cause de?) ces mystères, ces méthodes sont énormément utilisées par des gens qui ont développé un grand savoir-faire [Cottrell, Fort, et Pagès 1994].
- **l'optimisation** : certains réseaux (comme ceux de Hopfield) permettent de résoudre une certaine classe de problèmes d'optimisation (par exemple le problème classique du voyageur de commerce). D'autre part, comme l'approximation, l'optimisation est un cadre théorique très général, et on peut définir des sous-classes plus précises, la plus importante étant celle des **mémoires associatives**. Une mémoire associative est une mémoire qui permet de retrouver les éléments qu'elle stocke à partir d'une description partielle ou bruitée de ces éléments. Ce type de mémoire est adressable par son contenu, contrairement

aux mémoires traditionnellement utilisées en informatique. Il s'agit d'un des thèmes de recherche les plus importants dans le domaine des réseaux neuronaux, car ces mémoires peuvent servir à modéliser l'une des caractéristiques importantes de la mémoire humaine [Labbi 1993]. Les mémoires associatives permettent de réaliser une tâche voisine de la classification, appelée parfois catégorisation : elles permettent de retrouver un représentant de la classe d'un exemple X fourni en entrée (alors que la classification fournit la classe de X).

Cette petite hiérarchie de classes d'applications est bien sûr incomplète, et n'est pas entièrement satisfaisante, à cause de la grande variété des applications imaginées, de la variété du vocabulaire employé, du fait que certaines méthodes sont parfois « détournées » de leur objectif initial (par exemple, Murtagh et Hernández-Pajares [1994] citent un cas d'utilisation des cartes de Kohonen pour résoudre le problème du voyageur de commerce ; voir aussi la technique de Cottrell/Munro/Zipser mentionnée ci-dessus). Une autre difficulté pour hiérarchiser les classes d'applications provient aussi du fait qu'un domaine d'application comme le traitement du signal peut utiliser à la fois des techniques neuronales générales comme celles mentionnées ci-dessus et des techniques neuronales spécifiques, comme la séparation de sources [Jutten et Hérault 1994].

Cette hiérarchie montre que les possibilités d'application des réseaux de neurones artificiels sont très nombreuses (voir par exemple [Hecht-Nielsen 1989, Zeidenberg 1990, Hrycej 1992]), et il s'agit souvent d'applications auxquelles s'est intéressée l'Intelligence Artificielle, comme par exemple le traitement automatique des langues (reconnaissance, synthèse) [Jodouin 1993] ou la reconnaissance de chiffres manuscrits [Azcarraga 1993].

Toutes les tâches mentionnées ci-dessus sont toutefois étudiées, parfois depuis fort longtemps, dans d'autres disciplines. Il est souhaitable de déterminer les relations entre les travaux faits dans ces disciplines et les réseaux de neurones artificiels, afin de ne pas « réinventer la roue ». Un effort de recherche est actuellement mené dans cette direction, et il ne semble pas exister de résultat définitif [Cottrell 1993]. Quelques unes des disciplines concernées sont l'analyse des données (analyse discriminante, analyse en composantes principales, méthodes de regroupement), la théorie de la régularisation, la théorie des automates, la vision par ordinateur [Hrycej 1992]. Dans toutes ces disciplines on retrouve des algorithmes très proches (soit du point de vue des tâches réalisées, soit même du point de vue des calculs effectués) de ceux qui sont implémentés sous forme de réseaux.

Plusieurs questions importantes se posent donc au sujet de ces algorithmes implémentés sous forme de réseaux. Sont-ils meilleurs que les autres algorithmes développés précédemment en diverses branches des mathématiques ? Permettent-ils de résoudre des problèmes insolubles jusqu'alors ? Leur implémentation neuronale leur donne-t-elle des propriétés supplémentaires, nouvelles ?

Comme nous le verrons dans le paragraphe 1.3.7, ces questions sont encore ouvertes. Toutefois, avant d'essayer d'apporter des éléments de réponse à ces problèmes d'actualité, nous allons présenter les trois principaux modèles de réseaux de neurones artificiels que nous utilisons dans les chapitres suivants.

1.3.3 RNA multi-couches à rétro-propagation du gradient

Note : dans la suite du mémoire, nous parlerons souvent par abus de langage de « réseaux multicouches » au lieu de « réseaux multi-couches à rétro-propagation du gradient ».

Ce sont les réseaux les plus utilisés dans les applications, et ils ont beaucoup contribué à l'intérêt actuel pour les réseaux. Cette présentation s'inspire de [Bourret, Reggia, et Samuelides 1991].

Les unités de ces réseaux sont organisées en n ($n >= 3$) **couches** successives, comme sur la figure 1.1. Chaque unité ne peut être connectée qu'aux unités des couches suivantes, et l'information ne circule que dans un seul sens durant la phase d'utilisation, c'est pour cela qu'on parle aussi de réseaux *feedforward*. Les couches qui se trouvent entre la couche de sortie et celle d'entrée sont dites **cachées**.

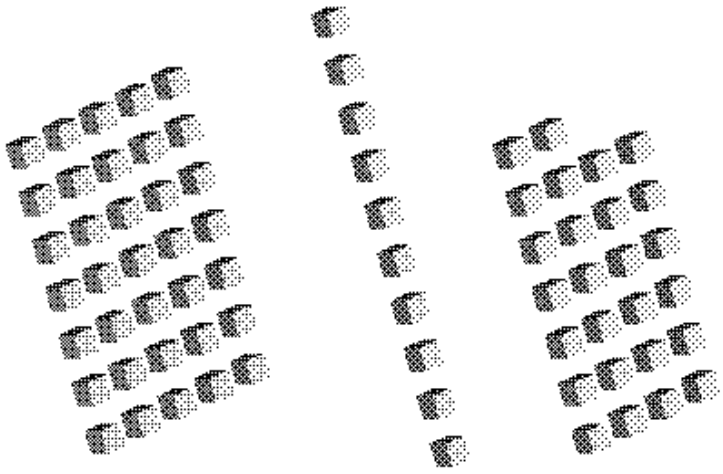


Fig. 1.1 - *Un exemple de réseau à trois couches, produit par le simulateur SNNS [Zell et al. 1993]. À gauche, la couche des unités d'entrée, qui reçoit une image de lettre de l'alphabet discrétisée en 7x5 pixels : chaque unité a pour activation une valeur réelle comprise entre 0 et 1 (0 = pixel blanc, 1 = pixel noir). Au milieu, la couche des unités cachées, dont le nombre a été déterminé empiriquement. À droite, la couche des 26 unités de sortie : chaque unité calcule une activation réelle entre 0 et 1, et l'apprentissage doit faire en sorte que pour chacune des 26 lettres de l'alphabet, une seule unité ait l'activation 1 et toutes autres l'activation 0. Les connexions ne sont pas indiquées car trop nombreuses : chaque unité d'entrée est connectée à toutes les unités cachées, chacune d'entre elles étant de même connectée à toutes les unités de sortie, ce qui fait* $35 \cdot 10 + 10 \cdot 26 = 610$ *connexions, chacune ayant un poids.*

Chaque unité i possède une **activation** réelle $a_i(t)$, qui est fonction de son entrée $e_i(t)$ calculée à partir des activations des neurones j des couches précédentes qui lui sont connectés :

$$a_i(t) = f(e_i(t)) \text{ avec } e_i(t) = \sum_j w_{ij}(t) a_j(t) \ + \ \theta_i$$

où $\begin{cases} w_{ij}(t) & \text{est le poids de la connexion entre les neurones } i \text{ et } j \\ \theta_i & \text{est un seuil propre à chaque neurone} \\ f & \text{est une fonction dite de transfert, différentiable et croissante,} \\ & \text{par exemple une sigmoïde comme } f(x) = \frac{1}{1+e^{-x}} \end{cases}$

Outre les sigmoïdes, de nombreuses fonctions de transfert peuvent être utilisées : le simulateur SNNS [Zell et al. 1993] en propose une vingtaine.

Nous allons maitenant préciser comment un tel réseau s'utilise. Il faut en fait distinguer deux phases.

Phase d'utilisation

Dans cette phase, le réseau calcule simplement les activations de ses neurones de sortie. Pour cela, l'utilisateur fixe les activations des neurones d'entrée, puis laisse le réseau effectuer le calcul des activations des unités cachées et de sortie, couche par couche. L'utilisateur exploite ensuite les activations des unités de sortie de la manière qui lui convient.

Remarque : dans la suite du document, nous appelons sorties du réseau les activations des unités de sortie du réseau.

Phase d'apprentissage

Les poids d'un tel réseau sont initialisés de manière aléatoire lorsque le réseau est créé. Les premières activations de sortie sont donc également aléatoires, et il est nécessaire de régler (ou encore apprendre) les valeurs des poids[4] jusqu'à obtenir les sorties désirées. L'utilisateur doit pour cela disposer d'un ensemble de couples (valeurs d'activation d'entrée, valeurs d'activation de sortie désirées). Ces couples sont appelés **exemples**, et l'apprentissage est dit supervisé car l'utilisateur est capable de dire quelle sortie il désire pour chacune des entrées.

Appelons $\{a_i(n)\}$ les sorties observées lors de la présentation du n-ième exemple, et $\{s_i(n)\}$ les sorties désirées pour cet exemple. Le but de l'apprentissage est de **minimiser** l'erreur quadratique observée sur cet exemple, $Q(n) = \sum_i [a_i(n) - s_i(n)]^2$.

Pour cela on fait évoluer les poids w_{ij} dans la direction indiquée par le gradient de $Q(n)$, ce qui donne $\Delta w_{ij}(n) = -\eta \frac{\partial Q(n)}{\partial w_{ij}}$, où $\eta \in]0,1[$ est une constante. Notons que les poids sont modifiés après chaque présentation d'exemple, ce qui ne constitue pas une véritable méthode de gradient[5] (les poids devraient être modifiés seulement

4. Il y a en fait un autre paramètre, le seuil θ_i, mais il peut être considéré comme un poids particulier reliant l'unité courante à une unité fictive ayant toujours 1 comme activation.

5. D'après [Bourret, Reggia, et Samuelides 1991], il s'agit d'une méthode de gradient stochastique, qui a des avantages (l'algorithme a peu de chances de se fixer dans des états minimaux peu accentués) et des inconvénients (moins bonne qualité de convergence que dans le cas du gradient total).

après la présentation de tous les exemples), mais est plus naturel.

Il faut maintenant calculer $\frac{\partial Q(n)}{\partial w_{ij}}$, et pour simplifier on supprime le numéro n de l'exemple, et on fait rentrer le facteur 2 résultant de la dérivation dans la constante η. D'après la règle de dérivation des fonctions composées,

$$\frac{\partial Q}{\partial w_{ij}} = \frac{\partial Q}{\partial a_i}\frac{\partial a_i}{\partial e_i}\frac{\partial e_i}{\partial w_{ij}}$$

D'après la définition de e_i, on obtient :

$$\frac{\partial e_i}{\partial w_{ij}} = a_j.$$

Posons

$$\delta_i = -\frac{\partial Q}{\partial a_i}\frac{\partial a_i}{\partial e_i}.$$

Cette quantité est appelée signal d'erreur de l'unité i, c'est en fait la contribution de l'entrée de l'unité i à l'erreur quadratique.

Avec cette notation, on obtient

$$\begin{aligned}\frac{\partial Q}{\partial w_{ij}} &= \delta_i a_j \\ \Delta w_{ij}(n) &= -\eta\delta_i a_j\end{aligned}$$

Si i caractérise une unité de sortie, on obtient par dérivation directe $\frac{\partial Q}{\partial a_i} = -(s_i - a_i)$, et de même $\frac{\partial a_i}{\partial e_i} = f'(e_i)$, d'où finalement

$$\delta_i = f'(e_i)(s_i - a_i)$$

Si i caractérise maintenant une unité cachée, le calcul de $\frac{\partial Q}{\partial a_i}$ est plus complexe. Soit k l'indice des unités de la couche suivant celle de notre unité cachée i. En fait Q dépend des a_k, qui dépendent de a_i. On obtient alors

$$\begin{aligned}\frac{\partial Q}{\partial a_i} &= \sum_k \frac{\partial Q}{\partial a_k}\frac{\partial a_k}{\partial a_i} \\ \text{puis } \frac{\partial Q}{\partial a_i} &= \sum_k \frac{\partial Q}{\partial a_k}\frac{\partial a_k}{\partial e_k}\frac{\partial e_k}{\partial a_i} \\ \text{et enfin } \frac{\partial Q}{\partial a_i} &= -\sum_k \delta_k w_{ki}\end{aligned}$$

D'où l'on déduit

$$\delta_i = f'(e_i)\sum_k \delta_k w_{ki}$$

On peut remarquer que, quelque soit la phase, une unité i peut effectuer tous ses calculs de manière **locale**, c'est-à-dire en ayant seulement accès aux unités voisines (couche précédente, couche suivante). on retrouve la propriété de localité des calculs que nous avons mentionnée plus haut.

D'autre part, le calcul des signaux d'erreur se fait d'abord pour les unités de la couche de sortie, disons de numéro p, on en déduit ensuite les signaux de la couche précédente $p-1$ (car ils sont fonction des signaux de la couche p), puis ceux de la couche $p-2$, ..., jusqu'à la couche d'entrée. Cette propagation d'information des sorties vers les entrées a conduit à appeler cette méthode **rétro-propagation du gradient**.

Bilan

Nous avons présenté seulement l'algorithme de base de la rétro-propagation, simple mais peu performant. Il existe de nombreuses variantes (rétro-propagation avec moment, rétro-propagation avec batch, Quickprop, Rprop, etc. – voir par exemple [Zell et al. 1993]). Ces réseaux sont assez faciles à programmer et peuvent être appliqués à des problèmes de taille réaliste. Par exemple Pican et al. [1993] présentent un contrôle de laminoir avec un réseau de ce type qui apprend environ 6000 exemples. Le calcul utilise entre deux et quarante heures de CPU sur une station de travail SPARC 10, selon la taille du réseau. Notons en passant que l'une des raisons du succès des réseaux de neurones artificiels est la disponibilité d'une puissance de calcul importante sur le bureau de chaque chercheur, et ce depuis quelques années seulement.

D'autre part, ces réseaux (et d'autres modèles aussi) n'apprennent pas les exemples « par cœur » (cela peut toutefois arriver si le réseau est mal configuré), ce qui veut dire qu'ils peuvent aussi donner une bonne réponse pour des exemples qu'ils n'ont pas appris. On parle à ce propos de la **capacité de généralisation** des réseaux de neurones artificiels, particulièrement séduisante d'un point de vue cognitif. Mais cette propriété peut également être étudiée d'un point de vue mathématique. Pour cela il faut se placer dans le cadre théorique de l'approximation de fonctions [Labbi 1993], grâce auquel on connaît mieux aujourd'hui les propriétés de certains réseaux. Nous précisons ci-dessous quelques éléments de ce cadre théorique, afin de pouvoir citer le théorème 1.2.

En 1957, le mathématicien russe A. Kolmogorov a publié un théorème fondamental concernant la représentation exacte de fonctions continues par des fonctions élémentaires à une seule variable. Ce théorème peut être énoncé comme suit [Labbi 1993] :

Théorème 1.1 *Étant donnée une fonction f continue,*

$$\begin{aligned} f : [0,1]^n &\rightarrow \Re \\ X &\mapsto f(X) \end{aligned}$$

Il existe un nombre réel λ et une fonction continue et monotone h définie sur $[0,1]$

qui ne dépendent pas de f (mais dépendent de n) tels que f s'écrit sous la forme suivante :

$$\begin{aligned} f(x_1, x_2, ..., x_n) &= \sum_{k=1}^{2n+1} g(z_k) \quad \forall X(x_1, x_2, ..., x_n) \in [0,1]^n \\ avec \qquad z_k &= \sum_{i=1}^{n} \lambda^k h(x_i + k\epsilon) + k \end{aligned}$$

où ϵ est une constante positive arbitraire et g est une fonction continue à une seule variable et qui dépend de f et de ϵ.

Dans la théorie des réseaux neuronaux, ce théorème constitue une base fondamentale des réseaux de neurones artificiels à couches. En termes de réseaux, le théorème se traduit comme suit :

Théorème 1.2 *Étant donnée une fonction continue,*

$$\begin{aligned} f : \ [0,1]^n &\to \Re \\ X &\mapsto f(X) \end{aligned}$$

f peut être calculée exactement par un réseau neuronal à trois couches dont la première couche (entrée) est constituée par n unités (qui sont les n variables de f), la deuxième couche (cachée) est constituée de $(2n+1)$ unités et la troisième couche (sortie) est constituée d'une unité.

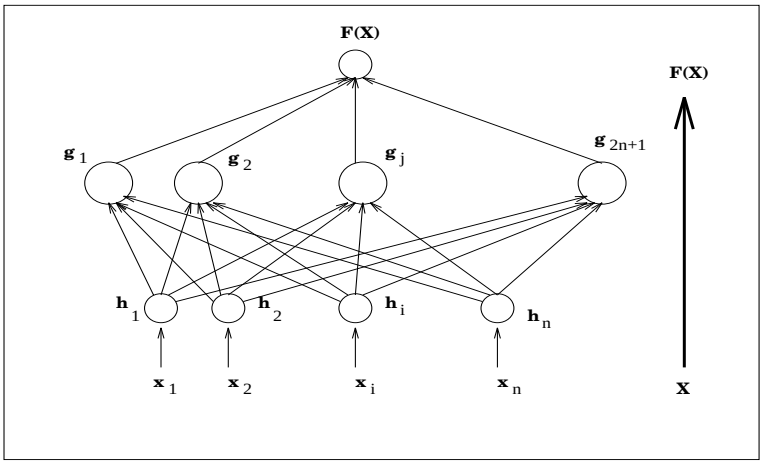


Fig. 1.2 - *D'après [Labbi 1993]. Architecture d'un réseau représentant exactement la fonction f.*

Le calcul exact de f est alors théoriquement possible par un réseau dont l'architecture est celle de la figure 1.2 où les unités de la première couche distribuent les valeurs des variables x_i sur les unités de la deuxième couche. Les unités de la deuxième couche sont connectées à toutes celles de la première et ont une entrée donnée par :

$$e_k = \sum_{i=1}^{n} \lambda^k . h(x_i + k\epsilon) + k$$

Ces unités ont toutes la même fonction de transfert g.

Au niveau de la troisième couche, la sortie est simplement la somme des sorties des unités de la deuxième couche.

$$f(X) = \sum_{j=1}^{2n+1} g(e_j)$$

Malheureusement, on ne connaît à présent aucun type de fonctions spécifiques (h et g) ni de constantes ϵ qui permettent de calculer de manière explicite la valeur exacte de f. Ceci est dû au fait que le théorème de Kolmogorov est un théorème d'existence seulement et sa démonstration n'est pas constructive dans le sens où les expressions explicites des fonctions h et g ne sont pas nécessaires à sa démonstration. Cependant, il constitue un outil théorique fondamental pour justifier l'approche *neuronale* de l'approximation de fonctions continues. Cet outil a permis une justification théorique solide de l'algorithme de la rétropropagation du gradient.

Dans un tel cadre théorique il est possible de définir et étudier précisément la notion de généralisation [Labbi 1993], mais cela sort du cadre de notre étude.

Toutefois ces réseaux présentent des inconvénients importants [Jutten 1991] :

- on ne sait pas combien d'unités il faut mettre sur chaque couche cachée, ni d'ailleurs combien il faut de couches cachées (en théorie trois suffisent, mais dans la pratique il est parfois nécessaire de faire des essais avec plusieurs couches cachées),
- l'architecture est figée, dans le sens où il est difficile d'ajouter de nouvelles unités après apprentissage,
- actuellement, on se sait pas dire facilement ce que représente chaque unité cachée,
- l'apprentissage est très long, et doit être entièrement refait si l'on souhaite apprendre un nouvel exemple.

Dans la section ci-dessous nous présentons des réseaux qui ont une architecture évolutive et qui permettent d'apprendre de manière incrémentale.

1.3.4 RNA incrémentaux à base de prototypes

Nous présentons ici un autre type de réseaux, les réseaux à base de prototypes, qui sont particulièrement étudiés à Grenoble, au LIFIA [Azcarraga 1993, Giacometti 1992], ainsi qu'au LTIRF [Jutten 1991], et ailleurs [Alpaydin 1991]. Ces réseaux, qui servent principalement à regrouper et classer des vecteurs de caractéristiques numériques, ont deux avantages principaux par rapport aux réseaux présentés ci-dessus :

ils fonctionnent sans supervision (la sortie désirée n'est pas fournie au réseau, et n'est donc pas utilisée dans la règle d'apprentissage), et d'autre part leur architecture peut être rendue évolutive, dans le sens où le nombre d'unités cachées peut augmenter ou diminuer. Pour cette raison, on parle de réseaux **incrémentaux**.

Les réseaux présentés ici sont également représentatifs d'une large gamme de modèles de réseaux utilisant une forme d'**apprentissage par compétition**, comme ceux de Grossberg ou Kohonen par exemple [Azcarraga 1993]. Toutefois ces autres modèles ne sont pas tous incrémentaux.

Architecture

L'architecture générale de ces réseaux est présentée sur la figure 1.3. On y distingue plusieurs couches de neurones : couche d'entrée, couche de sortie, couche de prototypes avant « *winner-take-all*[6] » et couche de prototypes après « *winner-take-all* ». À cause de cette sélection du prototype le plus activé par le « *winner-take-all* », on parle de compétition entre les prototypes, d'où le nom de la méthode d'apprentissage.

Par abus de langage, nous appellerons *prototype i* une unité U_i des couches cachées, alors qu'il faudrait parler du prototype représenté par l'unité en question.

Pourquoi dit-on que ces unités représentent des prototypes ?

La figure 1.3 montre que chacune de ces unités U_i est reliée à toutes les unités de la couche d'entrée, si bien qu'à chaque U_i est associé un vecteur de « poids » qui a le même nombre de composantes que le vecteur d'entrée qui sera présenté au réseau. En fait, ces « poids » ne sont pas des efficacités synaptiques comme dans le modèle de neurone formel que nous avons présenté plus haut : le vecteur de « poids » appartient au même espace vectoriel que le vecteur d'entrée. En d'autres termes, le vecteur de « poids » est **un vecteur particulier de l'espace d'entrée**, et c'est là une différence essentielle avec les réseaux présentés dans le paragraphe 1.3.3.

De plus, à chaque U_i est associé un seuil de vigilance θ (éventuellement variable). On peut alors dire que chaque U_i possède une **hyper-sphère** d'influence dans l'espace d'entrée, caractérisée par son centre (le vecteur de poids) et un rayon fonction de θ (voir figure 1.4). La figure 1.5 définit quant à elle la région d'activation d'une unité U_i. Chaque unité U_i regroupe donc un certain nombre de vecteurs de l'espace d'entrée, ceux qui appartiennent à sa région d'activation, et on dit qu'elle représente un **prototype** de ces vecteurs.

Fonctionnement

Le fonctionnement d'un réseau incrémental à base de prototypes se décrit facilement de manière algorithmique. Le tableau 1.1 décrit le fonctionnement du classifieur du

6. Il s'agit d'une méthode neuronale basée sur un principe de compétition. Cette méthode permet de sélectionner dans un groupe de neurones celui qui possède la plus grande activation : après une phase de convergence, tous les autres neurones ont 0 pour activation, et celui-ci a 1 pour activation, ce qui permet de l'identifier. Voir [Kaski et Kohonen 1994] pour plus de détails.

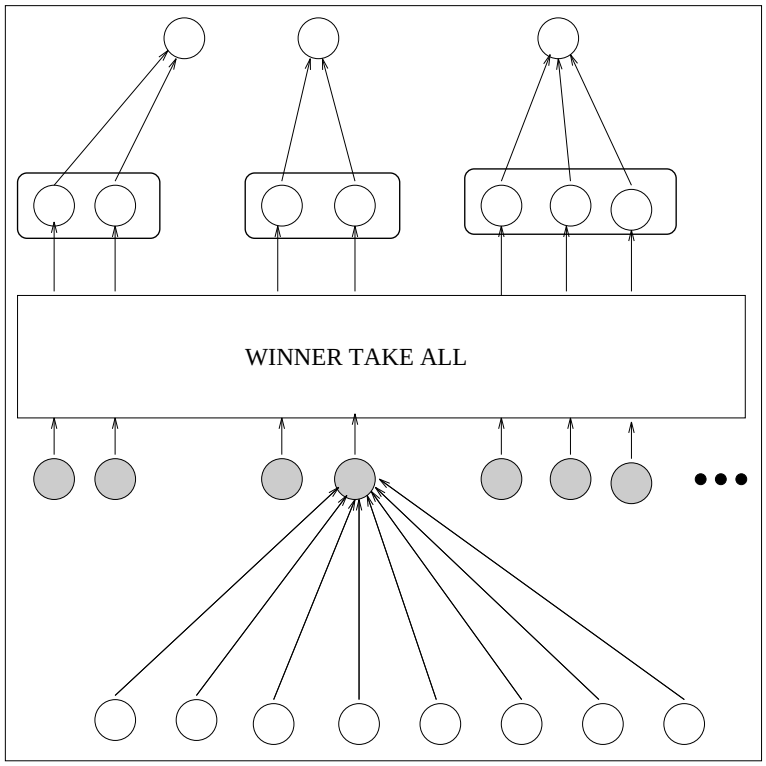


Fig. 1.3 - *Architecture d'un réseau à base de prototypes. Chaque unité de la couche des prototypes est connectée à toutes les unités de la couche d'entrée. La couche de « WINNER-TAKE-ALL » sert à sélectionner, par la méthode neuronale portant ce nom, l'unité prototype la plus activée. En sortie du « winner-take-all », une seule unité prototype est activée. Chaque unité de la couche de sortie est connectée à un certain nombre d'unités prototypes seulement. La couche d'entrée comporte autant d'unités que le vecteur que l'on souhaite classer a de composantes, et la couche de sortie a autant d'unités qu'il y a de classes possibles. L'apprentissage porte uniquement sur les poids des connexions entre la couche d'entrée et la couche des prototypes, car les poids des connexions entre la couche des prototypes et la couche de sortie sont fixes et valent 1 (ce qui traduit simplement la présence d'une connexion). NB : les • • • indiquent que le nombre de prototypes peut varier.*

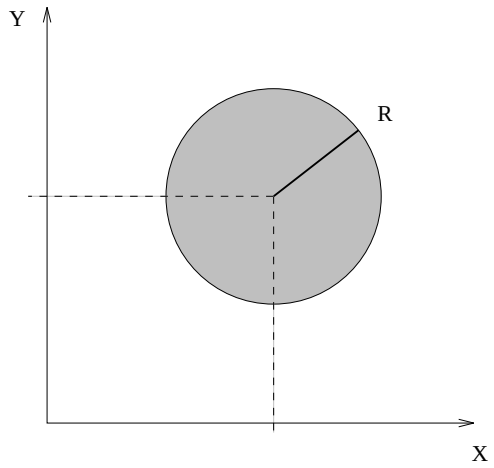


Fig. 1.4 - *La* **région d'influence** *d'un prototype i, c'est-à-dire l'ensemble des vecteurs pour lesquels le prototype a une entrée $e_i(t)$ non nulle. Ici on est dans le cas particulier où l'espace d'entrée est $\mathbb{R}^2$ et où $M(X,Y)$ (voir table 1.1) est une fonction décroissante de la distance euclidienne $d(X,Y)$. Cette région est alors un disque (ou une hyper-sphère dans $\mathbb{R}^n$) de centre le vecteur de poids W_i^t, et de rayon $R = f^{-1}(\theta)$, avec $M(X,Y) = f(d(X,Y))$. R est appelé* **rayon d'influence**.

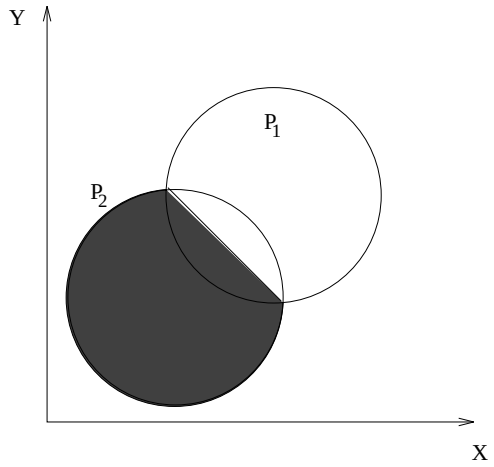


Fig. 1.5 - *On parle aussi de la* **région d'activation** *d'un prototype, qui est l'ensemble des vecteurs pour lesquels l'activation $a_i(t)$ du prototype est non nulle. Elle dépend de la compétition avec les autres prototypes, et est incluse dans la région d'influence. La figure donne un exemple dans le cas où l'on a seulement deux prototypes : la zone grisée est la région d'activation de P_2, tandis que la zone en blanc (incluse dans le disque de P_1) est celle de P_1 .*

modèle PB, Inc. [Azcarraga 1993], et nous verrons dans le chapitre 3 la variante de cet algorithme qui est utilisée dans le système SYNHESYS (remplacement de l'apprentissage non-supervisé par un apprentissage supervisé, utilisation de rayons variables).

1. présenter un vecteur d'entrée X^t,
2. pour tout prototype P_i, calculer la similitude entre son centre W_i^t et X^t : $m_i^t = M(X^t, W_i^t)$,
3. établir l'ensemble $E^t = \{P_i \mid m_i^t \geq \theta\}$,
4. - **si** $E^t = \emptyset$
     - X^t ne peut pas être classé,
     - créer un nouveau prototype P_n, avec $W_n^t \leftarrow X^t$
   - **sinon**
     - choisir $P_s \in E^t$ tel que $m_s = \max_{P_i \in E^t} \{m_i^t\}$,
     - classer X^t sous la même classe que P_s,
     - **rapprocher le centre** de P_s de X^t : $W_s^{t+1} = W_s^t + g(t)(X^t - W_s^t)$

Tab. 1.1 - *Algorithme décrivant le fonctionnement du classifieur du modèle PB, Inc., d'après [Azcarraga 1993]. L'apprentissage est* **non-supervisé** *car durant l'apprentissage le réseau ne connaît pas la sortie désirée pour le vecteur présenté. La classification nécessite par conséquent une phase d'***étiquetage***, durant laquelle l'utilisateur doit identifier « à la main » la classe à laquelle chaque prototype appartient. Cette phase d'étiquetage manuelle revient en fait à créer les connexions de poids 1 entre la couche cachée et la couche de sortie (fig. 1.3). L'aspect le plus original de cet algorithme est sans doute la loi d'ajustement des poids, dite loi de Grossberg, qui garantit la convergence de l'algorithme si certaines conditions sur $g(t)$ sont remplies. θ est un paramètre constant du réseau. Cet algorithme est très proche d'algorithmes statistiques (voir paragraphe 1.3.7).*

Réalisation neuronale

Si l'on cherche à réaliser sous forme de réseau de neurones artificiels l'algorithme du tableau 1.1, il suffit de dire que chaque unité i représentant un prototype possède une **activation** réelle $a_i(t)$, qui est fonction de son entrée $e_i(t)$:

$$e_i(t) = \begin{cases} 0 & \text{si } M(X^t, W_i^t) \geq \theta \\ M(X^t, W_i^t) & \text{sinon} \end{cases}$$

$$a_i(t) = \begin{cases} 1 & \text{si } e_i(t) = \max_j \{e_j(t)\} \\ 0 & \text{sinon} \end{cases}$$

Dans le cas d'une réalisation neuronale, il faut aussi se préoccuper du calcul des activations $a_i(t)$, qui dépendent d'un calcul de maximum ($\max_{P_i \in E^t} \{m_i^t\}$ dans l'algorithme). Or ce calcul est possible avec une méthode neuronale de type « winner-take-all » comme nous l'avons indiqué sur la figure 1.3. Dans ce cas, il y a réellement une forme de compétition entre les prototypes.

Bilan

Les données d'entrée de ces réseaux sont des vecteurs de $\mathbb{R}^n$. Or dans les applications on dispose souvent de données qualitatives, booléennes, ordinales ou nominales. Si on veut employer ces réseaux, il faut trouver un codage permettant de se ramener à des valeurs numériques. Dans le cas des booléens, il suffit de prendre comme convention *0 représente VRAI, 1 représente FAUX.* Dans le cas des variables ordinales une suite d'entiers peut convenir. Dans le cas des variables nominales (par exemple une variable pouvant prendre pour valeurs rouge, vert ou bleu) plusieurs solutions existent. Par exemple on peut prendre la convention *0 représente rouge, 1/2 représente vert, 1 représente bleu.* Mais dans ce cas on suppose implicitement que bleu est plus proche de vert que de rouge, ce qui est discutable... On peut résoudre ce premier problème en codant ces trois valeurs possibles à l'aide de trois unités d'entrée, et avec la convention *1 0 0 représente rouge, 0 1 0 représente vert, 0 0 1 représente bleu.* Avec la distance euclidienne, les trois couleurs sont alors à égale distance les unes des autres.

Toutefois, dans les applications on a souvent des données hétérogènes, c'est-à-dire de types différents. La distance euclidienne n'est pas nécessairement bien adaptée, et il faut être capable de définir une distance pour chaque problème. De plus, les données d'entrée n'ont pas toutes la même importance, et la distance que l'on construit doit éventuellement pouvoir le refléter, avec par exemple une pondération appropriée. Enfin, en pratique il est rare que l'on dispose de toutes les données d'entrée, mais ces réseaux n'apportent pas de solution naturelle à ce problème.

Ces réseaux sont donc confrontés à des problèmes classiques de l'analyse de données.

1.3.5 RNA récurrents

Dans la suite nous appelons état d'un réseau le vecteur des activations de ses neurones numérotés arbitrairement. Nous notons X cet état.

Dans les réseaux présentés ci-dessus, le temps n'intervient pas : les entrées présentées n'ont pas de relation temporelle entre elles, et l'on ne s'intéresse pas à l'évolution dans le temps des états des neurones. Mais on ne peut pas les utiliser directement pour traiter des applications dans lesquelles le temps joue un grand rôle, telles que la classification de séquences ou de trajectoires, la prédiction temporelle, etc. Aussi nous allons nous intéresser ici à une classe particulière de réseaux, celle des réseaux récurrents, qui permet de prendre en compte le temps de plusieurs manières. Étant donné que le graphe orienté des connexions de ces réseaux comporte des boucles, la propagation des activations est plus complexe et il est indispensable de s'intéresser au comportement de ces réseaux qui sont des systèmes dynamiques [Labbi 1993].

En général, les unités de ces réseaux mettent à jour leurs états d'activation en suivant une dynamique d'évolution décrite par un système d'équations différentielles non-linéaires de la forme suivante :

$$\frac{dX}{dt} = F(X, E)$$

avec $X \in I\!R^n$, $E \in I\!R^m$ et F définie de $I\!R^n \times I\!R^m$ vers $I\!R^n$.

Ayant fixé une **condition initiale** $X(0)$ et les **paramètres externes** E (les **entrées** du réseau), l'évolution du réseau peut être déterminée et son comportement asymptotique peut être prévu en analysant le comportement de X en fonction du temps. La majorité des réseaux récurrents étudiés possèdent une dynamique convergente ou périodique, c'est-à-dire que le comportement asymptotique du réseau se stabilise soit sur un état d'équilibre stationnaire (attracteur ponctuel), soit sur une orbite périodique (cycle limite).

On distingue notamment les réseaux dits isolés ayant une dynamique où les paramètres E sont fixés avec les conditions initiales $X(0)$ et les réseaux dits non-isolés où les paramètres E peuvent varier avec le temps en fonction de l'environnement externe et indépendamment des conditions initiales $X(0)$ [Labbi 1993].

Réseaux isolés

Les réseaux les plus étudiés sont les réseaux isolés, dans lesquels l'entrée n'est prise en compte qu'au début de la propagation d'activation. Dans cette classe on trouve notamment l'important modèle de Hopfield, qui permet de traiter des problèmes d'optimisation [Hecht-Nielsen 1989].

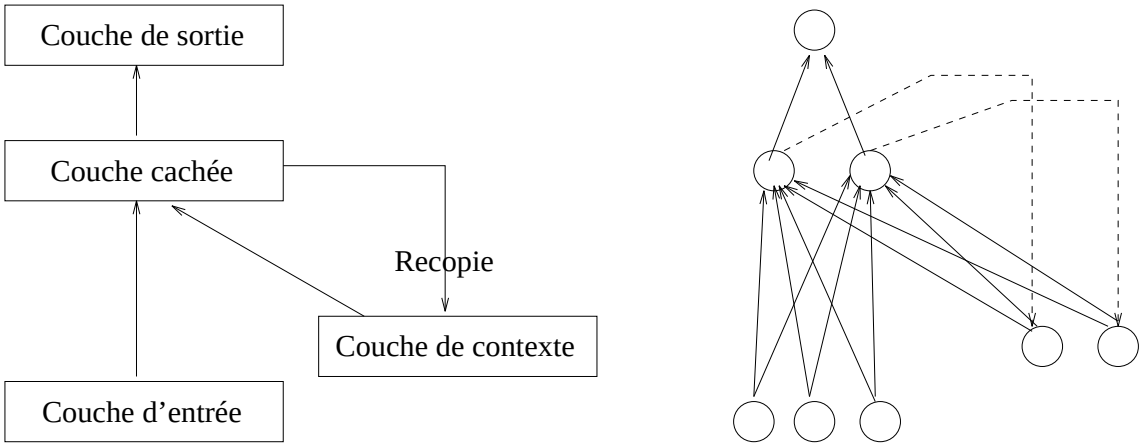
Nous nous limitons ici à une classe très particulière de réseaux récurrents isolés, les réseaux récurrents à couches, qui sont une simple extension des réseaux à couches, et dans cette classe nous nous limitons au modèle SRN (*Simple Recurrent Network*) de Elman [1990]. Ces réseaux ont l'intérêt de pouvoir être entraînés soit avec la rétropropagation standard, très simple, soit avec des versions de la rétroprogation adaptées au cas où les entrées du réseau sont une séquence temporelle (*Back-Propagation Through Time* et ses variantes) [Jodouin 1993].

L'architecture de ces réseaux est représentée sur la figure 1.6. La couche de contexte permet de donner au réseau une mémoire, ce qui rend possible certains traitements prenant en compte le temps, comme la prédiction des mots d'une séquence de mots [Elman 1990], ou encore la génération de phrases simples décrivant le mouvement d'une boule de billard [Bartell et Cottrell 1991].

Le principal inconvénient de ces réseaux est le coût très élevé des calculs effectués par les méthodes d'apprentissage.

Réseaux non-isolés

Dans les réseaux non-isolés, les paramètres E peuvent varier avec le temps et indépendamment des conditions initiales $X(0)$. Les rôles de $X(0)$ et E sont alors très différents. E détermine la dynamique d'évolution du système ; $X(0)$ représente la valeur initiale de la trajectoire d'évolution. Quand E change, on se trouve dans un système dynamique différent. En revanche, quand E est fixé, une valeur différente de $X(0)$ représente une valeur initiale d'une autre trajectoire mais du même système dynamique.



(a) SRN : Modèle général (b) SRN : exemple

Fig. 1.6 - *Le modèle de réseau récurrent SRN de Elman [1990]. Au temps t, l'état de la couche de contexte est égal à celui de la couche cachée au temps t-1. Ces réseaux sont entraînés par rétropropagation sans tenir compte des connexions de recopie dont le poids est toujours égal à 1.*

Considérons maintenant que $X(0)$ est fixé. E détermine donc la dynamique du réseau. Si le réseau converge, E détermine les attracteurs correspondant à cette dynamique. Si E est changé en E', alors le réseau a une nouvelle dynamique et donc de nouveaux attracteurs. Si maintenant E est ramené à sa valeur initiale, on retourne à la dynamique de départ, mais on n'est pas assuré de retomber sur l'état d'équilibre précédent.

Or on souhaite pouvoir utiliser ces réseaux comme classifieurs, car ils peuvent alors constituer un intéressant modèle de la prise de décision experte en temps réel [Labbi 1993]. Pour cela, ces réseaux doivent être en mesure de retomber sur un même état d'équilibre pour un E donné. Cela est possible si l'on impose à la dynamique du réseau d'être globalement convergente pour tout E fixé. Cela permet de définir une application de l'espace d'entrée (données provenant de l'environnement) vers l'espace de sortie (espace des décisions) : pour un état donné, on obtiendra une seule et unique décision.

Le modèle de réseau que nous utiliserons dans le chapitre 5 a été étudié de manière approfondie par Labbi [1993]. Il est défini par le système d'équations différentielles ci-dessous, qui peut être réalisé par le réseau de la figure 1.7.

$$\begin{aligned} \frac{dx_i}{dt} &= f_i(X,E) \\ &= -c_i.x_i + (1-x_i).(\sum_{k=1}^{m} w_{ik}.f_k(e_k)) - (1+x_i).(\sum_{j=1}^{n} d_{ij}.g_j(x_j)), \\ & \quad i = 1,...,n \end{aligned}$$

avec :

- $X \in [-1,1]^n$ variable d'état du réseau,
- $E \in [0,1]^m$ variable d'environnement,
- c_i coefficient de fatigue associé à x_i,
- $w_{ik} \geq 0$ poids des connexions excitatrices,
- d_{ij} poids des connexions inhibitrices,
- f_k, g_j sont des fonctions monotones positives que l'on peut identifier à des fonctions sigmoïdales.

A. Labbi a notamment démontré que :

- pour tout E, la dynamique du modèle est convergente
- pour tout E, le système d'équations différentielles précédent est asymptotiquement stable si l'on utilise des fonctions g_j de la forme :

$$g_j(x) = \frac{1}{1 + exp(-x)},\ j = 1, 2, ..., n$$

et si la condition suivante est respectée :

$$\min c_i > \max \sum_{j=1}^{n} |d_{ij}|,\ i \in 1, 2, ..., n$$

où les d_{ij} représentent les poids des liens d'inhibition.

Un réseau ayant cette propriété permet de définir une application, au sens mathématique, de l'espace d'entrée (environnement) vers un ensemble de classes ou d'associations (attracteurs) dans l'espace de décision.

Le principal inconvénient de ces réseaux est qu'aucune méthode d'apprentissage ne permet pour le moment de déterminer les différents coefficients (les c_i, d_{ij} et w_{ik}) qui doivent être déterminés empiriquement. Une méthode d'apprentissage est en cours d'étude au LIFIA.

1.3.6 Points forts et points faibles des réseaux de neurones artificiels

Les trois types de modèles que nous venons de décrire présentent des avantages et des inconvénients qui vont permettre d'illustrer l'analyse des points faibles et forts des réseaux de neurones artificiels en général que nous développons maintenant.

Points forts

Une des caractéristiques importantes des réseaux est leur parallélisme massif, qui permet leur réalisation sur machine parallèle généraliste ou dédiée (par exemple un

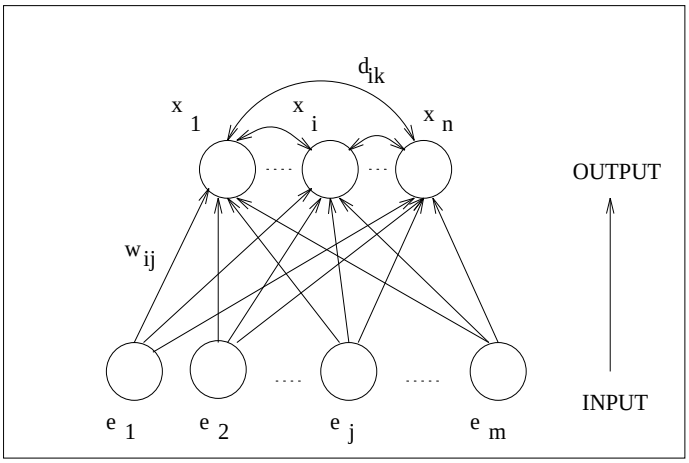


Fig. 1.7 - *D'après [Labbi 1993]. Un réseau permettant de réaliser le système d'équations différentielles* $\frac{dx_i}{dt} = f_i(X, E)$.

circuit intégré). Dans ce dernier cas, la réalisation est facilitée par le fait que les réseaux sont généralement composés d'un grand nombre d'unités simples souvent identiques. Par exemple le modèle COLD, Inc. [Azcarraga 1993], qui permet de reconnaître des chiffres manuscrits, comporte 3 338 unités toutes identiques.

Par conséquent, on peut généralement créditer un réseau de neurones artificiels d'une excellente adéquation entre l'algorithme qu'il met en œuvre et une architecture matérielle [Beaudot 1994], au moins potentielle. Cet avantage est potentiel car les réalisations matérielles sont encore peu nombreuses. Un exemple d'une telle réalisation est donné par Masa, Hoen, et Wallinga [1993] qui décrivent un circuit intégré VLSI en technologie CMOS. Ce circuit réalise un réseau à trois couches, comportant respectivement 70, 4 et 1 unités, et qui classe un vecteur de $I\!R^{70}$ en 50ns, soit 20 millions de vecteurs par seconde. Le circuit réalisé n'occupe qu'une surface de 6,5x4 mm^2, et sera intégré dans un système d'acquisition de données pour le synchrotron DESY (*Deutsches Elektronen SYnchrotron*) de Hambourg. Mais pour le moment, la plupart des utilisateurs doivent simuler les réseaux sur des ordinateurs séquentiels. Les simulations ont pour inconvénient de faire oublier les contraintes liées au parallélisme. La conséquence est que certains réseaux ainsi obtenus sont difficilement réalisables sur matériel parallèle.

Ce parallélisme intrinsèque permet des exécutions très rapides, et cela est très intéressant dans toutes les applications temps réel, notamment la robotique, d'autant plus que la possibilité de réaliser des réseaux en circuits intégrés permet de les embarquer facilement.

Une deuxième caractéristique intéressante des réseaux est leur robustesse. Par robustesse, on entend généralement :

- la résistance aux pannes des neurones : dans les réseaux distribués, la redondance des informations peut conduire le réseau à bien fonctionner même quand des unités sont en panne. Par exemple, Belala [1992] « distribue » des règles sous forme de réseau de neurones artificiels, selon différentes méthodes, et montre que dans certains cas on peut obtenir de bonnes réponses avec 40 %

des unités en panne. Il s'agit là aussi d'un avantage intéressant pour les systèmes embarqués ;

- la résistance au bruit : de nombreux modèles de réseaux donnent de bons résultats quand leurs entrées sont bruitées. Par exemple Lippman [1987] donne un exemple avec un réseau de Hopfield qui reconstitue des images de chiffres bruitées (application de type mémoire associative). Azcarraga [1993] montre que son classifieur neuronal COLD, Inc. résiste bien au bruit : le classifieur classe des images de chiffres manuscrits bruitées aléatoirement par inversion de pixels de 0 à 1 ou de 1 à 0. Le niveau de bruit $B \in [0, 1]$ est la probabilité d'inversion de chaque pixel. Azcarraga [1993] obtient les résultats suivants :

B	Taux de bon classement (%)
0.00	99.0
0.05	98.8
0.10	97.2
0.15	93.8
0.20	85.8
0.25	70.2

Un troisième point fort est l'apprentissage, même s'il n'est pas fondamentalement différent d'autres méthodes d'estimation de paramètres. L'apprentissage est intéressant comme alternative à l'acquisition des connaissances.

Par rapport aux problèmes posés par les applications, les réseaux de neurones artificiels présentent l'intérêt de pouvoir prendre en compte la non-linéarité (les fonctions d'activation sont non-linéaires en général), et dans certains cas de pouvoir prendre en compte le temps (voir les réseaux récurrents).

Enfin les réseaux, de par leur nature continue, ne fonctionnent pas en tout ou rien, et leurs performances ont plutôt tendance à diminuer progressivement en cas de problème (bruit, panne, entrée inconnue), si bien que l'on parle souvent de « *graceful degradation* » [Gutknecht et Pfeifer 1990], que nous traduisons par « dégradation progressive ». Cette propriété est très recherchée car les systèmes cognitifs vivants montrent une telle faculté.

Points faibles

Malgré ces propriétés intéressantes, que l'on peut d'ailleurs mettre en parallèle avec les difficultés des systèmes symboliques (vitesse et parallélisme neuronaux // lenteur symbolique, robustesse // fragilité, etc.), les réseaux présentent, en l'état actuel des recherches, plusieurs points faibles :

- le manque de transparence pour l'utilisateur : les poids des connexions d'un réseau n'ont en général pas de signification évidente, si bien qu'un réseau apparaît souvent comme une boîte noire [Medsker 1994]. Il n'est donc pas aisé d'expliquer (au sens des systèmes symboliques) le résultat produit par un réseau[7].

7. Soulignons toutefois que les réseaux ne sont pas moins transparents ni moins explicables que n'importe quelle procédure, par exemple d'analyse numérique. De plus il n'est pas nécessaire

De plus, si l'on peut montrer que certains réseaux sont capables de construire des représentations plus ou moins structurées de leur environnement, ces représentations restent opaques [Memmi 1993], et doivent être analysées avec des outils statistiques ;

- les réseaux ne sont pas faits pour travailler naturellement avec des symboles et des structures de symboles, ce qui les rend difficilement utilisables pour traiter des problèmes de haut niveau qui exigent souvent de telles structures (listes, frames, objets, etc.). Des solutions limitées existent toutefois : par exemple Pollack [1990] a développé un réseau permettant d'apprendre et de représenter des structures récursives comme des arbres, tandis que Sun [1991] et Belala [1992] présentent des méthodes d'unification connexionnistes. Mais ces méthodes sont de peu d'intérêt pratique. Dans le même ordre d'idées, les réseaux n'ont pas les bonnes propriétés cognitives comme la compositionnalité (voir le paragraphe 1.2.3), en tout cas d'après l'article célèbre de Fodor et Pylyshin [1988]. Là encore des solutions limitées existent, et les réseaux distribués peuvent montrer une forme limitée de compositionnalité, mais qui reste insuffisante d'un point de vue cognitif [Memmi 1993] ;
- un réseau donné fonctionne avec un nombre fixé d'unités d'entrée et de sortie. Cela pose d'importants problèmes de représentation et de codage quand le nombre d'entrée et de sortie est variable, ce qui arrive fréquemment dans les problèmes réels. Par exemple, dans SHADE [Orsier et al. 1994], les médecins effectuent des mesures sur des segments de nerfs, mais le nombre de segments peut être différent selon le patient ;
- les réseaux restent difficiles à composer et à combiner pour résoudre des problèmes complexes : il n'existe pas encore de bonnes méthodes pour construire et entraîner des grands réseaux, ainsi que des réseaux de réseaux. La modularité n'est en effet pas une caractéristique des méthodes neuronales. Des travaux de recherche sont effectués sur ce thème [Hrycej 1992] ;
- l'apprentissage à partir d'exemples n'est pas une panacée : en effet, le problème de l'acquisition de connaissances n'est pas complètement résolu mais plutôt déplacé, car en pratique obtenir un jeu d'exemples de bonne qualité demande beaucoup de travail ;
- divers problèmes plus techniques : certains réseaux ne sont pas encore bien analysés d'un point de vue mathématique (voir ci-dessous), tandis que d'autres sont construits de manière empirique (comme les réseaux à rétro-propagation). Les temps de calcul sont parfois très importants, alors que les réseaux réalisés restent petits (le nombre d'unités est généralement de l'ordre de la centaine ou du millier). Les conditions initiales ainsi que l'ordre de présentation des exemples peuvent avoir une grande influence sur les résultats. Travailler avec des réseaux demande donc beaucoup d'expérience, malgré leur apparente facilité d'utilisation.

La qualité parfois discutable des évaluations des travaux réalisés reste un problème important à l'heure actuelle. Il est certain que la publication de dizaines d'algo-

d'expliquer, au sens symbolique, le résultat d'un réseau dans tous les cas.

rithmes et de variantes, fréquemment mis au point sur des problèmes peu significatifs[8] rend difficile le choix d'un algorithme pour résoudre des problèmes réels. Une autre manifestation de ce problème se traduit par des allusions discrètes à la difficulté de reproduire tel ou tel résultat important (voir [Abdi 1994] au sujet des expérimentations de Hopfield sur le problème du voyageur de commerce) : en fait, les chercheurs ont pris l'habitude de faire de nombreuses expérimentations et de ne publier que les résultats de la meilleure, sans préciser s'il a fallu un, dix ou cent essais.

Une étude quantifiée de la qualité de l'évaluation a été réalisée récemment [Prechelt 1994]. Cette étude porte plus précisément sur l'évaluation expérimentale des algorithmes d'apprentissage pour les réseaux. L. Prechelt a examiné 271 articles parus entre en 1993 et en 1994 dans les deux principales revues, *Neural Networks* et *Neural Computation*, et a sélectionné les 113 articles dont la principale contribution était la conception d'un nouvel algorithme d'apprentissage destiné à être appliqué à des problèmes pratiques. Pour chacun de ces 113 articles, L. Prechelt a ensuite compté :

- le nombre de problèmes d'apprentissage différents utilisés dans l'application ; ces problèmes ont été répartis en trois classes :

  problème artificiel : les données sont générées à partir d'une simple formule logique ou arithmétique ;

  problème réaliste : les données sont générées à partir d'un modèle plus complexe possédant des propriétés voisines de ce qui existe dans la réalité ;

  problème réel : les données proviennent d'observations d'un phénomène réel ; de tels jeux de données ont la particularité de souvent contenir des erreurs et du bruit ;

- le nombre d'algorithmes plus anciens auquel le nouvel algorithme a été comparé.

Si les deux quantités mesurées sont élevées, cela ne prouve pas que la qualité des évaluations est bonne, par contre si elles sont faibles cela prouve que la qualité est mauvaise. Voici quelques résultats de l'étude :

- un tiers des articles n'utilise pas un seul problème réaliste ou réel,
- seulement 6% des articles présentent des résultats pour plus d'un problème réel,
- un tiers des articles ne présente pas de comparaison quantitative avec un autre algorithme,
- si l'on pose la définition suivante :

  > « Une évaluation d'algorithme est dite acceptable si elle utilise au moins deux problèmes réels ou réalistes et compare les résultats avec au moins un autre algorithme ».

8. Par exemple, le problème du XOR, où il s'agit d'apprendre à réaliser la fonction logique « ou exclusif ».

il se trouve que 82% des articles ne contiennent pas une évaluation acceptable, et pourtant le critère de la définition est assez modeste.

Pour L. Prechelt cette étude indique que les nouveaux algorithmes sont souvent publiés sous une forme qui ne représente pas une connaissance utile et validée. Alors qu'au contraire ils devraient apporter au minimum une réponse aux deux questions : « Pour quelles sortes de problèmes le nouvel algorithme fonctionne-t-il bien ou ne fonctionne-t-il pas bien ? » et « À quelles conditions doit-on préférer le nouvel algorithme à des algorithmes plus anciens ? ».

Nous sommes aussi concernés par ce problème de l'évaluation car les systèmes hybrides héritent directement de leur « parent » neuronal certaines caractéristiques, bonnes ou mauvaises. Ils héritent en particulier ce problème d'évaluation.

1.3.7 Réseaux et autres méthodes

En fait, le caractère parallèle des réseaux de neurones artificiels conduit certains à poser la question suivante, qui suscite une polémique grandissante actuellement : « **les réseaux sont-ils autre chose que de simples réalisations parallèles de méthodes déjà bien connues ?** ». Nous avons été confrontés à cette question en raison de nos travaux avec des géographes notamment. La question se pose pour tous les réseaux et en particulier pour les trois types de réseaux que nous avons présentés :

- les réseaux multicouches à rétro-propagation : le but est ici de minimiser l'erreur quadratique entre la réponse du réseau et la fonction f que l'on cherche à approcher. Plus précisément, si f_R est la fonction réalisée par le réseau, on cherche à minimiser

$$\bar{E}^2 = \frac{1}{N}\sum_{i=1}^{N}(f(x_i) - F_R(x_i))^2$$

  Or ce type d'erreur est utilisé dans la méthode des moindres carrés, méthode de base dans de nombreux domaines comme le contrôle, la reconnaissance des formes, la statistique [Fu 1994]. Il est donc légitime de s'interroger sur les rapports entre les méthodes neuronales et les méthodes employées dans ces domaines ;

- les réseaux à prototypes sont très proches des méthodes de regroupement employées en statistique et en reconnaissance des formes, notamment des méthodes de nuées dynamiques pour la phase de construction des prototypes, et des méthodes des k-plus-proches voisins pour la phase de reconnaissance (voir [Belaïd et Belaïd 1992] pour une présentation de ces méthodes) ;

- les réseaux récurrents, tout comme les réseaux multicouches, sont proches de méthodes employées dans le domaine du contrôle. En particulier, Nerrand et al. [1993] montrent que ces deux types de réseaux peuvent être considérés comme des filtres non-linéaires, et que les méthode neuronales et plusieurs méthodes mathématiques non-neuronales sont toutes des cas particuliers de méthodes générales d'apprentissage pour les réseaux récurrents.

Nous allons examiner quelques premières tentatives de réponse à la question ci-dessus.

Le point de vue de A. Azcarraga

Azcarraga [1993] a remarqué que l'algorithme que nous avons reproduit page 38 (tableau 1.1) est très proche de méthodes statistiques de regroupement, et il a proposé cinq éléments de comparaison que nous résumons ci-dessous :

1. les méthodes statistiques reposent souvent sur des calculs complexes, alors que chacune des nombreuses unités d'un réseau effectue des calculs très simples. D'autre part, un réseau **converge** vers un résultat, alors que les méthodes statistiques sont souvent directes.
2. les méthodes statistiques disposent de toutes les données à tout moment, alors que les réseaux ne connaissent que des « traces » des données présentées, ces traces étant stockées dans les poids de leurs connexions.
3. les classifieurs neuronaux font un traitement obligatoirement parallèle.
4. alors que certaines méthodes statistiques ne font que soit regrouper un ensemble d'objets, soit classer un nouvel objet, les classifieurs neuronaux font les deux à la fois. De plus il faut souligner que ces fonctions sont accomplies par une même structure d'unités, de connexions et de poids. Dans le modèle COLD, Inc. de A. Azcarraga, le type d'unité utilisé pour faire de la classification est en effet utilisé pour toutes les autres opérations.
5. les classifieurs neuronaux font plus que classifier. Ils transforment des données brutes en une représentation d'entrée adéquate. Dans COLD, Inc. l'image brute est en effet transformée en une configuration de détecteurs de segments.

Toutefois certains arguments nous paraissent sujets à discussion. Le point 1 en particulier, car parfois les méthodes statistiques sont itératives et non pas directes, mais cela est transparent pour l'utilisateur [Cottrell 1993]. D'autre part, la complexité de calcul évoquée est une notion assez subjective. Par rapport au point 2, on peut remarquer que certains modèles de réseaux exigent que toutes les données soient stockées quelque part, à l'extérieur du réseau (c'est le cas notamment de la rétropropagation du gradient qui doit recommencer tout l'apprentissage, pour pouvoir apprendre de nouveaux exemples, sinon il y a oubli des associations entrées/sorties précédemment apprises). Par rapport au point 5, des méthodes statistiques sont souvent utilisées pour transformer des données brutes en données convenables pour d'autres méthodes (pré-traitements), et donc cet argument ne tient pas. L'argument le plus fondamental est sans doute le quatrième, l'unicité de structure. Le classifieur de A. Azcarraga réalise effectivement toutes ses opérations avec plus de 3 000 unités toutes identiques, ce qui devrait grandement faciliter la réalisation sur circuits intégrés.

Le point de vue de M. Cottrell

D'autres auteurs se sont posés des questions analogues. Par exemple, Cottrell [1993] présente rapidement les méthodes statistiques (méthodes factorielles et méthodes de classification) puis réfléchit au point suivant : « quel peut être l'apport de techniques neuronales, parmi toutes ces méthodes largement expérimentées et disponibles ? ». Pour cet auteur, un premier apport des méthodes neuronales est la parallélisation facile, du moins en principe. Un apport plus fondamental est que certains réseaux de neurones artificiels sont des « sur-modèles » de modèles linéaires, et que l'on peut espérer faire mieux, ou au moins pas plus mal qu'avec un classique modèle linéaire (par exemple on remplace un problème de moindres carrés linéaires par un problème de moindres carrés non-linéaires). Toutefois, Cottrell [1993] mentionne deux restrictions :

- dans certains cas il a été montré qu'un réseau ne donne rien de mieux qu'une méthode non-neuronale. Toutefois les comparaisons inter-méthodes sont difficiles : on ne dispose pas en général d'indices de qualité pour les méthodes neuronales, et il est difficile de caractériser les données pour lesquelles une méthode neuronale serait supérieure. Il s'agit d'un sujet de recherche très actuel,
- les problèmes de convergence des algorithmes neuronaux sont loin d'être résolus (lenteur des calculs, minima locaux, choix délicat des paramètres).

Le problème reste donc très ouvert, et selon Cottrell [1993], avant de conclure, il faut attendre de mieux connaître les propriétés théoriques de certaines méthodes neuronales qui sont des alternatives itératives à des méthodes non-neuronales directes. Par exemple Cottrell [1993] mentionne les algorithmes du type auto-organisation de Kohonen, dans lesquels « on ne sait pas trop ce qu'on minimise, mais on obtient, grâce à la propriété de conservation topologique de cet algorithme des représentations planes très significatives des données analysées ».

Le point de vue de A. Weigend

Une contribution à cette question est également apportée par Weigend [1993], qui précise quelques différences entres les approches neuronales et statistiques :

- les modèles statistiques limitent en général l'**ordre des interactions** (par exemple des produits sont faits seulement entre deux entrées, ou trois entrées au plus), tandis que les réseaux de neurones artificiels tendent à limiter le **nombre de caractéristiques**. Ce nombre y est contrôlé par le nombre d'unités cachées et chaque unité peut représenter des interactions d'ordre élevé. Cet aspect des réseaux est intéressant dans des domaines comme la reconnaissance de caractères, où déterminer une caractéristique peut exiger de l'information provenant d'un grand nombre d'entrées (pixels).
- les règles d'ajustement dans les réseaux sont généralement **locales** (par exemple l'ajustement d'un poids ne requiert en général que des informations concernant l'erreur sur un seul exemple et des informations provenant de neurones voisins). Un autre aspect de cette localité est que la recherche d'une solution

et l'apprentissage sont effectués de manière **itérative**. Les statisticiens, en général, ne se focalisent pas sur la localité ni sur l'émergence d'une solution fonction du nombre d'itérations.

- les connexionnistes sont généralement plus généreux que les statisticiens sur le **nombre de paramètres** de leurs modèles. Ce qui rend souvent les analyses plus difficiles et augmente aussi le danger de *overfitting* (apprentissage trop parfait du jeu d'exemples, ce qui conduit à de mauvaises généralisations).

Weigend souligne aussi qu'il n'est pas surprenant que les connexionnistes aient redécouvert des problèmes déjà bien connus des statisticiens, mais que toutefois, le connexionnisme (et ses algorithmes relativement simples) permet aux chercheurs de **penser différemment les vieux problèmes et de trouver de nouvelles solutions... et de nouveaux problèmes**.

Réseaux de neurones artificiels : bilan

Les réseaux présentent donc bien des similitudes avec des méthodes mathématiques non-neuronales, notamment statistiques. Il existe toutefois des différences car les réseaux se basent généralement sur le **parallélisme** massif, la **localité** des calculs, l'obtention **itérative** d'une solution, la **non-linéarité** et une bonne **adéquation algorithme architecture**. L'apprentissage, pourtant beaucoup mis en avant dans la littérature connexionniste, n'est toutefois pas un élément de comparaison réellement significatif, car toutes les méthodes qui effectuent des réglages de paramètres au vu d'un ensemble d'exemples peuvent être créditées d'une capacité d'apprentissage au sens neuronal. Le caractère incrémental de certaines méthodes neuronales est par contre assez original.

Il reste à savoir si les méthodes neuronales sont compétitives face à ces autres méthodes, qui ont l'avantage d'être bien maîtrisées d'un point de vue mathématique. Là encore la réponse n'est pas tranchée, mais beaucoup d'auteurs insistent sur le fait que les méthodes neuronales, qui effectuent des calculs essentiellement non-linéaires, constituent des « sur-modèles » ou « super-modèles » [Hecht-Nielsen 1989, Cottrell 1993, Fu 1994] par rapport aux méthodes non-neuronales souvent linéaires. Les méthodes neuronales sont donc potentiellement plus puissantes, et permettent de travailler sur des problèmes réalistes, souvent non-linéaires. Nous avons écrit « potentiellement » car dans la pratique, il reste à trouver de bonnes méthodes d'apprentissage. Ainsi, la rétro-propagation n'est pas nécessairement la meilleure méthode pour entraîner les réseaux récurrents [Nerrand et al. 1993].

Enfin, une caractéristique des réseaux de neurones artificiels apparaît implicitement dans ce chapitre : la facilité d'utilisation par des non-mathématiciens. Cette caractéristique joue à la fois en faveur et en défaveur des réseaux. En faveur, car elle permet à des non-spécialistes d'utiliser des méthodes puissantes d'approximation, d'optimisation, etc., sans passer beaucoup de temps à comprendre ces méthodes et leurs conditions d'application. De plus, l'inspiration neurobiologique des réseaux les rend plus attractifs que d'autres algorithmes. En défaveur, car justement, la méconnaissance des conditions d'application, de la qualité des résultats, risque de conduire

à des résultats erronés. La façon idéale d'utiliser les réseaux de neurones artificiels serait de vérifier d'abord qu'il n'existe pas une méthode non-neuronale permettant de résoudre le problème, puis d'utiliser un réseau en l'absence d'une telle méthode. Une façon moins idéale, mais plus adaptée à la réalité, serait d'essayer d'abord de résoudre le problème avec un réseau, ce qui permet très vite de voir de quel type de méthodes on a besoin, puis ensuite d'utiliser les compétences d'un mathématicien spécialiste du type de méthodes en question.

1.4 Conclusion

Dans ce chapitre, nous avons présenté les trois types de réseaux de neurones artificiels que nous employons dans nos applications. Nous pouvons en retenir les caractéristiques suivantes :

1. Réseaux multi-couches à rétropropagation de l'erreur :
   - minimisation de l'erreur quadratique entre fonction réalisée par le réseau et fonction à approcher ;
   - apprentissage supervisé ;
   - facilité de programmation et traitement de problèmes de taille réaliste ;
   - capacité de généralisation, au prix d'une architecture figée et d'un apprentissage lourd.
2. Réseaux incrémentaux à base de prototypes
   - principes proches des nuées dynamiques et des k-plus-proches-voisins ;
   - apprentissage non supervisé et par compétition ;
   - architecture non figée (caractère incrémental).
3. Réseaux récurrents
   - un cas particulier de systèmes dynamiques ;
   - réseaux isolés ou non ;
   - prise en compte du temps.

La seconde contribution de ce chapitre est l'analyse des points forts et faibles des systèmes symboliques et des réseaux de neurones artificiels. Nous en donnons ci-dessous une synthèse très globale.

Les systèmes symboliques se caractérisent par une séparation des connaissances et de leur contrôle. Ils permettent la prise en compte de connaissances expertes pour résoudre des problèmes tels que le diagnostic ou la planification. Leur nature déclarative facilitant la formalisation et l'explication s'adapte bien à la représentation de connaissances structurées. Des raisonnements approximatifs sont maintenant possibles grâce à une synergie entre les mondes symboliques et numériques. Du point de vue cognitif, leurs propriétés sont la productivité, la systématicité et la compositionnalité. Notons que ces points forts sont valables quel que soit le système symbolique. Leurs points faibles, outre leur rigidité et leur fragilité sont : la diffi-

cile acquisition des connaissances ; l'absence d'amélioration des performances avec l'expérience ; la mauvaise adéquation à l'environnement réel, aux données provenant de capteurs (problème d'ancrage perceptif), aux données incomplètes ou bruitées, à l'absence de règles, aux interactions entre règles ; des temps de calcul importants (les ordinateurs ne sont pas spécialement conçus pour traiter des symboles) ; l'absence d'ancrage des symboles et des concepts.

D'inspiration neurobiologique, les réseaux de neurones artificiels se caractérisent par une bonne adéquation algorithme architecture matérielle. Ils sont utilisés dans l'approximation de fonctions pour la classification et le contrôle, la compression de données, le regroupement et la quantification, l'analyse de données, l'optimisation, etc. Les points forts d'un réseau de neurones artificiels idéal s'ajoutent à sa puissance potentielle de « super-modèle » : la redondance et la résistance aux pannes ; la dégradation progressive et la résistance au bruit ; l'apprentissage comme alternative au problème de l'acquisition des connaissances ; la prise en compte des non-linéarités et l'adéquation aux problèmes réels ; le parallélisme massif et la rapidité des calculs. La plupart des modèles neuronaux n'ont pas toutes ces qualités, par contre ils sont presque tous concernés par les points faibles suivants : manque de transparence et de capacité d'explication ; mauvaise adaptation au traitement et à la représentation de structures de symboles ; inadaptation à la résolution de problèmes de haut niveau ; compositionnalité difficile à obtenir ; nombre d'unités d'entrées et de sorties fixé ; problèmes de représentation et de codage des données ; faible modularité et difficulté de conception de réseaux de grande taille. Les méthodes d'apprentissage posent parfois des problèmes classiques de l'analyse numérique comme la sensibilité aux conditions initiales.

Cette synthèse met en évidence la complémentarité des points forts des deux types de systèmes. Toutefois les comparaisons sont délicates, en raison de la grande variété de modèles dans chaque paradigme, neuronal ou symbolique, et également parce qu'en général les grandeurs ne sont pas comparables : que penser de la « robustesse » d'un réseau de neurones artificiels qui approche une fonction continue par rapport à la « fragilité » d'un système expert d'ordre 1 ? Néanmoins, les concepteurs de SHNS veulent exploiter cette complémentarité de points forts, en apportant parfois une justification cognitive à cette démarche, comme le besoin de différents modes complémentaires d'expression des connaissances.

Nous avons mis l'accent dans ce chapitre sur le petit nombre de publications quantifiant objectivement la qualité réelle et l'utilité d'un système ou d'un algorithme, notamment en raison de la difficulté d'évaluer cette qualité et d'effectuer des comparaisons inter-méthodes. Nous rappelons enfin la nouveauté de ce domaine qui impose d'attendre notamment de mieux connaître les propriétés théoriques de certaines méthodes neuronales, qui sont des alternatives itératives à certaines méthodes plus traditionnelles, qui sont parfois novatrices, et qui résistent encore à l'analyse mathématique.

Chapitre 2

SHNS : architectures et applications spécifiques

Dans le chapitre 1, nous avons présenté les deux composants d'un SHNS. Nous allons maintenant examiner des réalisations concrètes de tels systèmes. Cela nous permettra de situer SYNHESYS, le SHNS du LIFIA, et d'essayer de répondre à un certain nombre de questions que nous nous posons a priori en nous basant notamment sur notre analogie de la voiture hybride :

- comment classer les divers systèmes hybrides ?
- quelles sont les différentes techniques de couplage ?
- quels problèmes permettent-ils de résoudre effectivement ? Résolvent-ils des problèmes fondamentaux, des problèmes que l'on ne savait pas résoudre auparavant ?
- le travail supplémentaire que requièrent la conception et la réalisation d'un système hybride est-il réellement rentable ?
- la maintenance d'un système hybride est-elle plus difficile ? En effet, une voiture hybride peut souffrir d'un plus grand nombre de pannes, et d'une manière générale l'augmentation de la complexité engendre l'augmentation du risque de panne ;
- les systèmes hybrides permettent-ils de résoudre certains des problèmes rencontrés par leurs composants, tels que la fragilité des systèmes symboliques ?
- quelles sont les directions de recherche futures ?

En particulier, pouvoir classer les divers systèmes est très important : comme nous le verrons dans ce chapitre, les systèmes hybrides neurosymboliques sont nombreux et très variés, de par leurs composants et par les techniques de couplage utilisées, mais aussi en raison de différents sens donnés au mot hybride dans le cadre général de l'intégration des points forts des systèmes symboliques et connexionnistes. Clarifier ce domaine en pleine évolution, comparer les systèmes, distinguer ce qui est nouveau sont des préalables indispensables à de nouvelles recherches, et une hiérarchie de classes est alors un outil fort utile.

Ce chapitre[1] est organisé de la manière suivante : la section 2.1 présente le cadre général dans lequel sont développés les SHNS, celui de l'intégration neurosymbolique ; la section 2.2 présente les SHNS proprement dits, tandis que la section 2.3 est consacrée à des systèmes souvent considérés comme hybrides, mais en fait purement connexionnistes ; enfin la section 2.4 décrit une approche intermédiaire basée sur des traductions (système symbolique traduit en système connexionniste ou inversement).

2.1 Les différentes approches pour l'intégration neurosymbolique

En fait, un grand nombre de travaux se situent dans le courant de l'intégration des meilleures caractéristiques de chacun des mondes symboliques et connexionnistes, et beaucoup ne sont pas exactement des systèmes hybrides au sens strict (c'est-à-dire des systèmes qui comportent au moins un composant de chaque type). Par exemple certains systèmes sont purement connexionnistes mais comprennent une base de règles codée sous forme de réseau de neurones. Il faut donc distinguer entre hybridité stricte et hybridité large.

Cette diversité d'approches apparaît notamment dans un compte-rendu du workshop AAAI-92 « *Integrating Neural and Symbolic Processes* » [Sun et Bookman 1993]. Ces deux auteurs distinguent quatre grandes approches pour l'intégration :

- l'approche localiste : elle regroupe tous les systèmes composés uniquement de réseaux de neurones dans lesquels chaque nœud représente un concept précis. Cela va de systèmes experts implémentés sous forme de réseaux à la construction de réseaux très spécialisés pour certains traitements symboliques. Il s'agit en fait d'un ensemble de travaux qui tentent de **paralléliser, à l'aide de réseaux localistes** (donc une classe particulière de machines parallèles), des traitements ou même des systèmes symboliques complets. Cette approche est décrite dans la section 2.3.1 ;
- l'approche distribuée : ici tous les systèmes sont composés de réseaux basés sur des représentations distribuées. L'objectif est là aussi d'arriver à implémenter sur de tels réseaux des systèmes ou traitements symboliques (comme l'unification), et d'étudier les éventuelles propriétés nouvelles qui pourraient en résulter. Voir la section 2.3.2 ;
- l'approche combinée localiste/distribuée : cette approche ressemble un peu à l'approche hybride proprement dite, mais les systèmes ne comportent toujours que des réseaux. Voir la section 2.3.3 ;
- les autres approches : tout ce qui ne relève pas des trois approches ci-dessus, et cela concerne en particulier nos systèmes hybrides, ainsi que les approches basées sur des traductions.

1. Ce chapitre doit beaucoup à de nombreuses discussions avec Mélanie Hilario (CUI Genève) et Yannick Lallement (CRIN Nancy), dans le cadre du projet Esprit MIX.

Ces auteurs accordent une place exagérée aux trois premières approches, purement connexionnistes, par rapport à leurs applications possibles. Aussi nous préférons distinguer trois classes d'importance comparable du point de vue des applications :

- approche **hybride** (section 2.2),
- approche **purement connexionniste** (section 2.3) ; cette dernière classe a pour filles l'approche distribuée, l'approche localiste et l'approche combinée,
- approche « **semi-hybride** » (section 2.4), pour les travaux relatifs aux traductions.

Nous présentons ces trois classes dans les sections suivantes, et elles seront de plus rappelées sur la figure 2.3.

2.2 L'approche hybride proprement dite

Il s'agit des systèmes qui sont effectivement composés d'au moins deux modules de types différents, modules qui doivent être complets : en particulier le module symbolique doit comporter à la fois une base de connaissances et un moteur d'inférences. Toutefois, il existe là aussi une grande variété d'approches, et il nous faut en premier lieu examiner des critères permettant de les répartir dans différentes classes. La difficulté ici est de trouver des critères suffisamment clairs et précis pour classer sans ambiguïté les systèmes. Pour cela nous allons examiner plusieurs classifications plus anciennes dans la section 2.2.1, avant de présenter dans les sections suivantes les systèmes à couplage respectivement faible, étroit, fort.

2.2.1 Les classes de systèmes hybrides

Une première classification de systèmes hybrides a été réalisée par Medsker et Bailey [1992] qui classent les divers systèmes selon leur **degré de couplage** (figure 2.1). Ils distinguent tout d'abord le cas où les deux modules sont totalement indépendants, qui n'est intéressant que pour faire des comparaisons et est assez secondaire, et le cas des « transformations » de modules, où l'on transforme globalement un système en un autre par une procédure plus ou moins automatique. Nous préférons d'ailleurs le terme de **traduction**. Ensuite ces auteurs examinent les moyens de communication entre les modules, ce qui leur permet de distinguer trois cas, allant du **couplage faible** (communications par fichiers) à l'**intégration totale** (les deux modules partagent des mécanismes et des structures de données) en passant par le **couplage étroit** (communications par la mémoire vive).

Cette première classification pose plusieurs problèmes. D'une part le critère utilisé (le moyen de communication, fichier ou mémoire vive par exemple) nous semble assez secondaire, et cette information technique est rarement donnée dans une publication. D'autre part, les auteurs utilisent ce critère pour mettre dans la catégorie « intégration totale » les travaux concernant les systèmes purement connexionnistes, ce qui va à l'encontre de la classification de Sun et Bookman [1993] donnée ci-dessus.

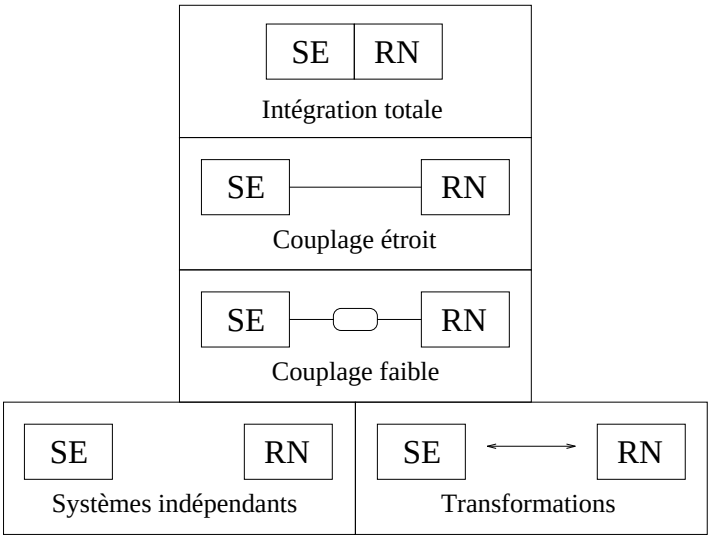


Fig. 2.1 - *Différents degrés de couplages entre modules, d'après Medsker et Bailey [1992]. Les modules sont soit des systèmes experts (SE) soit des réseaux de neurones (RN), mais n'importe quel système symbolique pourrait être mis à la place d'un SE dans cette classification. Nous préférons utiliser « traduction » à la place de « transformation ».*

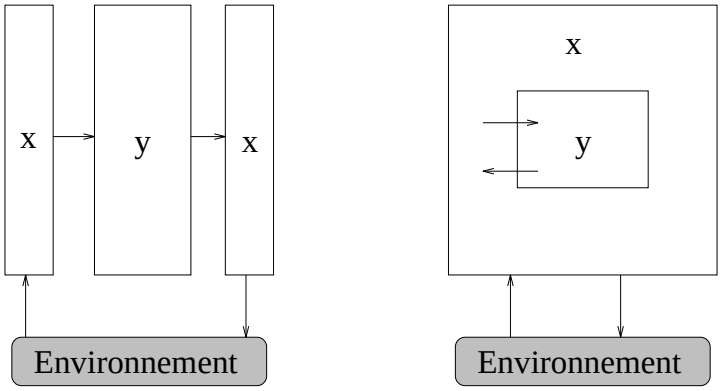
La nature hybride de ces travaux étant contestable, nous préférons donner raison à Sun et Bookman [1993] contre Medsker et Bailey [1992].

Une autre classification comprenant trois classes voisines est présentée par Giacometti [1992], qui utilise toutefois un critère qui nous semble plus satisfaisant. Il définit le couplage faible par l'existence d'une simple relation d'entrée/sortie entre deux modules, le couplage étroit, par la possibilité d'échanges bi-directionnels d'informations, le couplage fort par l'existence d'interactions « continues » entre les modules, dans le sens où **à tout moment** les modules peuvent s'influencer l'un l'autre. La classification de Giacometti [1992] ne comporte toutefois que peu de systèmes, et les exemples de couplage fort qui y sont donnés sont des systèmes ne comportant que des réseaux de neurones, donc pas hybrides.

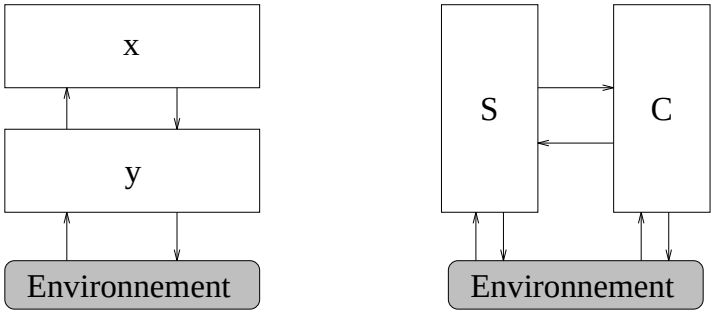
Une troisième classification a été établie par Hilario [1993], qui préfère utiliser le critère du **mode d'intégration**[2] (fig. 2.2). Cette dernière classification, plus complète, a toutefois l'inconvénient d'utiliser des définitions trop ou pas assez précises, ce qui conduit souvent à des désaccords sur la classe de tel ou tel système. D'autre part, cette classification ne fait pas apparaître le critère du **degré de couplage**. Pour nous, les deux critères sont également importants, aussi nous avons abouti à la classification de la figure 2.3 qui les fait apparaître tous les deux. Nous avons également affiné les définitions de Hilario [1993], comme nous l'indiquons ci-dessous.

D'une part, M. Hilario définit les coopérations (*co-processing*) de la manière suivante : « *Les sous-systèmes symboliques et connexionnistes sont des partenaires égaux dans le processus de résolution de problèmes. Chacun peut interagir directement avec l'environnement, le système symbolique peut transmettre de l'information par compilation de connaissances et le système connexionniste peut en transmettre par extrac-*

2. Le mot intégration étant sur-utilisé, nous parlerons plutôt dans la suite de mode d'interaction.



(a) Pré/post-traitements (b) Sous-traitance



(c) Méta-traitements (d) Coopérations

Fig. 2.2 - *D'après Hilario [1993], quatre modes d'intégration entre deux modules x et y, avec $x, y \in \{symbolique, neuronal\}$ mais $x \neq y$. En (a) les deux composants opèrent en séquence, dans une optique de pré/post-traitements. En (b), l'un des deux composants est subordonné à l'autre, et de plus n'a pas de contact direct avec l'environnement. En (c), l'un des composants est le principal résolveur de problèmes, et l'autre intervient à un méta-niveau (surveillance, contrôle, amélioration des performances). En (d), les modules sont sur un pied d'égalité, et chacun peut interagir directement avec l'environnement.*

tion de connaissances ». Cette définition est en fait taillée sur mesure pour ranger les travaux de type traduction, comme ceux de Towell [1992], dans les coopérations. Là encore, la nature hybride de tels travaux est assez particulière, si bien que nous avons préféré leur attribuer une classe propre dans notre hiérarchie, sous-classe de la classe racine (fig 2.3) : l'approche semi-hybride. La définition devient :

Définition 2.1 *Dans une interaction de type* ***coopération****, les sous-systèmes symboliques et connexionnistes sont des partenaires égaux dans le processus de résolution de problèmes. Chacun peut interagir directement avec l'environnement.*

D'autre part sa définition de la sous-traitance (*sub-processing*) et des méta-traitements (*meta-processing*) rend délicate la classification de certains systèmes. Dans le cas de la sous-traitance, « *l'un des deux sous-systèmes est encapsulé dans l'autre (et lui est subordonné), lequel est le résolveur de problèmes principal* » . Dans le cas des méta-traitements, « *l'un des deux sous-systèmes est le résolveur de problèmes de base, et l'autre a un rôle de méta-niveau (tel que surveillance, contrôle, amélioration des performances) vis-à-vis du premier* » . Le problème est que ces définitions de classes acceptent toutes les deux comme instance par exemple le démonstrateur de théorèmes SETHEO (qui sera mieux décrit plus loin), dans lequel un réseau de neurones est appelé à chaque pas de raisonnement, et dont le résultat sert à choisir de manière heuristique la branche suivante du raisonnement. Décrit de cette manière, ce système rentre bien dans la classe de la sous-traitance, mais Hilario [1993] le range dans celle des méta-traitements car le réseau lui paraît effectuer un traitement sur le traitement. D'autres systèmes cités par Hilario [1993] posent les mêmes problèmes.

Pour supprimer ces confusions, il faut examiner ces systèmes de manière très précise. Si l'on reprend l'exemple de SETHEO, il y a en effet deux cas possibles :

- le réseau rend **la** branche de raisonnement choisie, sans possibilité de remise en cause par le module symbolique : dans ce cas, le réseau a bien un rôle de méta-processeur ;
- le réseau, appelé autant de fois qu'il y a de branches de raisonnement, donne un score à chaque branche ; et c'est finalement le module symbolique qui choisit (éventuellement en utilisant d'autres heuristiques) la branche suivante : dans ce cas le méta-processeur est à l'intérieur du module symbolique, et il s'agit de sous-traitance.

Après un tel examen, il s'avère que le système SETHEO est plutôt composé d'un méta-processeur symbolique, si bien que du point de vue du mode d'interaction dans un SHNS, il s'agit de sous-traitance et non pas de méta-traitement.

Nous posons les définitions suivantes, de manière à supprimer ces ambiguïtés :

Définition 2.2 *Dans une interaction de type* ***méta-traitement****, l'un des deux sous-systèmes est le résolveur de problèmes de base, et l'autre a un rôle de méta-niveau (tel que surveillance, contrôle, amélioration des performances) par rapport au premier.*[3]

3. Cette notion de méta-niveau pourrait être éventuellement affinée. On pourrait par exemple exiger que le contrôle exercé par le méta-niveau soit parfaitement explicite. Une question intéres-

Définition 2.3 *Dans une interaction de type* ***sous-traitance****, l'un des sous-systèmes est entièrement subordonné à l'autre, qui décide quand l'appeler et comment utiliser ses sorties.*

Ainsi une seule de ces définitions, la sous-traitance, correspond alors à SETHEO.

Les interactions de type pré/post-traitements ne posent par contre pas de problème :

Définition 2.4 *Dans une interaction de type* ***pré/post-traitement****, les sous-systèmes sont reliés par une simple relation d'entrée/sortie.*

2.2.2 Notre propre classification

Notre classification, qui reprend certaines des classes mentionnées ci-dessus, est présentée sur la figure 2.3.

En dessous de la classe de l'intégration neurosymbolique, on trouve les trois grandes approches possibles :

- purement connexionniste : ses sous-classes sont les trois premières classes de Sun et Bookman [1993] :
  - approche localiste,
  - approche distribuée,
  - approche combinée ;
- hybride : c'est la quatrième classe distinguée par Sun et Bookman [1993], et nous distinguons deux sous-classes :
  - avec composants bien séparés, et dans ce cas il y a deux sous-classes, selon le degré de couplage, faible (existence d'une simple relation d'entrée/sortie entre deux modules) ou étroit (possibilité d'échanges unidirectionnels d'informations),
  - avec composants intégrés (couplage fort) ;
- semi-hybride : c'est la classe des systèmes réalisant des traductions[4], et nous indiquons deux sous-classes particulières fréquemment rencontrées, l'extraction et la compilation de structures symboliques, ces structures symboliques étant la plupart du temps des règles de production d'ordre 0.

Précisons quelques définitions :

- nous parlons d'approche hybride **au sens large** pour désigner les travaux relevant des approches purement connexionnistes et semi-hybrides ; nous qualifions les réseaux de neurones artificiels développés dans le cadre de cette approche

sante apparaît alors : un réseau de neurones peut-il représenter un contrôle explicite, et donc peut-il être un méta-processeur ?

4. Les mécanismes de traduction apparaissent parfois dans certains systèmes hybrides, comme SYNHESYS, mais ils sont également étudiés pour eux-mêmes, ce qui justifie une classe à part entière.

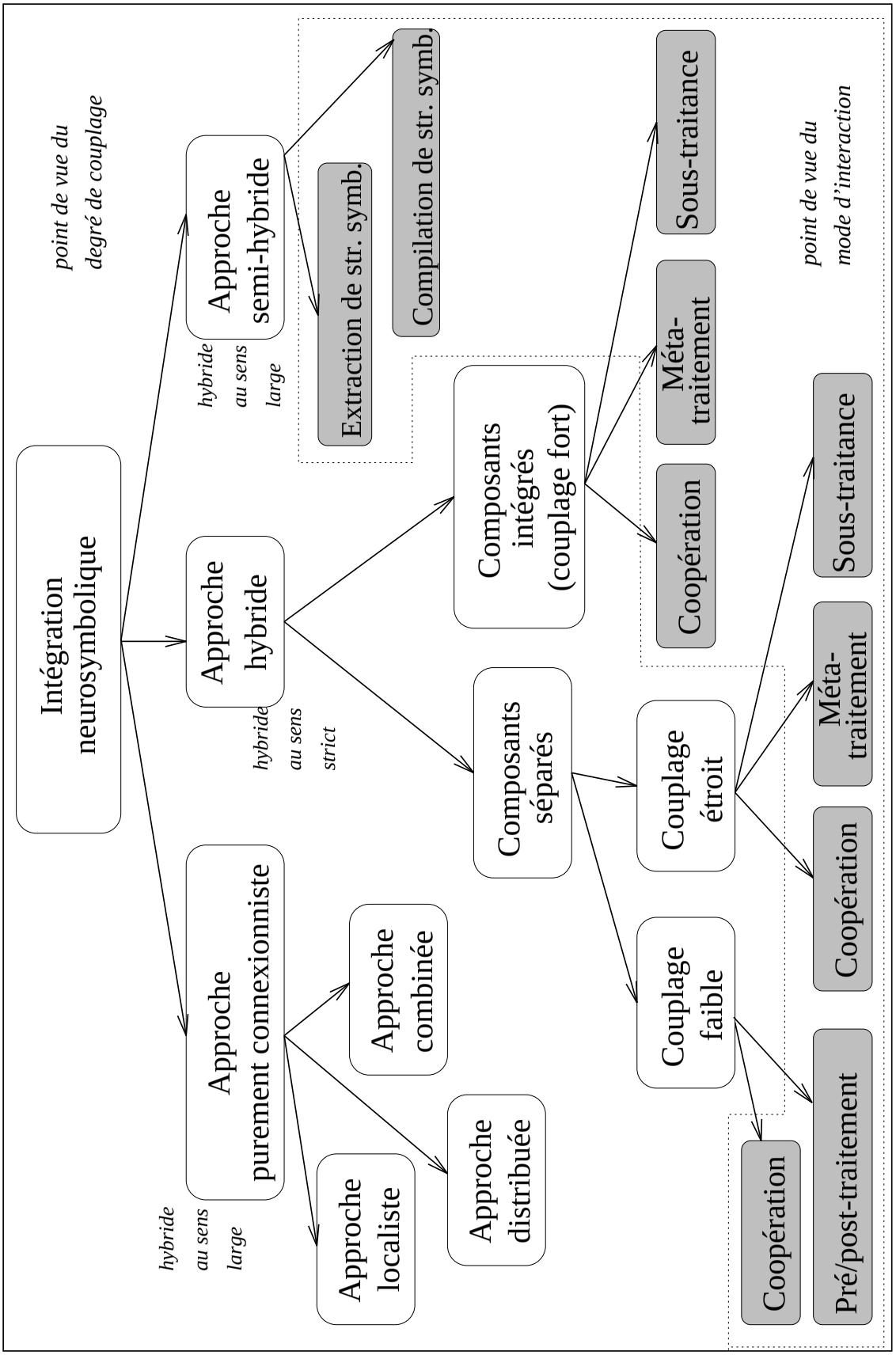


Fig. 2.3 - *Notre propre hiérarchie de classes de systèmes permettant l'« intégration neurosymbolique ». « str. symb. » signifie structures symboliques. Commentaire dans le texte (paragraphe 2.2.2).*

de **réseaux neurosymboliques** : en effet on cherche à donner à ces réseaux la capacité de représenter et manipuler des symboles ;

- nous parlons de **systèmes symboliques purs** pour désigner les systèmes symboliques qui ne sont pas implémentés sous forme de réseaux de neurones artificiels ;
- nous parlons de **réseaux neuronaux purs** pour désigner les réseaux qui ne visent pas particulièrement à représenter et manipuler des symboles.

Dans la taxonomie, au niveau le plus bas apparaissent des classes en grisé : elle correspondent aux modes d'interaction qu'offre chacune des classes distinguées en examinant le degré de couplage. Pour simplifier nous avons représenté ces différents points de vue sur un seul plan.

Nous allons maintenant illustrer ces différentes classes par quelques exemples représentatifs.

2.2.3 Systèmes à 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 uni-directionnelles.

Une application dans le domaine médical

Les capacités de classification des réseaux de neurones peuvent par exemple être utilisées pour fournir des faits symboliques traités ensuite par un système expert. C'est le cas notamment du système hybride de Ciesielski, Hayes, et Kelly [1992], utilisé dans le domaine de la surveillance temps réel de patients ayant des difficultés respiratoires, et dont le principe est résumé par la figure 2.4. Durant la réalisation d'un premier système expert symbolique, les auteurs ont constaté que la plus grande difficulté consistait à obtenir des experts une formulation précise de certaines notions qualitatives (« la pression est constante », « la pression augmente graduellement », « la pression augmente rapidement »), tandis que l'obtention de règles de plus haut niveau, recommandant des actions à prendre à partir de ces constations qualitatives, posait beaucoup moins de problèmes. D'où l'idée d'entraîner un réseau de neurones à reconnaître ces situations, et d'utiliser les sorties du réseau comme faits dans le système expert.

Un réseau multicouches, composé de 20 unités d'entrée, de deux couches cachées de 10 et 8 unités respectivement, de 6 unités de sortie, est entraîné séparément avec la méthode de la rétropropagation du gradient sur une cinquantaine d'exemples. Les unités de sortie correspondent chacune à une interprétation particulière des variations de pressions fournies par des capteurs et présentées aux entrées du réseau.

Puis en cours d'utilisation, à chaque intervalle de temps, les sorties du réseau sont passées en entrée au système expert, dans le sens où l'interprétation symbolique associée à l'unité de sortie qui est la plus activée est stockée dans la mémoire de travail

du système expert. D'après les comparaisons effectuées par les auteurs, le système hybride donne des résultats meilleurs que leur premier système expert (97.5 % de succès contre 74.5 %), et détecte les problèmes respiratoires plus rapidement. D'autre part les auteurs estiment que l'effort de développement est inférieur dans le cas du système hybride (2 mois contre 3 mois dans le cas du système expert symbolique).

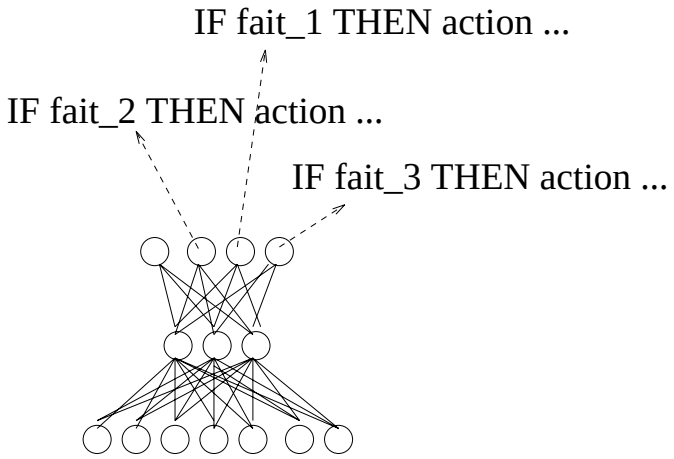


Fig. 2.4 - *Principe simplifié d'un système hybride à couplage faible du type de celui de Ciesielski, Hayes, et Kelly [1992]. Les sorties du réseau sont interprétées comme faits symboliques utilisables par un système expert.*

Une application en démonstration automatique

Le système SETHEO, dont nous avons parlé page 58, comprend en fait deux types de couplage. Le premier type de couplage (sous-traitance) a été décrit schématiquement ci-dessus et sera détaillé page 67. Le deuxième couplage est décrit ici.

Ce couplage est décrit dans [Kurfess et Reich 1989], et reste dans cette optique de prétraitement, mais ne produit pas de faits symboliques. Le système symbolique est un démonstrateur de théorèmes, qui explore l'arbre de recherche pour une formule donnée jusqu'à une certaine profondeur. Cette profondeur est en fait fournie par un réseau de neurones, qui a été entraîné sur des couples connus de la forme *(clause, profondeur optimale)*, où la profondeur optimale est la longueur de la plus courte preuve. L'objectif est de capturer une éventuelle relation entre la structure de la formule et la profondeur optimale. Sur 50 clauses inconnues, le réseau donne 56 % de bonnes réponses et 76 % de réponses dites utiles (la profondeur donnée par le réseau a un écart de 1 par rapport à la réponse attendue). Cependant, les essais ont été effectués sur des formules de logique des propositions, de taille limitée (4 clauses de 3 littéraux chacune), et il est donc difficile d'extrapoler à partir de ces résultats.

Diverses applications à des problèmes industriels

Des applications à des problèmes industriels réels existent également. Medsker [1994] décrit notamment :

- une étude réalisée par la firme Charles River Analytics Inc. dans le but de construire un système de surveillance d'un capteur utilisé dans une centrale nucléaire. Un prototype a été réalisé avec le générateur de systèmes hybrides NueX, commercialisé par la même firme. Les concepteurs du système ont réalisé une évaluation en travaillant sur des données réelles, et la performance est suffisamment bonne pour démontrer la faisabilité d'une telle approche. L'architecture du système est présentée par la figure 2.5 (noter que les interactions sont à la fois de type pré-traitement et coopération) ;
- un système d'interprétation d'images qui fusionne des informations provenant de différentes sources : un réseau multicouches interprète des informations extraites d'images tandis qu'un système expert vérifie les décisions du réseau en utilisant une base de données géographiques ;
- une autre application réalisée avec NueX, dans le domaine militaire : il s'agissait de réaliser un système de reconnaissance de cibles multiples, en utilisant des données provenant de nombreux capteurs (radars, etc.).

Wilson et Hendler [1993] ont réalisé un système qui contrôle le pH d'un réservoir de produits chimiques. Un contrôleur d'un modèle très courant est habituellement utilisé pour ce type de tâche, mais il ne permet pas toujours de retourner à un point d'équilibre en un temps optimal. Aussi ces auteurs ont imaginé de fournir une entrée supplémentaire au contrôleur : une prédiction de la valeur future du pH. Cette prédiction est effectuée par un réseau de neurones, multicouches à rétropropagation. Toutefois cette méthode pouvant conduire à des oscillations importantes, les auteurs ont ajouté un petit système expert qui décide à chaque pas de temps s'il faut utiliser ou pas la valeur prédite, en fonction de la valeur courante du pH et de la valeur prédite. Il s'agit donc bien d'un système hybride. Les résultats du système hybride sont meilleurs que ceux du contrôleur seul, car le retour au point d'équilibre est plus rapide.

Conclusion

Ces systèmes faiblement couplés ont donc des architectures très simples et sont faciles à réaliser, car le travail de couplage consiste essentiellement à définir une interface de communication entre des systèmes bien séparés, éventuellement préexistants. Ces systèmes ne présentent pas de problèmes de recherche fondamentaux, et ont surtout l'intérêt d'apporter des solutions peu coûteuses, et de pouvoir être utilisés dans des applications réelles ou réalistes. Ils sont surtout adaptés à la mise en œuvre des interactions de type pré/post-traitement (fig. 2.2), ce qui permettra par exemple d'utiliser des réseaux de neurones pré-entraînés pour faire de la reconnaissance des formes, de la classification ou du traitement du signal.

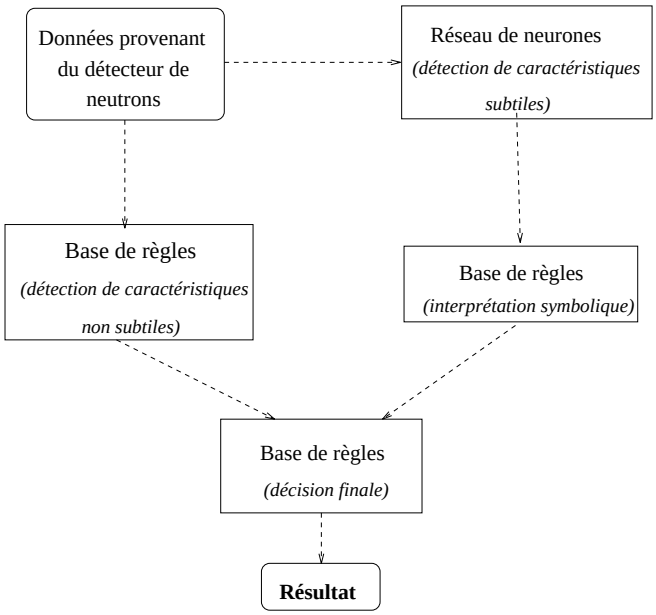


Fig. 2.5 - *Architecture d'un système hybride de surveillance d'un capteur. Nous présentons une simplification d'une figure de Medsker [1994]. Dans le système hybride, un réseau de neurones sert à détecter et isoler les erreurs d'un détecteur de neutrons, erreurs qui consistent en de subtiles (donc difficiles à modéliser explicitement) variations spatiales et temporelles ; il s'agit d'un réseau multicouches à rétropropagation auquel sont présentées simultanément les données mesurées à l'instant t et celles mesurées en $t-1$. Plusieurs bases de règles servent à déterminer des erreurs moins subtiles, à interpréter les résultats du réseau, et à donner le résultat final.*

Enfin, les réseaux sont utilisés dans cette catégorie de systèmes hybrides comme des boîtes noires, et leur caractère neuronal n'a pas réellement d'influence sur le fonctionnement global du système : les réseaux sont utilisés comme des procédures auxquelles on passe des paramètres.

2.2.4 Systèmes à couplage étroit

Dans cette catégorie, les interactions entre les modules sont plus souples car elles sont bi-directionnelles ; 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 (par exemple un module symbolique peut utiliser un réseau de neurones comme ci-dessus mais en plus piloter son apprentissage). Nous avons regroupé les divers systèmes selon le type d'interaction.

Sous-traitance

Utilisations de réseaux comme approximateurs (classifieurs) Tirri [1991] étudie de manière assez générale les couplages d'un système expert et de réseaux. Comme nous l'avons dit dans le chapitre précédent, H. Tirri critique les systèmes experts actuels sur plusieurs points :

- certaines parties du domaine des connaissances ne peuvent pas être représentées dans le formalisme qu'ils imposent (règles, frames, objets, etc.), en particulier tout ce qui n'est pas facilement « verbalisable », comme le savoir-faire [Giacometti 1992] ;
- ils demandent un travail important pour coder correctement les informations d'entrée (provenant de capteurs notamment) ; en particulier, les systèmes experts actuels n'apportent pas de réponse satisfaisante à la question suivante : comment comprimer des données sensorielles complexes multi-dimensionnelles sous une forme symbolique utilisable par un moteur d'inférence ?
- ils ne sont pas bien adaptés aux cas où ces mêmes données sont de plus bruitées ou incomplètes.

Par contre, il montre qu'une approche hybride peut remédier à ces problèmes. Remarquant qu'il est très difficile de formaliser un fait comme « la chaleur est moyenne », il propose de réaliser un détecteur spécialisé pour ce fait, à l'aide d'un réseau de neurones (soit un réseau multi-couches, à rétro-propagation, soit un réseau de Kohonen)[5]. On présentera ensuite à ce réseau des exemples de « chaleur moyenne » pour les lui faire apprendre. L'idée de base est donc très voisine de celle de Ciesielski, Hayes, et Kelly [1992], et revient à utiliser les capacités de classification des réseaux de neurones, qui peuvent notamment prendre en compte des données bruitées.

5. H. Tirri signale que la logique floue est une alternative, mais l'utilisation de réseaux lui semble une solution plus simple et plus pragmatique ; d'autre part il a observé que les règles elles-mêmes sont rarement floues.

Concrètement, lorsque le moteur d'inférence du composant symbolique rencontre dans une prémisse de règle un prédicat associé à un détecteur neuronal, il appelle ce dernier, les paramètres de l'appel étant l'adresse physique du capteur concerné et le numéro de l'une des unités de sortie du réseau (car le réseau implémente plusieurs prédicats en même temps), cette unité devant avoir une valeur suffisamment élevée pour que le prédicat soit considéré comme vérifié. Le réseau prend directement ses entrées sur les capteurs physiques, et son calcul est déclenché par l'appel du module symbolique. Il s'agit donc bien de sous-traitance.

L'un des intérêts de ce système est qu'il peut continuer à s'améliorer durant son utilisation. Ceci est indispensable, car il a souvent été constaté qu'il est difficile de réunir à l'avance un ensemble d'exemples suffisant. Aussi, en cours d'exploitation, si un fait déduit s'avère incorrect (là se pose le problème de la détection de l'incorrection, auquel H. Tirri n'apporte pas de solution), le système expert oblige le réseau à s'adapter (donner la bonne réponse pour les entrées correspondantes). Ce point mériterait une discussion plus approfondie de la part de H. Tirri, car les réseaux multi-couches qu'il envisage ne permettent pas d'apprendre de manière incrémentale, ce qui oblige à réapprendre à partir de zéro ; ceci peut être un inconvénient majeur dans certaines applications. Quoi qu'il en soit, cette possibilité qu'à le système expert de forcer l'apprentissage du réseau classe clairement le système dans le couplage étroit.

D'autres apports de ce type de système sont, d'une part, la réduction importante du nombre de règles, ce qui a d'évidents avantages (performances, lisibilité) et, d'autre part, la facilité d'implémentation, même dans des systèmes symboliques pré-existants. Par contre, le problème de la détection des cas où le réseau devrait s'adapter reste ouvert, ce qui est un inconvénient important selon nous : il devra être résolu pour que ce type de système puisse réellement apprendre de manière permanente, sans quoi les idées de H. Tirri restent fort voisines de celles des concepteurs de systèmes à couplage faible.

Kasabov et Petkov [1992] présentent des idées assez similaires, dans le cas d'un couplage programme Prolog/réseau de neurones.

Le système LAM est effectivement utilisé pour traiter un problème réel [Medsker 1994]. LAM est un SHNS qui aide ses utilisateurs à concevoir divers verres de vitres de gratte-ciels. Le système expert consulte durant ses inférences deux réseaux de neurones qui approximent des fonctions donnant des paramètres importants. Ces réseaux ont été entraînés à partir de données collectées dans la littérature relative au domaine.

Propagation de l'incertitude dans un système expert D'autres types de données peuvent toutefois circuler entre les modules : dans [Giambiasi, Lbath, et Touzet 1989], un réseau apprend (à partir d'exemples d'inférences floues donnés par un expert) le coefficient d'incertitude à donner à la conclusion d'une règle (exprimée en logique des propositions). Ceci permet d'avoir dans un système expert une propagation de l'incertitude adaptée à l'application, au lieu d'utiliser une formule relativement arbitraire comme cela est fait notamment dans MYCIN. En phase

d'exploitation, le module symbolique demande au réseau de produire les coefficients d'incertitude des faits conclusions, en lui donnant les coefficients d'incertitude des prémisses et de la règle sélectionnée. Il s'agit donc bien d'échanges bi-directionnels, et de sous-traitance. Là aussi il s'agit d'un réseau multicouches utilisant la rétropropagation du gradient, mais ne comportant que trois couches. Dans l'une des expériences réalisées, la couche d'entrée contenait 33 unités, la couche cachée 25 et la couche de sortie 11. Ce nombre important d'unités d'entrée et de sortie s'explique par la discrétisation des coefficients réels. Ainsi le codage d'un coefficient appartenant à $[0, 1]$ demande 11 unités, dont une signifie l'absence de valeur.

Pour tester le système, les auteurs tentent simplement de retrouver à partir d'exemples la formule de propagation d'incertitude utilisée dans MYCIN (1331 exemples, règles ayant au plus deux prémisses) et une formule voisine dite de Lee.

Ils y parviennent parfaitement et concluent que les résultats sont prometteurs, mais ceci montre simplement que ces fonctions peuvent être approximées par un réseau, et nous apporte peu de connaissances sur l'intérêt de cette approche. En effet, il reste à savoir si dans le cas d'un problème pratique on peut obtenir suffisamment d'exemples, si le réseau généralise suffisamment bien, etc. Le vrai travail d'évaluation reste donc à faire. D'autre part il reste des problèmes pratiques pour généraliser ce travail à des règles comportant plus de deux prémisses, car le nombre de neurones et le nombre d'exemples augmentent en fonction du nombre de prémisses et du pas de discrétisation choisi. De plus, un réseau comme celui qui est décrit ci-dessus ne peut traiter que des règles ayant un nombre de prémisses déterminé a priori, et il faut donc trouver de nouvelles solutions pour traiter une base de règles dont le nombre de prémisses n'est pas connu à l'avance.

Retour à SETHEO Une autre forme de sous-traitance est implémentée dans le démonstrateur de théorèmes SETHEO, cité ci-dessus. Pour éviter l'exploration systématique de l'arbre de recherche, un réseau permet de choisir la branche de l'arbre la plus intéressante. Un réseau à rétro-propagation, préalablement entraîné, reçoit des caractéristiques « statiques », c'est-à-dire pouvant être extraites avant le début de la preuve (par exemple, nombre de littéraux dans une clause, symboles de prédicats, variables, constantes, variables partagées, etc.) et des caractéristiques « dynamiques », obtenues durant l'exécution (profondeur de preuve courante, variables actuellement instanciées, nombre d'occurrences de tel prédicat, etc.). Il permet d'attribuer un score à chacune des branches possibles. L'intérêt du système est d'implémenter assez facilement un mécanisme heuristique, et aussi de constituer un outil utile pour étudier l'importance des caractéristiques ci-dessus pour choisir des stratégies locales ou globales. Là aussi le réseau sert d'approximateur de fonction. Une expérimentation a été réalisée avec la démonstration de théorèmes simples de la théorie des groupes et des anneaux [Goller 1994]. L'heuristique réalisée sous forme de réseau de neurones à 3 couches permet de démontrer environ dix fois plus rapidement les deux théorèmes les plus difficiles. Le gain de temps est donc significatif. Pour réaliser des heuristiques plus puissantes, il serait nécessaire de pouvoir évaluer ou classer des structures récursives de taille arbitraire (tentatives de preuves, sous-buts, termes). L'approche ci-dessus est alors limitée car elle oblige à décrire ces

structures par un ensemble fixé de caractéristiques. Aussi Goller [1994] s'intéresse maintenant aux réseaux qui permettent de représenter de telles structures, comme RAAM [Pollack 1990].

Conclusion Dans cette classe, le caractère neuronal prend plus d'importance et pourrait notamment se manifester par l'exploitation en cours de fonctionnement de la capacité d'apprentissage des réseaux. Mais cela ne semble pas avoir été exploité dans des applications, en raison notamment de la difficulté de réaliser un apprentissage incrémental avec des réseaux à rétropropagation, qui restent les plus employés. Le couplage le plus employé est l'appel d'un réseau de neurones par un système expert.

Méta-traitements

Une application au pilotage d'un avion Handelman, Lane, et Gelfand [1992] décrivent un système hybride à couplage étroit assez complexe, appelé RSA2 (*Robotic Skill Acquisition Architecture*) et qui présente plusieurs types d'interactions, à savoir méta-traitement et coopération (fig. 2.6). L'objectif du système est d'intégrer des systèmes à base de connaissances et des réseaux de neurones pour faire du contrôle en robotique, cette intégration étant fondée sur des modèles **comportementaux** (*behavioral*) de l'acquisition biologique de capacités sensorimotrices. En particulier les auteurs s'intéressent aux mécanismes qui permettent à un individu d'apprendre peu à peu des réponses sensorimotrices complexes et pourtant efficaces lorsqu'on lui donne, pour accomplir une certaine tâche, des explications verbales, des exemples des mouvements typiques qu'elle nécessite, et du temps pour s'entraîner.

Le fonctionnement détaillé du système serait trop long à décrire ici. Nous nous limitons donc à une description de haut niveau. Les auteurs utilisent des analogies avec des modèles biologiques pour définir quatre grandes phases de fonctionnement de leur système. Durant la **phase déclarative**, les composants à base de connaissances effectuent la tâche de pilotage, d'une manière approximative[6]. Durant la **phase hybride**, les composants neuronaux commencent tout d'abord à apprendre, grâce aux exemples fournis par les systèmes à base de connaissances, à accomplir des parties de la tâche, mais sans contribuer à sa réalisation. Puis, dans la même phase, ils se mettent à partager la responsabilité du contrôle effectif, leur performance s'améliorant au fil du temps ; ce partage est facilement réalisé car la commande est la somme algébrique des commandes des divers composants. Finalement, durant la **phase réflexive**, le contrôle effectué par les réseaux est optimisé par apprentissage renforcé.

La composante de méta-traitement a plusieurs fonctions, dont par exemple celle de surveiller les erreurs d'apprentissage des réseaux et de modifier des paramètres de la méthode d'apprentissage. Cela peut conduire des méta-règles à décider de remplacer

6. Les auteurs disent « *During the declarative phase of skill acquisition, knowledge-based systems components* ***discover*** *how to obtain rough-cut task performance* ». Mais il ne décrivent aucun mécanisme permettant une telle découverte, si bien que nous pensons que c'est surtout le concepteur de la base de connaissances qui s'en charge en expérimentant diverses règles.

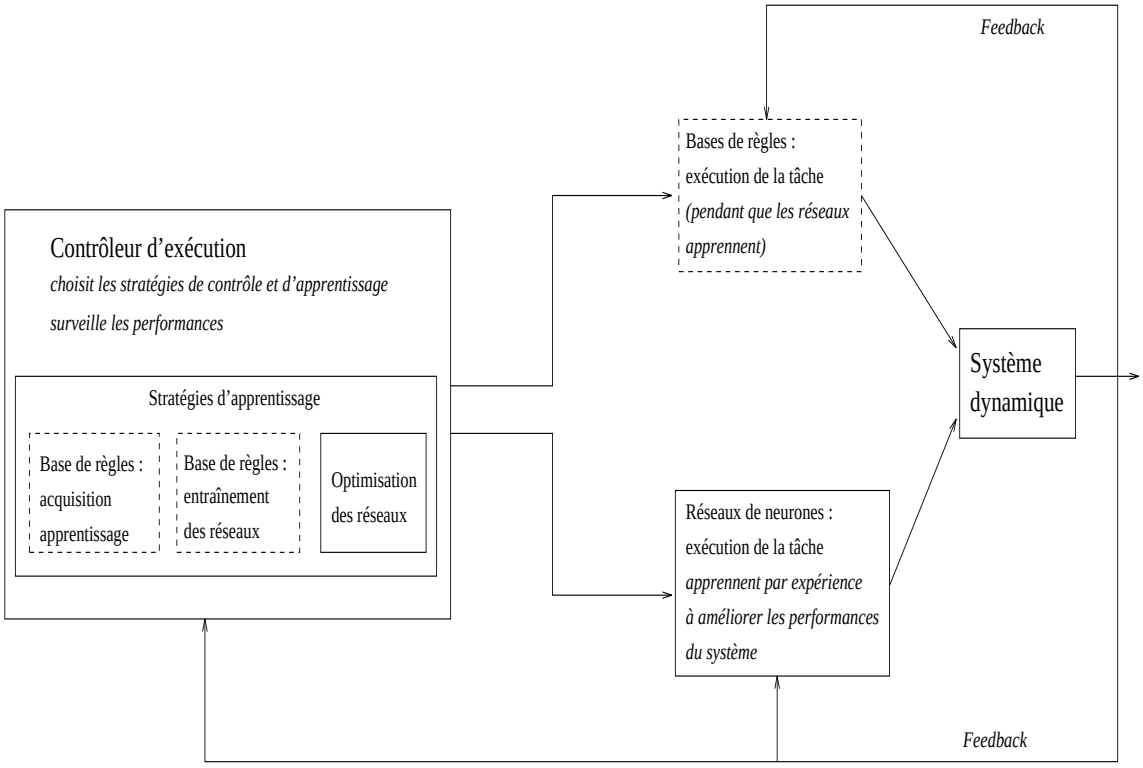


Fig. 2.6 - *Architecture globale du système hybride RSA2 de Handelman, Lane, et Gelfand [1992], intégrant des systèmes à base de connaissances et des réseaux de neurones pour l'acquisition de capacités robotiques.*

des groupes de règles de la partie exécution par des réseaux lorsque ceux-ci sont jugés suffisamment entraînés.

Ce système est appliqué dans une simulation de contrôle d'un avion durant les phases d'approches et d'atterrissage. Diverses expériences montrent que les performances sont meilleures en fin de phase hybride que dans la phase déclarative (l'avion atterrit « mieux » en termes de vitesse, distance, etc.), et satisfaisantes en phase réflexive, dans laquelle les réseaux sont utilisés seuls.

Pour les auteurs, un tel système est conçu de manière à pouvoir intégrer des techniques de contrôle conventionnelles, quand elles peuvent s'appliquer, tout en utilisant des techniques d'apprentissage neuronal pour réduire certains des coûts de la modélisation de systèmes complexes. D'autre part, ils supposent qu'une telle architecture, d'inspiration biologique, fournira une bonne interface homme-machine, permettant à la fois non seulement de dire quoi faire à une machine, mais aussi de le lui montrer.

Coopération

Résolution de problèmes de cinématique Gutknecht et Pfeifer [1990] ont réalisé l'un des tout premiers systèmes hybrides. Tout comme dans le cas de SYNHESYS, il s'agit d'une approche plutôt de type psycho-mimétique. En effet le système tente de passer progressivement d'un raisonnement dirigé par les buts, caractéristique d'un débutant, à un raisonnement dirigé par les données, caractéristique de l'expert. Le problème à résoudre est un problème simple de cinématique, caractérisé par des données (un ensemble de variables dont les valeurs sont données) et une variable but dont la valeur doit être déterminée. Pour cela, le système dispose de connaissances de cinématique, à savoir les diverses équations liant les variables ; ces équations sont exploitées en chaînage avant ou arrière par un système expert. On dispose également d'un réseau multi-couches à rétro-propagation du gradient, qui en cours d'utilisation apprend à prédire, en fonction de l'ensemble des variables dont on connaît la valeur et de la variable but, quelle variable peut être déterminée directement, c'est-à-dire en appliquant une seule équation.

Voici un exemple de règle représentant la relation $v(t) = v_0 + a \cdot t$:

```
IF (or (the final_velocity is searched_for)
       (the final_velocity is unknown))
   (the initial_velocity is known)
   (the acceleration is known)
   (the time is known)
THEN
   (the final_velocity is known)
   (lisp (print ``Applying P1 : v = v0 + a.t''))
```

Le fonctionnement du système est le suivant (voir aussi la figure 2.7) : en fonction des données initiales (par exemple, v_0 et a sont donnés), le réseau « propose » éventuellement une variable. Pour cela, il faut que l'une de ses unités de sortie possède une valeur suffisamment grande par rapport à un seuil fixé. Il y a deux cas :

1. Si cette condition n'est pas remplie, le système expert calcule alors la solution

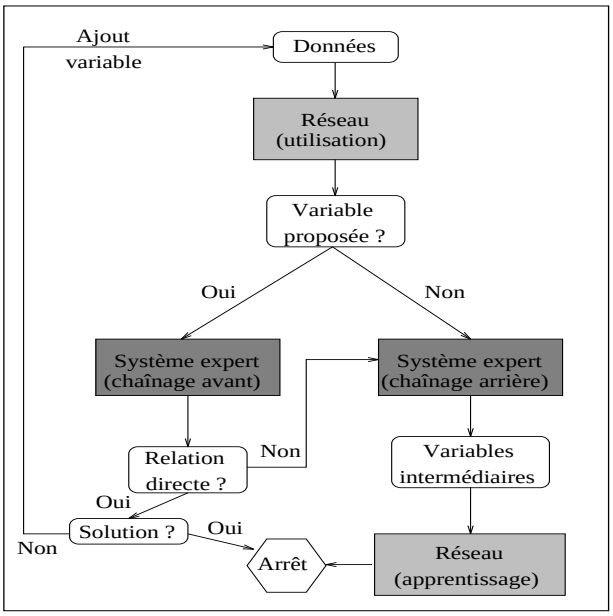


Fig. 2.7 - *Schéma d'interaction inter-modules dans le cas du système hybride de Gutknecht et Pfeifer [1990]. Commentaires dans le texte.*

du problème par chaînage arrière, ce qui permet d'obtenir une suite de variables intermédiaires qui ont dû être déterminées pour obtenir la variable but. Le réseau apprend alors à prédire le passage d'une variable intermédiaire à une autre. Dans notre exemple, le réseau apprendrait à proposer v quand v_0 et a sont donnés.

2. Si par contre le réseau a proposé une variable, le système expert cherche (en chaînage avant) si une relation directe est effectivement disponible. Si oui, il infère que cette variable est connue, et on dispose donc d'une donnée supplémentaire ; le processus recommence tant qu'on n'a pas la variable but est inconnue. Si non, on effectue la phase de chaînage arrière décrite ci-dessus.

Le système passe donc d'un fonctionnement essentiellement en chaînage arrière au début à un fonctionnement utilisant de plus en plus un chaînage avant piloté par le réseau. Les auteurs ne tentent malheureusement pas d'évaluer un éventuel gain de performance[7]. Pour eux, l'intérêt du couplage est d'éviter au système expert de poser des questions inutiles à l'utilisateur et de focaliser rapidement le système sur l'hypothèse la plus probable. D'autre part, c'est un moyen d'intégrer l'« expérience » du système au fur et à mesure de son utilisation.

Enfin, l'un des grands intérêts de ce travail réside dans le fait que les auteurs mettent en avant quatre problèmes fondamentaux pour les systèmes hybrides :

1. l'apprentissage du réseau doit être **incrémental**. Traditionnellement, la plupart des réseaux de neurones sont en effet entraînés sur un jeu d'exemples fixé, et ensuite utilisés sans aucune modification. Cela est difficilement accep-

7. C'est l'une des difficultés posées par les systèmes d'inspiration psycho-mimétique : faut-il les évaluer d'un point de vue informatique (étude de complexité, mesure de performance, etc.) ou d'un point de vue psychologique ?

table dans les nombreux domaines où de nouveaux exemples devraient pouvoir être pris en compte en cours d'utilisation (leur système comporte d'ailleurs un apprentissage incrémental que nous n'avons pas décrit).

2. les réseaux multicouches produisent des décisions **en une étape**, ce qui les rend difficiles d'emploi quand la résolution du problème exige des décisions ou calculs intermédiaires ou des interactions avec l'environnement, ou encore des retours arrières dans le raisonnement. Ils ne peuvent donc effectuer que des parties de raisonnement.
3. il faut définir une **interface** entre les mondes symboliques et « sous-symboliques » (les réseaux), car leur façon de représenter des connaissances est fondamentalement différente. Toutefois la plupart du temps, les systèmes hybrides utilisent des solutions très simples comme associer un concept symbolique à un nœud dans le réseau. D'autre part, ces liens sont le plus souvent « câblés » (*hard-wired*).
4. il faut réfléchir à la **répartition du travail** entre les sous-systèmes. Pour cela il faudrait se baser sur une discussion des principes sous-jacents aux divers systèmes hybrides.

Autres systèmes Enfin le système SYNHESYS (chapitre 3) a également sa place dans les systèmes à couplage étroit et à interactions de type coopération. C'est également le cas du système de Handelman, Lane, et Gelfand [1992] présenté ci-dessus.

2.2.5 Systèmes à couplage fort

Dans ces systèmes, les composants partagent des structures de données communes (par exemple, un nœud de réseau est en même temps un symbole dans un module symbolique), ou au moins l'un des modules a un accès direct à certaines structures de données de l'autre. Tout cela autorise des communications beaucoup plus souples et plus fines. C'est pourquoi ce sont les systèmes les plus intéressants potentiellement. Très peu ont été réalisés jusqu'à présent, car ils sont encore plus complexes que les précédents, comme on va le voir dans cette section. On peut remarquer que la complexité d'un système hybride grandit proportionnellement au degré de couplage.

Un exemple de couplage fort est donné par une liaison neuronale directe dans le système SYNHESYS (chapitre 3). Dans son cas, le type d'interaction est la **coopération**.

En pratique, ces systèmes à couplage fort doivent souvent posséder des mécanismes de traitement assez voisins pour pouvoir exploiter ces structures de données communes. Nous avons distingué deux approches[8] basées sur des mécanismes de propagation, qui mettent en œuvre des interactions du type **sous-traitance**.

8. Les arbres de décision hybrides [d'Alché Buc 1993] constituent probablement une troisième approche de couplage fort que nous n'avons pas explorée.

Propagations symboliques et numériques

Un exemple de cette approche mixte est la combinaison du réseau sémantique SCRAPS avec un réseau de neurones à rétro-propagation [Hendler 1989]. Le réseau est entraîné séparément, et apprend à classifier des instances décrites par des attributs. En cours d'utilisation, le principe de circulation de l'information, basé sur des algorithmes de passage de marques (*marker-passing*), est le suivant :

1. une activation symbolique est propagée dans le réseau sémantique ;
2. lorsqu'un lien vers le réseau de neurones est rencontré, une activation numérique est calculée à partir de l'activation symbolique. Elle est propagée le long du lien reliant les deux modules vers l'une des unités de sortie du réseau de neurones;
3. il y a ensuite dans le réseau de neurones une propagation d'information des unités des sortie vers les unités cachées, puis en sens inverse ; les unités de sortie reçoivent une valeur calculée à partir des poids des connexions et des activations des unités cachées (voir l'exemple ci-dessous) ;
4. enfin, les activités des nœuds de sortie sont propagées vers le module symbolique. Cela active certains nœuds symboliques qui deviennent capables de propager de l'information symbolique. Ces nœuds symboliques activés sont proches, au sens d'une mesure de similarité implicitement calculée par le système, du ou des nœuds de l'étape 2 ci-dessus.

Tout cela permet de déterminer des chemins dans le réseau sémantique, chemins qui sont ensuite traités de manière classique par un évaluateur de chemins. Un exemple d'utilisation du système est donné par la figure 2.8. Pour comprendre comment se passe la propagation d'activation, imaginons qu'elle commence au nœud n_1 avec pour valeur 1. Cette activation est envoyée sur l'unité cachée h_i par la connexion correspondante, de poids $w(n_1, h_i)$, si bien que l'activation de h_i devient $Activation(h_i) = w(n_1, h_i)/3$. L'activation est ensuite propagée selon le même principe vers les unités de sortie. Ainsi le neurone n_2 reçoit l'activation

$$\sum_{i=1}^{3} Activation(h_i) \cdot \frac{w(n_2, h_i)}{4} = \sum_{i=1}^{3} \frac{w(n_1, h_i)}{3} \cdot \frac{w(n_2, h_i)}{4}$$

Et de même pour les autres nœuds n_3 et n_4. Celui qui aura la plus grande activation sera, pour Hendler, celui qui partage le plus de caractéristiques avec n_1.

L'idée de l'auteur est d'exploiter la capacité des réseaux de neurones à développer des représentations internes distribuées de leurs entrées : une entrée particulière produit un vecteur particulier d'activités des unités cachées, qui est considéré comme une représentation interne de l'entrée. Dans le cas général, il n'est pas possible d'associer un symbole précis à chaque unité cachée, si bien que l'on considère qu'une unité cachée code une *micro-caractéristique*, non symbolique. C'est ce qui intéresse Hendler, et de nombreux autres chercheurs d'ailleurs.

Il est difficile de dire si ces idées sont utilisables dans une application réaliste, car le système semble n'avoir été expérimenté que sur des cas très petits et très simples

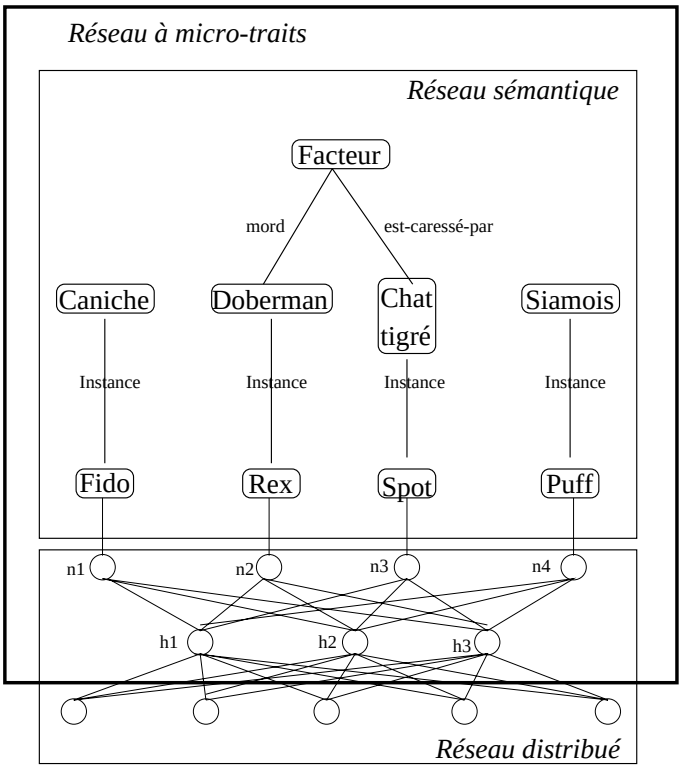


Fig. 2.8 - *D'après Hendler [1989]. Architecture d'un système hybride à couplage fort, et exemple d'utilisation. Pour simplifier, nous avons réduit le nombre d'unités du réseau distribué, sans perte de généralité. Dans cet exemple d'utilisation, l'objectif est de compléter la phrase Fido —— le facteur, bien qu'aucune relation ne soit codée entre les nœuds correspondants dans le réseau sémantique. Le réseau a été entraîné sur des jeux d'essais où soit Fido et Rex, soit Fido et Spot partagent des caractéristiques communes. Concrètement, les exemples présentés sont de la forme (caractéristiques de l'animal codées sur 5 bits ; nom de l'animal codé sur 4 bits). Les propagations permettent d'établir que Fido mord ou bien est-caressé-par le facteur, selon que Fido est considéré proche de Rex ou de Spot, qui eux ont une relation explicite avec le facteur.*

comme celui de la figure 2.8. D'autre part, à notre avis, il serait nécessaire d'étudier plus en détail les propriétés de cette mesure de similarité implicitement calculée, car les propriétés et les limites du système ne sont pas connues.

Propagation d'activité numérique

Le système MOSAIC [Pham et Degoulet 1989] est appelé *macro-connexionniste* par ses auteurs, car il s'inspire de la neurobiologie en s'intéressant à des groupes de neurones et à leurs fonctionnalités. Cependant, contrairement au macro-connexionnisme étudié notamment par Alexandre [1990], il s'agit ici d'une inspiration assez lointaine. Ce système connecte des « Neuronics » par des connections pondérées, et c'est donc un système connexionniste, tandis que dans le système de Hendler le contrôle est plutôt exercé par le module symbolique.

Le Neuronic, devenu ultérieurement Neuro-Agent dans la version commerciale de MOSAIC, IntelliSphere, [Pham 1992], est un neurone assez complexe, représentant généralement un concept particulier, comme dans un réseau sémantique, et est composé de trois zones :

- la zone nominale contient un certain nombre de caractéristiques (attributs) symboliques de l'entité ;
- l'enveloppe de communication et d'activation est elle même constitué de plusieurs autres zones, les plus importantes étant la zone d'excitation nécessaire et la zone d'excitation facultative, sur lesquelles arrivent des connexions provenant d'autres neurones.
- le processus interne est en fait une procédure, permettant de communiquer avec le monde extérieur par exemple, ce qui donne une certaine capacité d'ouverture au système ; le processus interne peut également permettre d'encapsuler dans un seul Neuronic toute une assemblée de Neuronics, et c'est pourquoi le système est dit macro-connexionniste.

Il n'est pas indispensable ici de détailler le fonctionnement exact de ces zones, assez complexe. Soulignons d'autre part que les poids des connexions entre Neuronics peuvent être déterminés par apprentissage à partir d'exemples. D'après [Pham 1992], *Après apprentissage, IntelliSphere est capable :*

- *d'établir les différentes connexions entre les Neuro-Agents,*
- *d'évaluer leur poids respectifs,*
- *de ne tenir compte que des évocations pertinentes,*
- *de dégager les informations spécifiques,*
- *de préciser les informations apparaissant simultanément,*
- *de déterminer les informations absolument nécessaires,*
- *de distinguer le bruit de l'information.*

Mais on n'a pas plus d'informations sur cet algorithme qui semble avoir d'intéres-

santes propriétés.

L'évaluation de ce système est difficile, d'une part à cause de la complexité des neurones, et d'autre part par manque d'informations suffisamment précises, notamment sur les algorithmes d'apprentissage. MOSAIC est devenu un système commercial, utilisé principalement dans le domaine médical, et vendu à un prix élevé, aussi nous n'avons eu accès qu'à une brochure commerciale. Il serait en particulier nécessaire de pouvoir le comparer aux réseaux sémantiques dont il semble être un très proche cousin.

Néanmoins, nous considérons qu'il s'agit d'un système à couplage fort car chaque Neuronic, qui est un composant « symbolique » du système hybride, effectue diverses opérations en fonction des activations numériques qu'il reçoit, et peut produire lui aussi une activation. Pour nous, il y a bien partage de structures de données au niveau des couches externes du Neuronic, si bien qu'il s'agit de couplage fort. En ce qui concerne le type d'interaction, nous pensons que ce système peut mettre en œuvre coopération et sous-traitance.

2.3 Les approches purement connexionnistes

Nous avons présenté ci-dessus les systèmes hybrides qui sont réellement composés de plusieurs modules distincts. Cependant d'autres systèmes prennent actuellement de plus en plus d'importance dans la littérature : il s'agit de systèmes purement connexionnistes, dont les motivations sont variées, mais qui cherchent assez souvent à répondre à une sévère critique des modèles connexionnistes faite en 1988 par Fodor et Pylyshin [1988] ; ces deux auteurs ont en effet essayé de montrer que les systèmes connexionnistes étaient incapables de s'attaquer aux tâches de haut niveau que requière l'étude de la cognition humaine. Pour ces auteurs, seuls des systèmes symboliques peuvent exhiber les bonnes propriétés comme la systématicité et la compositionnalité, ce qui a bien sûr piqué au vif les connexionnistes convaincus.

Nous reprenons ci-dessous les trois classes définies par Sun et Bookman [1993], avec un petit nombre de systèmes caractéristiques.

2.3.1 L'approche localiste

Émulation de systèmes symboliques

Cette tendance, sans doute la plus ancienne, a consisté à tenter d'émuler avec des systèmes connexionnistes des systèmes symboliques, dans le but de prouver la capacité des systèmes connexionnistes à effectuer des traitements de haut niveau et d'étudier l'éventuelle apparition de propriétés nouvelles. Nous parlons d'émulation car il s'agit de reproduire le fonctionnement et l'architecture d'un système symbolique.

Par exemple, le système DCPS [Touretzky et Hinton 1986] est un système expert à base de règles de production, avec une base de faits, une base de règles, et un moteur d'inférence, entièrement réalisé avec des neurones. Dans une première version sans variables, les faits sont des triplets de symboles comme par exemple $(F\ A\ B)$, ce qui revient à travailler avec des propositions valuées puisque l'on peut représenter un triplet *(prédicat, proposition, valeur)*. 25 symboles sont disponibles, et par conséquent il y a $25^3 = 15\ 625$ triplets possibles. Les règles sont du type :

$$(F\ A\ B)\ (F\ C\ D)\ \longrightarrow\ +(G\ A\ B)\ +(P\ Q\ R)\ -(F\ C\ D)$$

Cette règle signifie « ajouter les faits $(G\ A\ B)$ et $(P\ Q\ R)$ et retirer $(F\ C\ D)$ si les faits $(F\ A\ B)$ et $(F\ C\ D)$ sont présents dans la mémoire de travail ». La difficulté est de faire ces opérations de manière purement neuronale. Pour cela, les faits sont codés de manière distribuée par les valeurs d'activités d'un ensemble de neurones. Quant aux règles, elles sont représentées à la fois par des neurones et par des connexions appropriées vers les neurones représentant la base de faits. Les inférences sont effectuées grâce aux calculs effectués par l'ensemble de ces neurones, mais leur description précise prendrait une place trop importante dans ce mémoire.

L'une des propriétés intéressantes est la robustesse d'un tel système, qui continue à donner de bons résultats même si certains neurones sont supprimés (ou tombent en panne). Cependant, il s'agit d'un système très limité, et dans lequel la représentation distribuée conduit à divers problèmes dont de mauvaises unifications dans le cas de la version de DCPS qui comporte des règles avec variables [Belala 1992].

L. Shastri, membre de l'école de Rochester (qui regroupe des chercheurs s'intéressant surtout aux réseaux localistes), a étudié le problème de la réalisation d'une forme limitée de réseau sémantique avec des réseaux de neurones [Shastri 1987]. Il montre formellement que son réseau peut résoudre très rapidement une classe intéressante de problèmes de catégorisation (retrouver un concept à partir de certaines de ses propriétés) et d'héritage (déterminer les propriétés d'un concept). Son système est de plus capable de traiter les exceptions et les conflits d'héritage, et de tenir compte de données incomplètes ou incertaines. Toutefois, les neurones utilisés sont très complexes.

Enfin, il faut citer le système CRUCS [Brachman et McGuinness 1988] car il est un des rares systèmes hybrides faisant intervenir une représentation de connaissances orientée objet. L'objectif est d'améliorer la recherche d'informations dans un système de représentation de connaissances à base d'objets. En effet, d'après ces auteurs, la rigidité des mécanismes d'un système symbolique ne permet pas de recherches suffisamment souples, comme retrouver tous les objets « proches » d'une définition donnée. Malheureusement, le bilan de CRUCS est assez mitigé [Brachman et McGuinness 1988] : il a été possible sur de petites bases de construire un réseau permettant de répondre à des requêtes, mais l'obtention de bonnes réponses dépend du bon réglage des divers paramètres du réseau, ce qui ne permet pas l'extension de ce mécanisme à des bases plus importantes.

Réseaux localistes spécialisés

L'approche choisie ici se distingue des précédentes en ce qu'elle conduit à construire des réseaux localistes très spécialisés pour effectuer des traitements normalement réservés à des systèmes symboliques.

Un exemple caractéristique de cette approche est le travail de Pinkas [1992], qui veut exprimer le calcul des prédicats dans des réseaux fonctionnant sur le principe de la minimisation d'une fonction énergie. Étant donné une base de connaissances en logique du premier ordre et une borne k, il construit un réseau symétrique (un peu comme une machine de Boltzman ou un réseau de Hopfield) qui cherche une preuve pour une requête donnée. Si une preuve (basée sur la résolution) de longueur inférieure ou égale à k existe, alors le minimum global de la fonction d'énergie associée au réseau représente de telles preuves ; si aucune preuve n'existe, alors ce minimum global indique l'absence de preuve.

Un autre exemple est donné par le système OSIRIS, développé à Grenoble [Simonet 1984, Simonet et al. 1994], et qui est fondé lui aussi sur une représentation de connaissances de haut niveau, orientée objet. Une analyse très approfondie de la base de connaissances permet de construire un réseau de neurones à trois couches parfaitement déterminé (la topologie et les poids des connexions sont fixés à la construction), et dont les nœuds semblent effectuer des opérations booléennes. Ce réseau permet ensuite de classer des objets : les valeurs des attributs de ces objets sont présentées au réseau qui donne alors la « vue » (ensemble de classes) auquel appartient l'objet (fig. 2.9). Une version plus sophistiquée de ce réseau permet aussi de prendre en compte les incertitudes dues à des attributs inconnus. Dans ce nouveau réseau, les unités effectuent des calculs de probabilités.

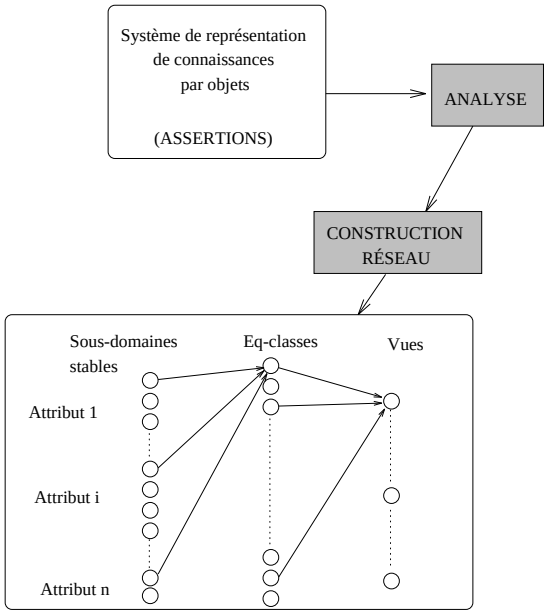


Fig. 2.9 - *Principe très simplifié du système OSIRIS.*

L'objectif principal de ces auteurs est la performance, de manière à pouvoir travailler

sur des bases de connaissances de grande taille. Aussi les réseaux de neurones sont utilisés uniquement pour leur rapidité, et le travail effectué sur OSIRIS se poursuit actuellement par une réalisation sur un calculateur parallèle (une machine SIMD, « Maspar 8192 »).

OSIRIS n'est pas dans la classe « émulation », car contrairement à CRUCS, il ne s'agit pas d'émuler tout le système symbolique, mais simplement d'effectuer une partie du traitement. À la limite OSIRIS pourrait être placé dans la classe des systèmes hybrides à couplage étroit et à sous-traitance, mais pour le moment ses auteurs ne mettent pas l'accent sur les interactions entre les deux modules.

2.3.2 L'approche distribuée

L'un des thèmes les plus étudiés dans ce type d'approche est la représentation distribuée de structures symboliques (listes, arbres).

L'un des systèmes les plus connus est BoltzCONS [Touretzky 1989], réalisé à partir de l'expérience du système DCPS. Il permet de créer et manipuler dynamiquement dans un réseau des structures comme des piles, des arbres ou plus généralement des graphes orientés. Le but de Touretzky est de voir ce qu'apportent les propriétés d'une implémentation connexionniste au traitement symbolique. Il pense que ce type d'implémentation pourra faire apparaître d'intéressantes propriétés émergentes, qui offriront de nouvelles possibilités au traitement symbolique. Par exemple, BoltzCONS permet un accès direct à un nœud d'un arbre étant donné son label (ce que ne permet pas la représentation par listes chaînées de Lisp) ; plus intéressant encore, il permet de retrouver une structure même à partir d'une description partielle.

L'architecture RAAM (pour « Recursive Auto-Associative Memory ») présentée par Pollack [1990] vise à résoudre l'un des problèmes importants auquel est confronté le connexionnisme : comment représenter ces structures de données récursives et de taille variable (tableaux, listes, arbres ...), que manipulent si facilement les programmes classiques, dans des systèmes connexionnistes ? Les systèmes connexionnistes ne manipulent en effet pas les notions de cellule de mémoire ou d'adresse, et de plus ont des nombres d'entrées/sorties déterminés a priori. RAAM permet d'obtenir des représentations réduites de structures de données complexes, sur lesquelles il est possible de faire des traitements (inférences). Récemment, Stolcke et Wu [1992] ont proposé un algorithme permettant l'unification sur des arbres codés par RAAM (en fait un autre réseau apprend à faire des unifications). Enfin, RAAM semble posséder certaines capacités de généralisation [Barnden 1992a, Jodouin 1993], ce qui lui permettrait de représenter des structures non explicitement apprises.

2.3.3 L'approche combinée

Nous consacrons une place relativement importante à cette approche car elle est la plus proche des travaux sur les systèmes hybrides.

Un réseau de réseaux

Plusieurs chercheurs pensent que cette approche combinée est une bonne façon de profiter des avantages des mondes symboliques et connexionnistes, ou plus exactement des mondes localistes et distribués. En effet, dans cette approche, un système symbolique sera codé sous la forme d'un réseau localiste, et sera couplé avec un réseau de neurones proprement dit, qui apportera son caractère distribué.

Par exemple, Lacher [1991] s'est tout d'abord intéressé à la traduction d'un système expert de type MYCIN (c'est-à-dire avec des règles et des faits portant des coefficients numériques) sous la forme d'un réseau de neurones, puis il a étudié des règles d'apprentissage adaptées à ce type de réseau, appelé réseau expert.

Cela permet de voir un réseau de neurones comme un système à base de règles et inversement, ce qui facilite la migration des idées entre les deux mondes (mais Lacher a une vision assez pauvre du système à base de règles). Toutefois Lacher n'est pas entièrement satisfait de cette approche purement localiste : bien qu'un réseau expert possède certains des attributs d'un système connexionniste, il effectue tous les traitements à un niveau symbolique (il est possible de donner une signification précise à l'entrée, au traitement et à la sortie de chaque nœud du réseau). Lacher souhaite donc se rapprocher d'un « vrai » réseau de neurones, en utilisant des nœuds qui soient les plus simples possibles. Pour cela, il propose de remplacer chaque nœud de type « et » par un petit réseau de neurones, qui apprendra à faire le traitement correspondant.

Cela permet de construire un réseau de réseaux, dans lequel il y a deux niveaux, l'un symbolique/localiste, l'autre sous-symbolique/distribué. Pour Lacher, le niveau symbolique représente une connaissance sur le domaine d'expertise, tandis que le niveau sous-symbolique représente une méta-connaissance (une connaissance sur l'utilisation de la connaissance du domaine). Cette partie du travail de Lacher reste hélas spéculative et ni l'intérêt ni même la faisabilité de son réseau à deux niveaux ne sont prouvés. Nous l'avons néanmoins inclus dans cet état de l'art car cette approche est fondamentalement différente de l'approche combinée la plus importante, celle de Sun [1991]. On peut toutefois retenir des travaux de Lacher l'idée que les réseaux de neurones peuvent offrir un cadre général pour réaliser à la fois des traitements symboliques et sous-symboliques. Cette idée nous semble importante car il s'agit d'une autre façon de voir les apports des réseaux de neurones au traitement de l'information.

Le système CONSYDERR

Description Nous avons déjà mentionné son auteur à propos du problème de la fragilité des systèmes symboliques. L'idée de Sun est également de se placer dans un cadre purement connexionniste, et de construire un système à deux niveaux, l'un localiste, qui est une sorte d'implémentation connexionniste d'un système expert symbolique, l'autre distribué, qui apportera en plus des capacités purement neuronales. Ce système s'appelle CONSYDERR (*CONnectionist SYstem with Dual representation for Evidential Robust Reasoning*) [Sun 1991, Sun 1992].

Rappelons que le cadre de travail de Sun est le raisonnement de sens commun. Il veut parvenir à reproduire certaines formes de raisonnement que font couramment les gens. Ces formes, identifiées et analysées par des psychologues, ont été décrites par onze petits protocoles consistant en une question et une réponse. En voici deux :

Q : Is the Chaco the cattle country?

R : It is like Western Texas, so in some sens I guess it's cattle country.

Q : Are there roses in England?

R : There are a lot of flowers in England. So I guess there are roses.

Certains systèmes, connexionnistes ou non, peuvent rendre compte de certaines de ces formes, mais aucun n'est capable de toutes les traiter dans un cadre unifié. Réaliser ce cadre unifié est le but que veut atteindre Sun pour résoudre ce problème important qu'est la fragilité (voir le chapitre 1). Il définit justement la fragilité comme l'incapacité à traiter divers aspects du raisonnement dans un cadre unifié, et ces divers aspects correspondent aux protocoles ci-dessus : par exemple, le premier protocole correspond à l'absence de règle applicable.

Pour obtenir ce cadre unifié, il montre que, bien que ces problèmes semblent assez disparates, ils peuvent tous être résolus par l'emploi de règles de production et d'une mesure de similarité conceptuelle utilisant les caractéristiques des concepts (voir tableau 2.1).

Tout le système CONSYDERR va donc reposer sur deux mécanismes, l'application de règles et le calcul de similarités. Ils sont programmés sur une architecture connexionniste à deux niveaux. Le premier, CL, est localiste (chaque nœud représente un concept précis) et sert à représenter des règles de production d'ordre 0. Sun a démontré que CL peut manipuler un sur-ensemble de la logique des clauses de Horn et de la logique modale de Shoham, et qu'il peut traiter des informations partielles et incertaines ; CL peut donc faire au moins aussi bien qu'un système symbolique. Dans CL l'opération de base est le calcul de sommes pondérées. Par exemple, soit la règle A_1 *et* A_2 *et* $A_3 \longrightarrow B$ où les A_i et B sont des propositions ; elle est codée dans CL en connectant les nœuds A_i à B et en faisant calculer par le nœud B l'activation $ACT_B = \sum_{i=1}^{3} W_i * ACT_i$, les poids W_i étant donnés[9].

Le deuxième niveau, CD, est distribué, dans le sens où un concept A de CL est représenté par un ensemble de nœuds dans CD, qui correspondent à ses caractéristiques de F_A (voir tab. 2.1). De même une règle de CL est représentée par un ensemble de liens dans CD. D'autre part, les caractéristiques représentées par les nœuds de CD peuvent être communes à plusieurs concepts de CL : par exemple *désertique* est une caractéristique du concept *Chaco* mais aussi du concept *Western-Texas* ; en fait le nombre de nœuds de CD communs à deux concepts de CL est proportionnel au

9. Sun reconnaît toutefois que l'obtention des W_i peut être délicate. Il propose de se placer dans le cadre de la logique floue ou bien des probabilités, ce qui permettrait de donner un sens précis au calcul effectué par le nœud B : en termes de probabilités conditionnelles, avec les bonnes hypothèses, le calcul ci-dessus peut en effet être vu comme $p(B) = \sum_{i=1}^{3} p(B|A_i)p(A_i)$. Mais il ne propose pas de méthode concrète et se contente de signaler que divers algorithmes d'apprentissage peuvent permettre de déterminer les $p(B|A_i)$.

Soient A et B deux concepts définis respectivement par les ensembles de caractéristiques F_A et F_B. Dans la suite, $|F|$ représente le cardinal de l'ensemble F. À chaque concept i est associée une valeur numérique appelée *activation* et notée ACT_i.

La mesure de similarité, notée $(A \sim B)$, est définie par :

$$(A \sim B) = \frac{|F_A \cap F_B|}{|F_A|} \in [0,1]$$

telle que si $ACT_A = a$ alors $ACT_B = a * (A \sim B)$ (si rien d'autre que A n'affecte B).

Une mesure sur les règles, notée $(A \longrightarrow B)$, est également utilisée. Elle est notée par $(A \longrightarrow B) = r \in [0,1]$ et est telle que si $ACT_A = a$ alors $ACT_B = a * (A \longrightarrow B)$. En fait la valeur r est donnée par le concepteur de la base de règles. Cette mesure peut être généralisée au cas où une règle a plusieurs prémisses.

Chaque cas de fragilité peut alors être exprimé avec ces notations et résolu. Par exemple l'absence de règle applicable se traduit par :

$$\begin{cases} A \sim B \\ B \longrightarrow C \\ \text{A est activé } (ACT_A \neq 0) \end{cases}$$

Alors on peut calculer $ACT_C = ACT_A * (A \sim B) * (B \longrightarrow C)$.

Tab. 2.1 - *Quelques détails sur la mesure de similarité conceptuelle utilisée par R. Sun.*

degré de similarité entre ces deux concepts. Cette propriété permettra le calcul de la mesure de similarité du tableau 2.1 par un simple jeu de propagations d'activités numériques dans le système.

Pour permettre ces propagations, les deux niveaux sont reliés : chaque nœud A de CL est relié à tous les nœuds de F_A dans CD (voir fig. 2.10). Les liens portent des poids appropriés : les bu et td de la figure 2.10. D'autre part, les règles codées dans CL par des liens inter-nœuds sont dupliquées dans CD, en reliant chaque caractéristique d'un concept prémisse à chaque caractéristique du concept conclusion : ce sont les liens étiquetés par les poids lw sur la figure 2.10.

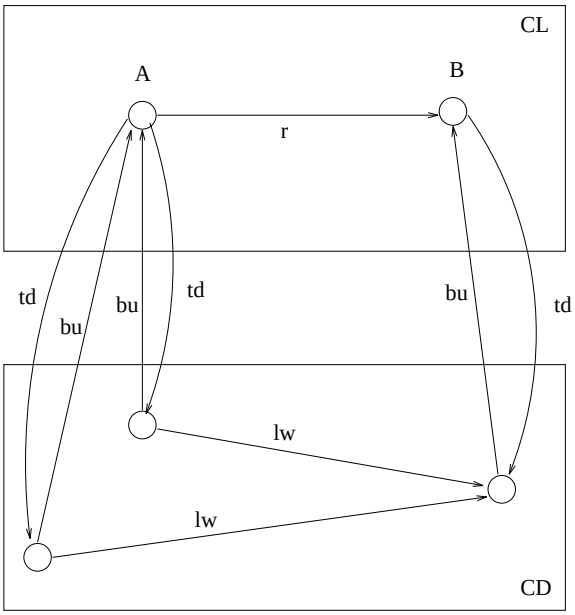


Fig. 2.10 - *D'après Sun [1991]. Représentation dans CONSYDERR de la règle $A \longrightarrow B$, de deux micro-caractéristiques de A et d'une micro-caractéristique de B. Les poids sont calculés formellement :*

$$td = 1 \ ; \ lw = \frac{r}{f(|F_A|)} \ ; \ bu_{dest} = \frac{1}{g(|F_{dest}|)}$$

où f et g sont des fonctions à choisir par l'utilisateur en tenant compte de certaines contraintes.

Tous les paramètres du système (dont les poids bu, td, lw) sont calculés de manière formelle (une formule analytique est obtenue pour chaque poids) à partir des contraintes imposées par les différents protocoles que veut traiter Sun (d'ailleurs, la mesure de similarité est elle-même choisie en fonction de ces contraintes). Nous décrivons ci-dessous le fonctionnement du système, de manière informelle (ce fonctionnement est gouverné par des équations précises, mais qui ont peu d'intérêt pour notre propos). Ce fonctionnement se compose de cycles de quatre étapes successives :

Réception des entrées : l'activation de certains nœuds de CL permet à l'utilisateur d'indiquer quels concepts sont présents (voir (1) sur fig. 2.11) ;

Phase CL $\longrightarrow$ CD : les activations des nœuds de CL sont propagées vers le bas, si bien que certains nœuds de CD sont activés (voir (2) sur fig. 2.11) ;

Phase de relaxation : dans les deux niveaux, les activations sont propagées « horizontalement », ce qui correspond à l'activation des règles. De nouveaux nœuds sont donc activés dans CL et CD (voir (3) sur fig. 2.11) ;

Phase CD $\longrightarrow$ CL : les activations de CD sont propagées vers CL. Les nœuds de CL prennent pour activation le maximum de leur activation à la fin de la phase de relaxation et d'une valeur calculée à partir de ce qui remonte de CD (voir (4) sur fig. 2.11). C'est l'utilisateur qui doit alors exploiter le résultat (*cattle-country*) pour répondre à la question posée.

Le principe général est donc assez proche de celui du système de Hendler : accent mis sur les micro-caractéristiques, propagations d'activités de la partie symbolique/localiste vers la partie connexionniste/distribuée suivies de propagations en sens inverse, calcul d'une mesure de similarité. Par contre il y a des différences importantes :

- dans CONSYDERR la mesure de similarité apparaît explicitement,
- dans le système de Hendler, une partie du système symbolique (l'évaluateur de chemins) sert à exploiter les résultats, tandis que dans CONSYDERR cette exploitation est à la charge de l'utilisateur,
- dans CONSYDERR les propagations sont uniquement numériques.

Évaluation CONSYDERR a été validé sur une petite application dans le domaine de la géographie : il s'agit du système GIRO, qui détermine le type d'utilisation agricole (*cattle-country, rice-growing area*, etc.) d'une région à partir de la donnée de son nom. Mais il s'agit d'une application jouet dont les résultats ne sont d'ailleurs pas présentés en détail.

L'une des limites de CONSYDERR est son faible pouvoir d'expression des connaissances. Pour augmenter ce pouvoir, R. Sun a proposé une extension qui permet de traiter des règles avec variables (par exemple $A(x, y) \longrightarrow B(x)$). Mais cette extension ne semble pas avoir été implémentée et conduit à des réseaux beaucoup plus complexes.

CONSYDERR tient une place assez importante dans la littérature, en raison du travail théorique et formel réalisé par son auteur et du nombre d'articles qu'il a publiés sur le sujet. Malgré cette place importante, le système est difficile à évaluer et il nous paraît significatif que pour le moment, ses principes n'aient pas été (à notre connaissance) repris par d'autres auteurs.

Selon nous, CONSYDERR a en effet plusieurs limites importantes. D'une part, il n'y a pas d'apprentissage : une fois réalisée, une application est complètement figée ; plus généralement le système n'a pas particulièrement de bonnes propriétés connexionnistes, comme la résistance au bruit ou bien à la suppression d'unité. Introduire une capacité d'apprentissage serait très difficile puisque tous les poids sont pré-calculés de manière formelle pour que les interactions entre CL et CD implémentent bien le calcul de similarité et l'application de règles. L'apprentissage serait pourtant indispensable pour que CD soit réellement un niveau distribué : en effet, pour le moment,

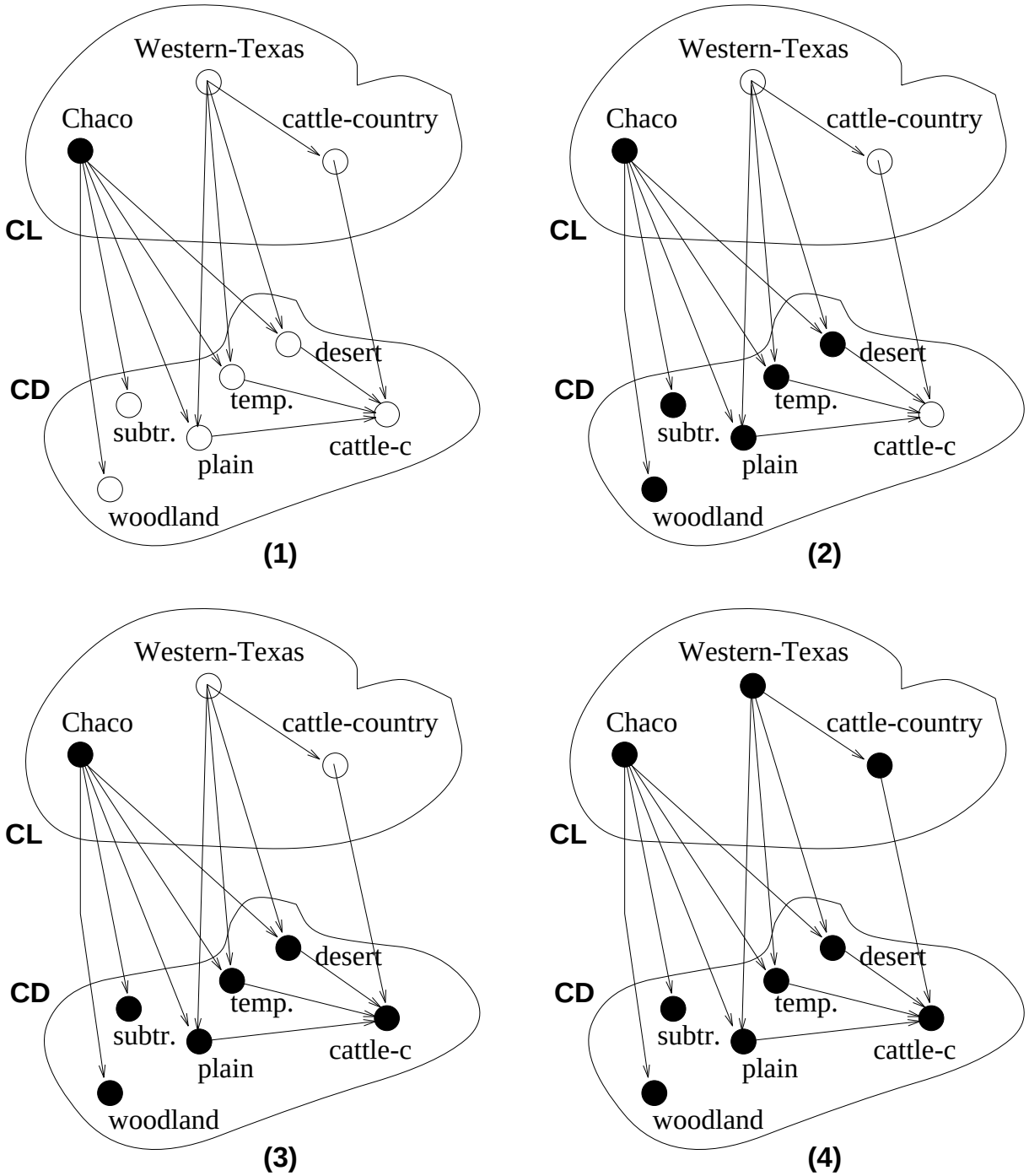


Fig. 2.11 - *D'après Sun [1991]. Résolution avec CONSYDERR du protocole « Chaco ». Commentaires dans le texte. subtr. = subtropical, desert. = desertic, temp. = temperate, cattle-c. = cattle-country.*

R. Sun a toujours présenté des exemples d'utilisation provenant de GIRO dans lequel chaque nœud de CD représente une caractéristique symbolique bien précise (comme *désertique* par exemple). Sun affirme que cela peut se généraliser facilement à des caractéristiques non-symboliques (généralement appelées micro-caractéristiques) mais cela nous semble douteux car ce qui est non-symbolique doit justement être appris automatiquement puisque l'on est incapable de l'identifier et de le nommer. D'autre part, il n'est pas évident que le système puisse s'appliquer à des problèmes de taille réaliste, d'autant plus que l'exploitation des résultats (qui sont sous la forme d'un ensemble de nœuds plus ou moins activés) est difficile.

Enfin, il reste une question majeure à propos de CONSYDERR : ne s'agit-il pas d'une simple implémentation connexionniste d'un système de nature symbolique ? Pour Sun, la différence essentielle entre CONSYDERR et un système symbolique, c'est que dans CONSYDERR les règles ne sont pas des éléments de connaissance dépourvus de contexte ; au contraire les règles forment des processus complexes qui interagissent de nombreuses manières. Dans CONSYDERR, il n'y a pas de structure de contrôle qui choisit les règles à appliquer et résout les conflits : toutes les règles, conflictuelles ou pas, peuvent être exécutées simultanément. C'est là aussi une différence essentielle avec des systèmes symboliques parallèles, qui permettent aussi l'exécution simultanée de plusieurs règles, mais non-conflictuelles. D'autre part, grâce à ces interactions, CONSYDERR a la capacité de générer de nouveaux concepts dynamiquement, et ainsi il n'est pas limité à ce qui lui a été explicitement donné. Par exemple, si l'on veut traiter le problème suivant :

> « If carrying cargo, buy utility vehicles. If carrying passengers, buy passengers vehicles. If carrying both cargo and passengers, what shall one buy? »

On utilise les deux règles

$$\begin{aligned} carrying - cargo &\longrightarrow utility - vehicles \\ carrying - passengers &\longrightarrow passengers - vehicles \end{aligned}$$

et différents types de véhicules sont représentés par leurs caractéristiques dans CD. Si les deux règles ci-dessus sont activées (en réponse à la question), toutes les caractéristiques correspondant à la fois aux véhicules utilitaires et de tourisme seront activées après la phase CL $\longrightarrow$ CD ; puis après la phase CD $\longrightarrow$ CL, tous les nœuds correspondant à l'intersection des véhicules utilitaires et de tourisme seront fortement activés (parce qu'ils ont toutes les caractéristiques) ; ainsi les nœuds de CL représentant quelque chose comme « van » seront les plus activés, alors qu'il n'y pas de règle explicite les concernant. Mais s'agit-il vraiment de génération dynamique de nouveaux concepts ? D'autre part Sun travaille toujours sur de tels petits exemples, dépourvus de contexte[10] : que se passe-t-il si l'on résout plusieurs problèmes en même temps ? Comment interprète-t-on les résultats ?

À notre avis, le système de Sun permet certainement de résoudre les onze protocoles mentionnés ci-dessus page 81, à condition probablement qu'on les pose séparément.

10. Si ses règles ont un contexte, ses exemples n'en ont pas.

Cela permet de mieux traiter certains aspects du raisonnement de sens commun. Par contre il ne s'agit pas d'une solution générale aux problèmes posés par les systèmes symboliques tels que la fragilité.

Toutefois, même si le travail de Sun semble difficile à utiliser pour résoudre des problèmes réalistes, il met en avant plusieurs idées intéressantes, comme celle d'étudier des mécanismes simples mais suffisamment puissants pour traiter différents aspects du raisonnement, ainsi que l'idée de réaliser ces mécanismes dans un cadre unifié. L'idée de réaliser un système symbolique (le niveau CL) sous forme connexionniste pour pouvoir interagir de manière très fine avec un système connexionniste (le niveau CD) est également importante et peut constituer une direction de recherche pour réaliser des systèmes hybrides à couplage fort.

Une autre architecture de type combinée a été présentée dans [Grumbach 1993]. Elle comporte en fait trois niveaux (symbolique supérieur, symbolique inférieur, sub-symbolique), mais seules les interactions entre les niveaux symbolique inférieur et sub-symbolique sont examinées, ce qui la classe bien, dans son état actuel, parmi les approches combinées. Nous décrivons superficiellement ce système, car sa description précise demanderait une discussion détaillée de la notion de « genèse du symbole artificiel » qui intéresse Grumbach [1993]. Le système permet d'illustrer concrètement une démarche qui consiste à permettre à une machine d'acquérir ses symboles par l'expérience et par l'interaction avec son environnement. Le domaine d'application est l'étude de la coordination sensori-motrice d'un mobile. Après un apprentissage que nous ne détaillons pas ici, le système fonctionne de la manière suivante :

- le système perçoit une image de son environnement,
- il associe via un RNA une action a (tourner à gauche par exemple) à cette image,
- en même temps,
  - un symbole S_p décrivant la perception est aussi associé à cette image, via un deuxième RNA,
  - via un troisième RNA, un symbole S_a décrivant l'action est associé à S_p,
  - enfin S_a est utilisé en entrée d'un quatrième RNA qui sert à renforcer ou à inhiber l'action a.

Ce système met donc en oeuvre une coopération entre niveaux sub-symbolique et symbolique, et permet d'exercer une sorte de contrôle symbolique sur les actions choisies par un RNA. Par rapport à notre problématique de construction d'une architecture multi-niveaux (chapitre 5), ces idées pourraient permettre de réaliser des interactions originales entre niveaux.

2.4 L'approche semi-hybride

Pour mémoire, nous mentionnons ici les systèmes effectuant des transformations complètes d'un module. Cela regroupe la compilation de base de règles en réseau,

l'extraction de règles à partir d'un réseau, le raffinement de règles à l'aide d'un réseau, la compilation et l'extraction de structures symboliques autres que des règles.

Par exemple, le système le plus représentatif, KBANN [Towell 1992], transforme des règles de production grossières en réseau multicouche à rétropropagation du gradient, qui apprend ensuite à partir d'exemples, et duquel on extrait ensuite de nouvelles règles, a priori plus précises que les précédentes. L'un des domaines d'application est la biologie moléculaire.

Le système SYNHESYS utilise des techniques de compilation et d'extraction de règles, parmi d'autres mécanismes, et illustre bien le fait que ces techniques peuvent apparaître dans les systèmes hybrides proprement dits.

2.5 Conclusion, discussion, perspectives

Nous avons présenté ci-dessus un état de l'art de l'intégration neurosymbolique. D'autres systèmes auraient sans doute mérité d'y apparaître, notamment des travaux concernant l'émergence de symboles dans les réseaux. Bien que très prometteur, ce thème est trop vaste pour être abordé ici, et les possibilités de réaliser des applications pratiques basées sur ce principe sont limitées dans l'état actuel des recherches. Aussi nous renvoyons le lecteur à des publications spécialisées comme [Amy et al. 1992].

Fallait-il se limiter dans cet état de l'art aux systèmes hybrides proprement dits ? Nous ne le pensons pas, car des idées originales provenant des systèmes purement connexionnistes (approche combinée notamment) peuvent certainement être utilisées pour faciliter les couplages dans les systèmes hybrides. D'autre part, il règne une certaine confusion au sujet de l'intégration neurosymbolique, et il nous paraissait indispensable de situer tous ces systèmes les uns par rapport aux autres.

Notre étude met en évidence plusieurs traits saillants des systèmes hybrides :

- l'omniprésence des réseaux à rétropropagation du gradient. Cela n'est pas surprenant car ce sont les réseaux les plus utilisés actuellement, mais ils ont des limites importantes, notamment le caractère empirique du choix de leur architecture et les difficultés qu'ils ont à réaliser un apprentissage incrémental. Ces réseaux sont donc presque toujours entraînés hors du SHNS, et sont ensuite utilisés comme approximateurs de fonctions (en particulier comme classifieurs ou comme contrôleurs). Assez curieusement, leurs performances ne sont presque jamais comparées à d'autres méthodes d'approximation ;
- seuls les systèmes à composants nettement séparés ont été appliqués à des problèmes réels ou réalistes ;
- les systèmes à couplage fort sont sans doute plus riches d'un point de vue théorique, mais restent difficiles à construire, mettre en œuvre et valider ;
- il existe une grande diversité d'approches, en raison notamment de la variété des systèmes symboliques et des réseaux disponibles actuellement ;

- toutefois, les systèmes symboliques employés sont la plupart du temps des systèmes experts simples voire simplistes ;
- les systèmes appliqués à des problèmes réels ou réalistes comportent souvent plus de deux composants, et mettent en œuvre plus d'un mode d'interaction. Le système du LIFIA, SYNHESYS, est l'un des plus complexes car il relève à la fois de l'approche hybride et de l'approche semi-hybride, du couplage étroit et du couplage fort ;
- la validation et l'évaluation des SHNS restent pour le moment relativement négligées : beaucoup de systèmes en sont restés au niveau de la proposition d'un couplage original.

Nous avons apporté des éléments de réponse à certaines des questions que nous posions en introduction de ce chapitre :

- nous avons établi une hiérarchie permettant de classer les systèmes hybrides et plus généralement les systèmes neurosymboliques ;
- nous avons mis en évidence les diverses techniques de couplage qui ont été imaginées ;
- les problèmes que les SHNS peuvent résoudre sont variés, mais pour le moment les approches les plus avancées du point de vue des applications sont aussi les plus simples : exploitation par un système expert symbolique des capacités d'approximation des réseaux de neurones.

Par contre les autres questions (rentabilité, maintenance, problèmes fondamentaux résolus, directions de recherche) ne sont pas abordées dans la littérature. Aussi nous proposons des éléments de réflexion ci-dessous.

2.5.1 Quelle répartition des tâches entre les composants ?

Pour construire des SHNS d'une manière moins empirique, il serait nécessaire de déterminer les répartitions des tâches entre composants symboliques et neuronaux les plus satisfaisantes, afin de disposer à l'avance d'un choix de couplages possibles face à une application. Qu'entendons nous par satisfaisantes ? Il n'y a pas de réponse unique, mais pour nous il s'agit des répartitions qui permettent de résoudre un problème avec un faible coût de réalisation et de maintenance, et surtout avec une supériorité claire sur les solutions qui ne seraient pas hybrides.

Les pages précédentes montrent que la répartition qui consiste à utiliser dans un système symbolique (système expert, programme Prolog, démonstrateur de théorèmes, etc.) un réseau de neurones comme une procédure est assez satisfaisante. En effet, les réseaux sont entraînés séparément, et apportent leurs capacités à effectuer les tâches que nous avons distinguées dans le paragraphe 1.3.2, capacités qui sont intéressantes par rapport à d'autres méthodes (paragraphe 1.3.7). Ces systèmes ne sont pas très difficiles à construire et permettent même de réutiliser des modules précédemment réalisés.

Cette répartition revient à donner des rôles bien différents à chacun des composants, chacun étant utilisé dans sa « spécialité ». Dans l'approche hybride, une autre répartition consiste à donner des rôles très voisins aux composants, comme dans RSA2 et dans SYNHESYS : les composants prennent les mêmes entrées et produisent des sorties similaires qu'il s'agit de choisir ou de combiner. Ce type de répartition est présent également dans l'approche semi-hybride. Dans les deux approches les systèmes sont beaucoup plus complexes puisqu'il faut gérer l'exécution des composants et exploiter leurs résultats. Le coût de la construction du système hybride est plus important. Ce type de répartition est-il satisfaisant ? Dans le cas de RSA2, il est difficile de le dire sans une étude très approfondie du système : il y a certes des résultats intéressants mais le système paraît énorme (trois bases de règles, plusieurs réseaux, d'autres modules). On est donc conduit à se demander si les résultats obtenus sont suffisamment importants et significatifs par rapport au travail qu'il a fallu fournir pour les obtenir. Dans le cas de SYNHESYS, une étude plus approfondie sera menée dans les chapitres suivants.

Enfin, nous avons vu que certains chercheurs tentent de faire des traitements symboliques, voire d'émuler des systèmes symboliques avec des réseaux de neurones. C'est l'approche purement connexionniste. On sort ici de la discussion sur la répartition des tâches entre composants, puisqu'il n'y a qu'un type de composant. Mais ces études visent à faire accomplir de nouvelles tâches aux réseaux, tâches qui pourraient éventuellement être exploitées dans de futurs systèmes hybrides.

2.5.2 Quels problèmes fondamentaux résolvent les SHNS ?

Tous les systèmes hybrides n'ont pas pour but de résoudre des problèmes fondamentaux. Beaucoup ont des motivations très pragmatiques et visent simplement à résoudre un problème avec le minimum d'efforts et de recherche. Quelques problèmes fondamentaux sont tout de même abordés, surtout par les approches connexionnistes (mais pas exclusivement) :

- représentation de connaissances avec des réseaux de neurones,
- ancrage des symboles,
- modélisation hybride de connaissances (voir SYNHESYS),
- fragilité des systèmes symboliques (voir CONSYDERR),
- coopération entre systèmes symboliques et non-symboliques.

Mais on ne peut pas dire que ces problèmes sont résolus à l'heure actuelle, et d'autre part les systèmes hybrides ne sont souvent qu'une tentative de solution parmi d'autres.

En fait, pour nous, le problème fondamental des SH est de parvenir à profiter des avantages de leurs composants sans en garder les faiblesses (au minimum, il faudrait que la somme des avantages soit supérieure à la somme des faiblesses). Mais cela reste difficile à évaluer, tout comme la rentabilité. Pour qu'un système hybride n'hérite que les avantages de chacun de ses composants, il faudrait être capable d'intégrer

l'essence des composants plus que les composants eux-mêmes, mais seuls Gould et Levinson [1991], dont nous avons décrit le système MORPH page 16, semblent s'être intéressés à ce point presque « philosophique ».

2.5.3 Quelles sont les directions de recherche intéressantes ?

L'examen de la littérature nous conduit à penser que la période de court terme évoquée par Hendler (p. 15) est maintenant terminée : de nombreux couplages simples ont été réalisés, et pour continuer à exister comme thème de recherches à part entière les SHNS doivent trouver un second souffle, en s'intéressant à de nouveaux problèmes :

- prise en compte du temps : les réseaux récurrents ainsi que les systèmes symboliques temporels ne sont pas utilisés dans les systèmes que nous avons étudiés ;
- conception de systèmes hybrides à couplage fort ;
- répartition des tâches ;
- réflexion sur la façon de ne garder que les avantages des composants ;
- le but de beaucoup de recherches a été d'apporter les capacités intrinsèques d'apprentissage des réseaux à des systèmes symboliques incapables d'apprendre, mais souvent en ignorant les études menées depuis longtemps dans le domaine de l'apprentissage automatique symbolique. Aussi il serait intéressant de développer l'intégration de mécanismes d'apprentissage symboliques et neuronaux [Hilario 1993], d'autant plus que les réseaux les plus utilisés (rétropropagation) ne permettent pas facilement l'apprentissage incrémental ;
- augmenter le pouvoir de représentation de connaissances des SHNS, dont le module symbolique est pour l'instant souvent limité à la logique des propositions ;
- conception d'architectures hybrides plus génériques : la plupart des systèmes hybrides (à l'exception notable de SYNHESYS) sont conçus de manière ad-hoc pour résoudre un problème précis, ce qui limite leur portée. Dans le projet Esprit MIX [Hilario 1993], un cadre multi-agent permettra de réaliser plusieurs architectures hybrides plus générales, et un compilateur pour un langage de description de systèmes hybrides est en cours de développement à l'Université Polytechnique de Madrid. D'autres chercheurs travaillent maintenant sur la réalisation d'environnements de développement spécialement adaptés aux systèmes hybrides [Wilson et Hendler 1993]. Cela témoigne de la vitalité de ce domaine de recherches. Toutefois il reste difficile de faciliter le développement de tous les systèmes hybrides envisageables, si bien que Wilson et Hendler [1993] préfèrent se limiter aux systèmes à composants bien séparés, et aux interactions de type sous-traitance. De plus, ils considèrent implicitement que le composant symbolique est le principal résolveur de problèmes.

Nous avons contribué dans ce chapitre à **structurer** ce jeune domaine de l'intégration neurosymbolique. Avant d'aborder les directions de recherche que nous mentionnons ci-dessus, il est indispensable d'exploiter l'existant et d'appliquer à des

problèmes réels des SHNS déjà développés. Nous décrivons un tel travail dans les chapitres 3 et 4. Forts de cette expérience concrète, nous proposons dans le chapitre 5 une architecture originale de SHNS qui servira à l'étude de plusieurs des directions de recherche que nous venons de décrire.

Chapitre 3

Étude du SHNS SYNHESYS

Dans ce chapitre, nous présentons une étude du système SYNHESYS, développé au LIFIA [Giacometti 1992]. C'est en fait un noyau de système hybride, qui a été utilisé dans plusieurs applications, notamment SHADE (aide au diagnostic en électromyographie) [Giacometti, Iordanova, Amy, Vila, et al. 1992] et SATAN (aide à la manœuvre navale) [Giacometti 1992]. Ce système hybride est très représentatif de l'état de l'art, car il utilise plusieurs des techniques que nous avons présentées dans le chapitre 2.

Cette étude de SYNHESYS a trois motivations :

- d'une part, l'importance de SYNHESYS dans le monde des SHNS, comme nous l'avons mentionné ci-dessus. Approfondir notre connaissance de ce système nous permet de compléter l'état de l'art entrepris dans le chapitre précédent, et partant de ce système particulier, de tirer des conclusions éventuellement valables pour d'autres SHNS ;
- d'autre part, SYNHESYS a été le point de départ de notre travail sur les SHNS et nous l'avons utilisé dans deux applications, décrites dans le chapitre 4. Aussi il est nécessaire de présenter ici les principes généraux du système ;
- enfin, SYNHESYS est un système de référence dans notre équipe de recherche, mais le travail effectué sur SYNHESYS est décrit dans plusieurs documents [Jacobsen, Iordanova, et Giacometti 1994, Giacometti 1992, Duval 1991, Broissiat 1991]. Aussi il nous semble nécessaire de donner une présentation synthétique du fonctionnement du système et de faire le point sur son état actuel, en regroupant des informations provenant de ces divers documents.

Une première partie (§3.1) présente le contexte du développement de SYNHESYS (étude psycho-mimétique de l'expertise), puis l'architecture du système est décrite d'une façon générale (§3.2) avant de passer à une description plus détaillée (§3.3, §3.4, §3.5). Nous étudions alors un mécanisme de couplage fort (§3.6) et nous mentionnons une étude en cours sur l'intégration de la logique floue dans SYNHESYS (§3.7).

3.1 Contexte du développement de SYNHESYS

SYNHESYS a été développé dans le cadre plus général d'une étude psycho-mimétique de l'expertise [Amy 1991], visant à étudier la démarche experte humaine, dans le but de la modéliser pour construire de meilleurs systèmes experts.

Giacometti [1992] explique les difficultés rencontrés par les premiers systèmes experts par le fait que leurs concepteurs tenaient généralement pour acquise l'hypothèse suivante :

> **Hypothèse H1 :** un expert est un sujet qui possède une théorie sur son domaine. Cette théorie met en jeu des lois générales ou règles qu'il manipule en situation d'expertise.

Cette hypothèse a conduit les cogniticiens à demander aux experts de formuler leur expertise sous forme de règles ou d'autres représentations de connaissances symboliques. Mais devant les difficultés rencontrés, notamment dans l'obtention de cette formulation, une hypothèse rivale a été formulée :

> **Hypothèse H2 :** un expert n'applique pas de règles (de manière consciente ou inconsciente). Il se borne à mémoriser et discriminer des milliers de cas particuliers.

La vraisemblance de H2 s'est trouvée renforcée avec le développement récent des travaux sur les réseaux de neurones[1]. Ces travaux ont notamment conduit à penser qu'il était possible de construire des systèmes experts ne manipulant pas de règles. Ils ont également conduit à mettre davantage l'accent sur les processus qui génèrent des compétences expertes, alors que l'approche symbolique mentionnée ci-dessus s'était plutôt concentrée, au moins dans un premier temps, sur la modélisation des connaissances expertes.

Considérant que l'hypothèse H2 est également insuffisante et que les modèles connexionnistes ont aussi des limites, A. Giacometti défend une troisième hypothèse, H3, qui cherche à concilier H1 et H2 au lieu de les opposer comme cela est parfois fait par leurs partisans respectifs. En effet, des chercheurs ont suggéré qu'opposer les systèmes symboliques et les systèmes connexionnistes ne correspondrait pas à la réalité des mécanismes mis en œuvre dans les traitements cognitifs [Amy 1991]. Pour ces chercheurs les traitements symboliques modélisés en IA et les traitements sub-symboliques modélisés par le connexionnisme se complètent et interagissent constamment. Cette troisième hypothèse s'énonce comme suit :

> **Hypothèse H3 :** 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[2] 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

1. Il faut aujourd'hui ajouter les travaux sur le raisonnement à partir de cas.
2. Toutefois, un psychologue préfère le terme d'« être différencié » [Mendelsohn 1992].

analytique	*versus*	holistique
contrôlé	*versus*	automatique
orienté par les concepts	*versus*	orienté par les buts
à base de règles	*versus*	à base d'exemples

Tab. 3.1 - *Huit modes de prise de décision pouvant intervenir chez un expert, regroupés en quatre oppositions. Tableau construit d'après l'étude de résultats de psychologie expérimentale faite par Giacometti [1992].*

l'interaction entre des modes de pensée et d'apprentissage différents, mais pouvant être complémentaires.

A. Giacometti poursuit alors son étude par l'examen de résultats d'expériences de psychologie expérimentale, qui permettent de montrer l'existence chez l'homme de différents modes de prise de décision (tab. 3.1) et d'apprentissage, et de montrer également leur complémentarité. Il aborde également le problème du passage du stade de novice au stade d'expert, et l'on retrouvera cette idée dans l'un des modes d'interactions de SYNHESYS. Cette réflexion basée sur la psychologie a donc pour but de valider l'hypothèse H3. Cette hypothèse étant validée, elle sert de justification à la construction de systèmes hybrides pour modéliser l'expertise, comme SYNHESYS.

Dans la partie suivante, nous allons présenter les principes généraux de ce système, entièrement programmé en C/Motif au LIFIA et disponible sur stations SUN et HP. L'ensemble du logiciel représente environ 20 000 lignes de C.

3.2 Architecture du système et principe de fonctionnement

Nous présentons ici une description très simplifiée du fonctionnement du système SYNHESYS pour donner un aperçu général du système.

La figure 3.1 présente l'architecture du système, composé de deux modules principaux. Le module symbolique (M_S) est en fait un générateur de systèmes experts (règles de production d'ordre 0^+) fonctionnant en chaînage avant et arrière. Le module connexionniste (M_C), présenté dans la section 3.3, est composé d'un réseau incrémental à base de prototypes, qui est une variante du modèle général présenté dans le chapitre 1.

Cette architecture classe SYNHESYS dans la catégorie des SHNS à **couplage étroit**, et le mode d'interactions est la **coopération** (voir chapitre 2).

La figure 3.1 met également en évidence le fait que les deux modules ont exactement la même entrée, appelée une **situation** (en fait un vecteur de caractéristiques booléennes et numériques), et produisent une décision (en fait un symbole dénotant une action).

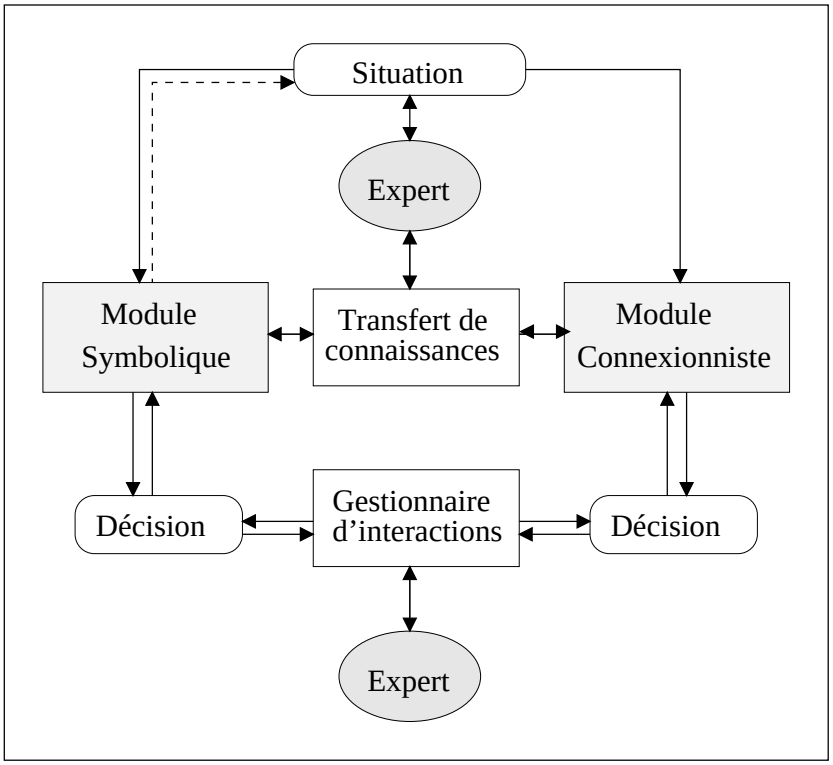


Fig. 3.1 - *Architecture générale de SYNHESYS. Commentaires dans le texte.*

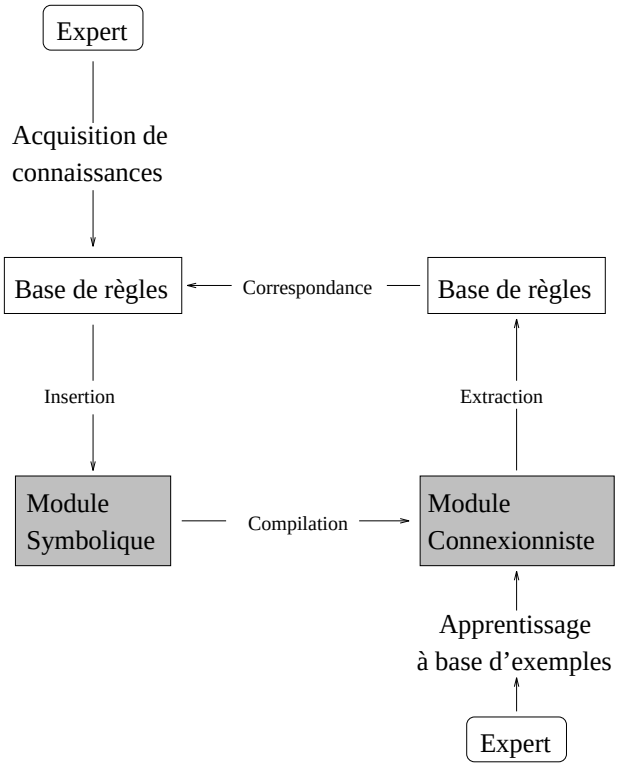


Fig. 3.2 - *Transferts de connaissances dans SYNHESYS. Commentaires dans le texte.*

Différentes utilisations du système hybride sont possibles à partir de cette architecture : on peut envisager de laisser les deux modules calculer en parallèle leur décision, et prendre la première ou la seule fournie (il y a des cas où aucune règle n'est applicable et d'autres où le réseau à base de prototypes ne donne pas de solution). Si aucune solution n'est fournie, le système hybride est en situation d'échec, mais par contre si deux solutions différentes sont données, il faut en choisir une selon certains critères. On peut encore utiliser le module symbolique pour valider les réponses du réseau, ou bien pour former ce dernier en lui fournissant des jeux d'exemples à apprendre.

En fonction de la gestion des interactions entre ces deux modules (voir section 3.4), de nombreuses utilisations sont donc possibles, ce qui met en évidence un premier intérêt fondamental des systèmes hybrides. Un module supplémentaire effectue cette gestion dans SYNHESYS, sous la supervision de l'expert ou l'utilisateur (fig. 3.1).

Ces nombreuses utilisations possibles sont encore augmentées par les possibilités de transferts de connaissances entre les deux modules (section 3.5), montrées par la figure 3.2. Selon le nœud par lequel on aborde le graphe de cette figure, divers scénarios sont possibles :

- l'expert donne une base de règles grossières, compilées ensuite sous la forme d'un réseau, qui apprendra à partir d'exemples complémentaires, et dont on extrait de nouvelles règles finalement mises en correspondance avec les règles initiales ; il s'agit de raffinement de règles, thème de recherches relevant de l'approche semi-hybride ;
- on peut aussi partir d'une base de règles vides, et apprendre uniquement à partir d'exemples ; cette fois il s'agit d'induction de règles ;
- enfin, on peut effectuer les opérations ci-dessus de manière itérative.

Enfin, des connexions (au sens neuronal) directes entre les deux modules permettent une forme de couplage plus fin, et qui a plusieurs intérêts : amélioration des performances du chaînage arrière, possibilité d'explications par « analogie » (section 3.6).

3.3 Le module connexionniste

Ce module permet d'exploiter **deux réseaux de neurones bien distincts**. Ces réseaux sont à base de prototypes comme ceux présentés dans la section 1.3.4, et la différence essentielle entre les deux provient de la nature des prototypes manipulés : dans le premier cas, il s'agit d'une variante des hyper-sphères de la section 1.3.4 (hyper-sphères à rayon variable), et dans l'autre cas il s'agit d'hyper-rectangles. Dans l'état actuel de SYNHESYS, **l'utilisateur doit choisir exclusivement l'un ou l'autre de ces réseaux**, en sachant que seul un réseau à base d'hyper-rectangles permettra d'extraire des règles symboliques.

Ces réseaux à base de prototypes ont été retenus lors de la conception de SYNHESYS

pour deux raisons :

1. ils sont **localistes**, ce qui permet d'identifier facilement chaque unité et finalement simplifie considérablement tout transfert de connaissance de ou vers un module symbolique,
2. ils sont **incrémentaux**, dans le sens où leur architecture n'est pas figée (le nombre d'unités de la couche cachée peut augmenter). L'avantage d'une architecture incrémentale est de permettre un apprentissage permanent, ce qui est une caractéristique importante d'un expert, et il semble souhaitable de reproduire cette caractéristique dans un système expert artificiel. De plus, les réseaux à architecture évolutive semblent très importants pour les applications réalistes [Jutten 1991, Perez, Hall, Romaniuk, et Lilkendey 1992].

3.3.1 Les réseaux à base d'hyper-sphères

Différences par rapport à l'algorithme de base

Par rapport à l'algorithme du tableau 1.1 de la page 38, l'algorithme utilisé dans SYNHESYS a les particularités suivantes :

1. il est utilisé de manière supervisée : chaque exemple de la base d'apprentissage doit être fourni avec la classe à laquelle il appartient. Il n'y a donc pas de phase d'étiquetage manuelle,
2. afin d'obtenir une meilleure capacité de généralisation que l'algorithme de base [Azcarraga et Giacometti 1991], SYNHESYS utilise une variante dans laquelle le rayon des hyper-sphères peut varier, ce qui introduit une notion supplémentaire par rapport à l'algorithme de base, la **différentiation** : lorsque le réseau donne une mauvaise réponse, ce qui veut dire que le prototype le plus activé classifie mal l'exemple présenté, le rayon d'activation de ce prototype est réduit de façon à exclure l'exemple en question, qui sera ensuite présenté une nouvelle fois au réseau (voir tab. 3.2).

Notons toutefois que ces nouvelles phases de différentiation obligent à présenter plusieurs fois de suite le fichier d'exemples [Jutten 1991], jusqu'à obtenir une stabilisation de l'architecture du réseau. En effet, la réduction du rayon d'influence pour exclure un certain vecteur peut conduire à une mauvaise classification de vecteurs auparavant bien classifiés, et il faut présenter à nouveau tout le fichier d'exemples pour corriger cet effet. Nous pensons que cela rend l'incrémentalité un peu plus délicate à gérer que ce qui avait été initialement imaginé (voir plus loin).

Amélioration des performances

D'après Broissiat [1991] cet algorithme a de bonnes performances, notamment si l'on dispose de peu d'exemples (ce qui semble arriver assez souvent dans la pratique). Par contre, près des frontières entre classes de décision, le nombre de prototypes

1. présenter un vecteur d'entrée X^t, **de classe** C,
2. pour tout prototype P_i, calculer la similitude entre son centre W_i^t et X^t : $m_i^t = M(X^t, W_i^t)$,
3. établir l'ensemble $E^t = \{P_i \mid m_i^t \geq \theta_i^t\}$,
4. – **si** $E^t = \emptyset$ Assimilation
   - créer un nouveau prototype P_n, avec $W_n^t \leftarrow X^t$ et $\theta_n^t \leftarrow \theta_0$
   - associer la classe C à P_n
   – **sinon**
   - choisir $P_s \in E^t$ tel que $m_s = \max_{P_i \in E^t} \{m_i^t\}$, et obtenir C_s la classe de P_s
   - **si** $C_s = C$ Accommodation
     rapprocher le centre de P_s de X^t :
     $W_s^{t+1} = W_s^t + g(t)(X^t - W_s^t)$
   - **sinon** Différentiation
     - **augmentation** du seuil de vigilance pour exclure X^t de la zone d'influence de P_s :
       $\theta_s^{t+1} = M(X^t, W_s^t) + \epsilon$
     - puis retourner en 2.

Tab. 3.2 - *Algorithme de la phase d'apprentissage des réseaux à base d'hyper-sphères dans SYNHESYS. Par rapport à l'algorithme de base (page 38), l'apprentissage est supervisé et les rayons des sphères peuvent varier, d'où la nouvelle phase de différentiation. θ_0 et ϵ sont des paramètres constants du réseau. Noter qu'augmenter le seuil de vigilance revient a réduire le rayon d'influence (voir fig. 1.5 page 37).*

1. présenter un vecteur d'entrée X^t, de classe **inconnue**,
2. pour tout prototype P_i, calculer la similitude entre son centre W_i^t et X^t : $m_i^t = M(X^t, W_i^t)$,
3. établir l'ensemble $E^t = \{P_i \mid m_i^t \geq \theta_i^t\}$,
4. – **si** $E^t = \emptyset$ **alors** pas de réponse
   – **sinon**
   - choisir $P_s \in E^t$ tel que $m_s = \max_{P_i \in E^t} \{m_i^t\}$, et obtenir C_s la classe de P_s
   - rendre comme résultat C_s

Tab. 3.3 - *Algorithme de la phase de classification des réseaux à base d'hyper-sphères dans SYNHESYS : il n'y a pas d'apprentissage, le réseau est simplement utilisé tel quel. Noter que ce réseau peut parfois ne donner aucune réponse, alors que d'autres modèles fournissent une réponse dans tous les cas.*

peut devenir très important. De même si certaines entrées ne sont pas pertinentes pour décider de l'appartenance à une classe donnée.

Le nombre très important de prototypes près des frontières entre classes a conduit à introduire un paramètre supplémentaire permettant d'exercer un certain contrôle sur la création de nouveaux prototypes. Ce nouveau paramètre, μ, intervient dans le critère utilisé pour savoir si le réseau est dans un contexte d'accommodation ou de différentiation (voir tab. 3.2). Ce critère était initialement $C_s = C$. S'il était vérifié, il y avait accommodation, sinon il y avait différentiation.

Le nouveau critère fait intervenir la position du vecteur de situation courant par rapport aux frontières entre les classes de décision (on parle ici des frontières implicitement modélisées par le module connexionniste à l'instant t). Il y a différentiation si le vecteur de situation courant n'est pas proche d'une frontière entre deux classes, ce qui revient à ne pas trop modifier les prototypes proches des frontières (ou encore à privilégier les accommodations par rapport aux différentiations, ces dernières étant des opérations assez perturbantes pour le réseau).

Étant donné que les frontières ne sont pas modélisées explicitement dans le module connexionniste, cette proximité du vecteur de situation est **estimée** en comparant les entrées des deux prototypes les plus actifs associés à des classes de situations différentes. Le nouveau critère est alors : $\exists\ k\ /\ |e_k(t) - e_s(t)| < \mu$ et $C_k = C$. Il y a différentiation si et seulement si un tel k n'existe pas. Sinon il y a accommodation.

Examinons plus en détail ce critère :

- supposons tout d'abord que $C_s = C$. Alors $k = s$, et il y a accommodation : on retrouve l'ancien critère,
- si par contre $C_s \neq C$: la réponse du réseau est mauvaise, et il y a deux situations :
  - soit il existe un k qui vérifie le critère : alors notre vecteur de situation courant X^t se trouve près d'une frontière entre classes de décision (parce qu'il est près de deux prototypes de classes différentes), et on décide alors de faire seulement une accommodation (on rapproche le prototype k de X^t),
  - soit un tel k n'existe pas : alors on considère que X^t est suffisamment loin de toute frontière, et dans ce cas on autorise la différentiation.

Les essais effectués montrent que ce nouveau critère permet effectivement de réduire le nombre total de prototypes, au prix toutefois d'une moins bonne approximation des frontières. C'est donc à l'utilisateur du système de choisir une bonne valeur pour μ, en fonction de ce qui est le plus important pour lui, soit la qualité de l'approximation des frontières, soit la réduction du nombre de prototypes (on peut par exemple désirer cette réduction si cela facilite des opérations ultérieures sur les prototypes, comme une extraction de règles par exemple).

Une question se pose toutefois : cette complication du modèle initial était-elle indispensable ? En effet, le besoin de ce mécanisme est apparu dans des essais réalisés avec des applications jouets, avec une répartition uniforme des exemples. Or dans

la réalité, du moins dans certains domaines comme la médecine [Rialle 1993], la répartition des exemples est loin d'être uniforme : les exemples sont tous proches des « centres » des classes et il y a très peu d'exemples près des frontières entre classes.

3.3.2 Les réseaux à base d'hyper-rectangles

L'un des objectifs de SYNHESYS est de permettre des transferts de connaissances entre les deux modules, et notamment l'extraction de règles symboliques à partir du module connexionniste. Une telle extraction est très difficile à partir des réseaux à base d'hyper-sphères qui ont été initialement utilisés dans SYNHESYS, et cela pour deux raisons :

- la région d'activation d'un prototype est le résultat d'une compétition avec tous les autres prototypes (fig. 1.5 p. 37) ; c'est donc l'intersection de plusieurs sphères, et une telle intersection est assez difficile à décrire et à manipuler,
- on pourrait éventuellement modifier la phase d'apprentissage de façon à interdire les intersections entre régions d'influence [Giacometti 1992], et dans ce cas la région d'activation serait égale à la région d'influence, qui est une hyper-sphère. Mais les seules règles que l'on pourrait obtenir seraient alors de la forme « **si** le vecteur de situation courant est inclus dans telle hyper-sphère **alors** telle décision ». De telles règles font intervenir systématiquement toutes les dimensions de l'espace d'entrée, et les concepteurs de SYNHESYS les ont jugées peu utilisables par un expert du domaine. D'autre part, de telles règles ne sont pas structurées, dans le sens où elles associent toutes une décision à une partie de l'espace d'entrée. Or il serait plus intéressant pour l'expert d'obtenir des règles faisant intervenir par exemple des conclusions intermédiaires. Enfin, il est souhaitable que les règles extraites puissent être insérées dans le module symbolique. Or ce dernier peut uniquement utiliser des règles qui permettent de représenter en intention des ensembles de vecteurs de situation pouvant être recouverts par des réunions d'hyper-rectangles. Les règles peuvent en effet avoir uniquement la forme **Si** T_1 et ... T_p **alors** X_k, où :
  - X_k est une proposition atomique, à valeur vrai ou faux,
  - $\forall\ i\ \in\ \{1, ..., p\}$, T_i est un test portant :
    - soit sur la vérité d'une proposition atomique X_j,
    - soit sur la valeur d'une variable s_j : $T_i = s_j \in [\alpha_j, \beta_j]$.

C'est donc pour permettre l'extraction de règles que les hyper-sphères associées aux prototypes ont tout simplement été remplacées par des hyper-rectangles, et les principales phases de l'algorithme d'apprentissage modifiées en conséquence (des traitements particuliers sont également effectués dans le cas de données booléennes) :

accommodation : le réseau reste maintenant inchangé,

assimilation : un nouveau prototype doit en principe être ajouté au réseau, mais dans la version actuelle de SYNHESYS on tente auparavant de faire **absorber** le vecteur de situation courant en dilatant la région d'influence d'un des prototypes existants, ce qui est possible à certaines conditions (voir [Giacometti

1992] pour plus de détails). Cette tentative d'absorption a pour but de réduire le nombre total d'hyper-rectangles pour faciliter ultérieurement l'extraction de règles,

différentiation : il faut ici réduire la région d'influence d'un prototype pour que le vecteur de situation courant, mal classé, en soit exclu. Comme la région d'influence est maintenant un hyper-rectangle, le principal problème est de choisir quelle face il faut reculer car il y a plusieurs possibilités. L'heuristique retenue tient compte à la fois de la face la plus proche du vecteur de situation mais aussi des unités voisines de l'unité à différencier. D'autre part, la différentiation est parfois impossible (quand un vecteur de situation est « trop à l'intérieur » de l'hyper-rectangle à différencier), l'hyper-rectangle est alors supprimé car on considère qu'il s'agit d'une erreur d'apprentissage. Broissiat [1991] souligne toutefois que cela est dangereux car il pourrait se produire des oscillations entre suppression et création.

Nos essais de SYNHESYS nous ont conduit à remarquer quelques problèmes liés à la programmation de ces trois phases, et à leur apporter des corrections qui sont indiquées en annexe A.

3.4 Les interactions entre les deux modules

Nous considérons ici les interactions gérées par le module appelé « gestionnaire d'interactions » ; d'autres interactions, les transferts de connaissances, sont possibles, mais ne sont pas du même niveau, et c'est pourquoi nous les décrirons dans la section suivante. Dans toute cette section, le module connexionniste peut contenir soit un réseau à base d'hyper-sphères, soit un réseau à base d'hyper-rectangles.

La figure 3.1 page 96 présentait l'architecture générale du système, et montrait clairement que les deux modules recevaient la même entrée, une situation, et donnaient une décision (aussi appelée jugement), et qui est en fait la classe à laquelle appartient la situation. Le gestionnaire d'interactions a pour rôle par exemple de déterminer quel module doit être appelé en premier, comment il faut utiliser sa décision, comment choisir une décision dans le cas où l'on en a deux différentes, etc.

Dans le cadre des modèles hybrides de l'expertise, trois contextes d'utilisation de SYNHESYS ont été définis, qui déterminent trois schémas d'interactions présentés fig. 3.3, 3.4 et 3.5.

Le premier contexte est « l'automatisation des prises de décision par le module connexionniste », fig. 3.3. M_C est toujours appelé en premier, mais M_S a toujours le dernier mot. Ce schéma a une certaine plausibilité psychologique, mais d'un point de vue purement informatique, ce schéma permet principalement de « former » un réseau de neurones, qui sera ensuite utilisé seul.

Le deuxième type d'interaction, qui a aussi une certaine plausibilité psychologique, s'appelle « contrôle des prises de décisions par le module symbolique », fig. 3.4, et c'est en fait un cas particulier du schéma précédent. Dans ce mode, M_S empêche M_C

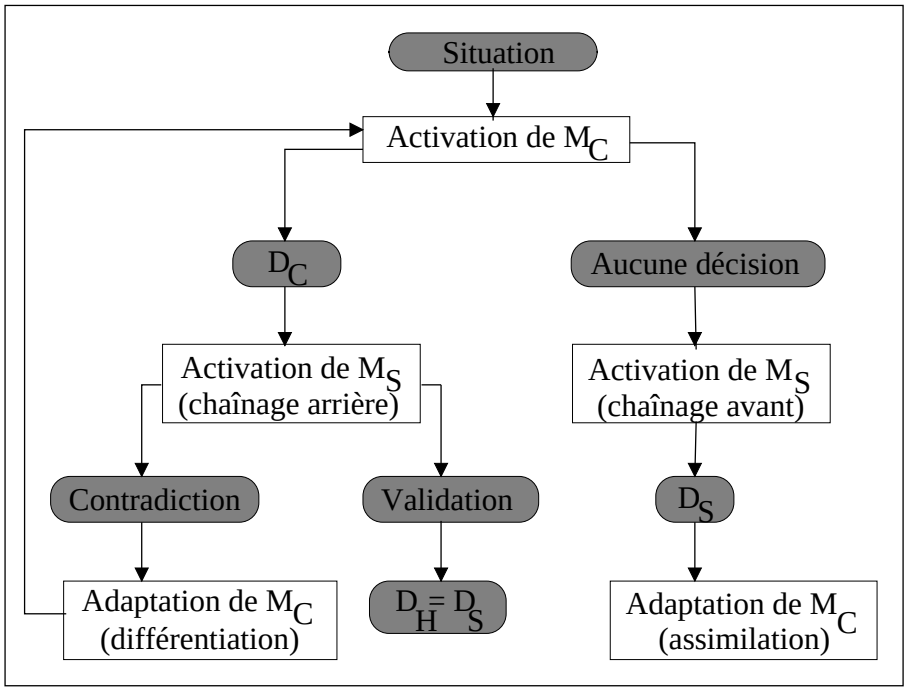


Fig. 3.3 - *Premier schéma d'interaction dans SYNHESYS, dans un contexte d'« automatisation des prises de décision par le module connexionniste ». D_S est la décision du module symbolique, D_C celle du module connexionniste, D_H la décision finale du système hybride.*

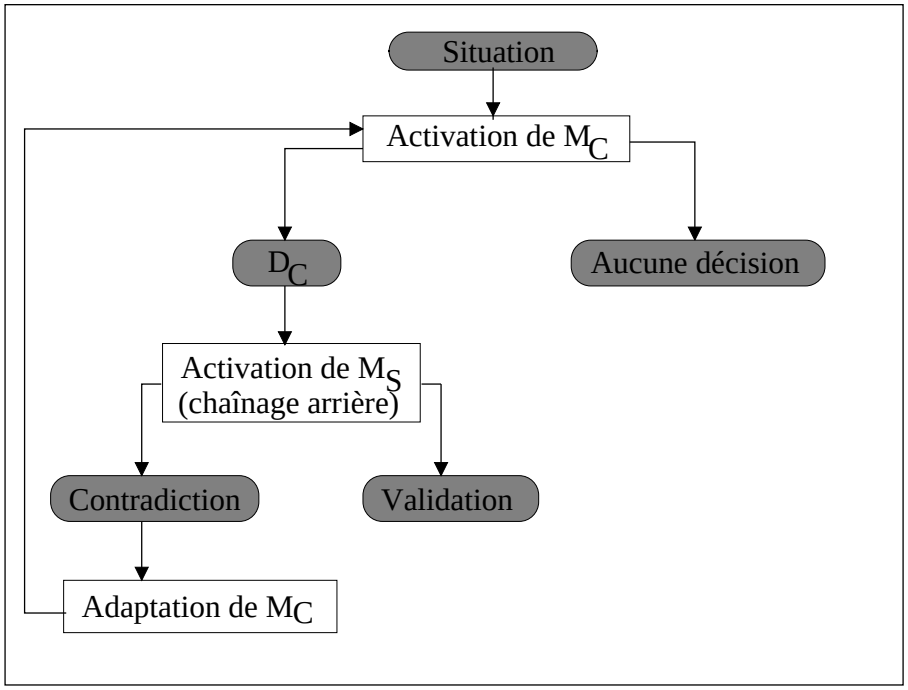


Fig. 3.4 - *Deuxième schéma d'interaction dans SYNHESYS, dans un contexte de « contrôle des prises de décisions par le module symbolique ».*

de donner des décisions contredisant des conditions exprimées explicitement dans le module symbolique. On dispose donc en quelque sorte de deux modes complémentaires d'expression de connaissances. Une application possible serait d'interdire certaines décisions d'un réseau de neurones dans un contexte de prise de décisions à haut risque.

Du point de vue de la modélisation de l'expertise, le troisième contexte (fig. 3.5) est probablement le plus riche de possibilités : il s'agit d'utiliser SYNHESYS comme « un système expert hybride incrémental[3] ». M_S permet à un expert d'exprimer des connaissances explicites sous forme de règles de production, tandis que M_C lui permet d'exprimer des connaissances non explicites sous la forme d'exemples de décision. Il faut noter que quelle que soit la solution fournie par le système, l'expert peut toujours reprendre la main et imposer sa propre solution pour corriger celle du système (dans ce cas, c'est M_C, seul composant capable d'apprentissage, qui est chargé d'apprendre la bonne solution). D'autre part, ce troisième schéma introduit un nouveau problème : dans certaines situations, M_S et M_C donnent deux solutions différentes, et ils sont donc en situation de conflit (voir fig. 3.5). Il n'y avait pas de telles situations dans les deux premiers schémas car un rôle prépondérant était donné au module symbolique. En cas de conflit, il faut trancher entre les deux solutions, et une méthode heuristique est utilisée dans SYNHESYS pour faire automatiquement ce choix.

M_C étant seul capable d'apprendre de nouvelles connaissances, il est probable que les situations de conflits deviennent de plus en plus nombreuses lors de l'utilisation du logiciel. C'est l'une des motivations de l'étude d'un mécanisme d'extraction de règles, qui permettra de « réactualiser » les connaissances du module symbolique.

Remarque Nos propres expériences avec SYNHESYS ont toutefois montré que l'incrémentalité était plus délicate à gérer que ne le laisse penser le schéma de la figure 3.5. Nous avons en effet signalé page 98 l'inconvénient du mécanisme de différentiation, qui oblige à présenter plusieurs fois l'ensemble des exemples pour stabiliser l'architecture du réseau. Les corrections faites par l'expert peuvent donc avoir des effets très importants sur le réseau (voir les figures 3.6, 3.7, 3.8 et 3.9), et il faut « corriger les corrections » en présentant de nouveaux exemples. On pourrait probablement résoudre, au moins partiellement, ce problème en imaginant des algorithmes d'apprentissage plus évolués, par exemple ne modifiant effectivement le réseau qu'après évaluation de l'impact de la modification (ceci est possible en utilisant deux versions du réseau, l'une modifiée, l'autre non modifiée, et en comparant leurs performances).

Cette remarque doit toutefois être nuancée par le fait que l'apprentissage humain présente de telles caractéristiques : l'apprentissage d'un nouveau concept peut entraîner l'oubli de concepts précédemment appris [Plunket et Marchman 1991]. Il n'est donc pas anormal de retrouver ce phénomène dans un système d'apprentissage artificiel.

3. Un système expert est dit **incrémental** s'il est capable d'accroître son expertise par l'expérience, par exemple par l'observation d'experts en situation d'expertise [Giacometti 1992].

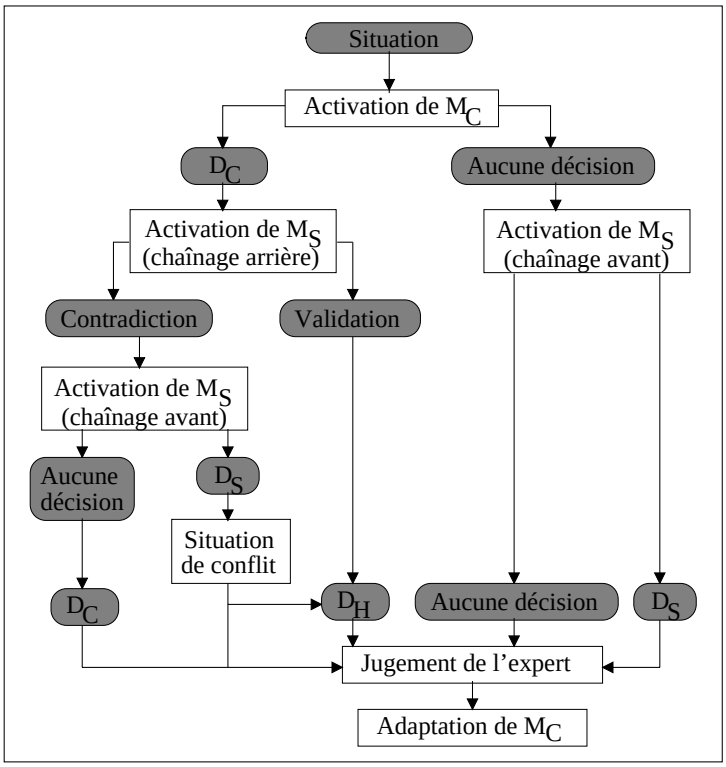


Fig. 3.5 - *Troisième schéma d'interaction dans SYNHESYS. Noter que l'expert peut toujours reprendre la main pour corriger la décision finale du système, et qu'il y a des situations de conflits.*

3.5 Les transferts de connaissances

D'autres interactions sont possibles, mais il ne s'agit plus ici de gérer l'enchaînement des modules ou de choisir une bonne réponse : il s'agit de transférer des connaissances d'un module à l'autre. Les transferts de M_S vers M_C sont simples : dans le cas où un réseau à base d'hyper-sphères est employé, des exemples sont générés aléatoirement grâce à M_S utilisé en chaînage avant, et sont appris par le réseau[4]. Dans le cas où les réseaux sont à base d'hyper-rectangles, il est par contre possible de compiler directement les règles sous forme de réseau, car ces règles ne servent finalement qu'à associer des classes à des réunions d'hyper-rectangles dans l'espace d'entrée. Au contraire les transferts de M_C vers M_S sont beaucoup plus complexes, car il faut non seulement extraire des règles à partir du réseau, mais il faut aussi les rendre exploitables par un expert en les simplifiant et les structurant. Dans cette section, nous allons uniquement développer ce transfert de M_C vers M_S.

A. Giacometti ayant déjà présenté ce processus d'extraction de manière formalisée, nous allons nous contenter ici d'en décrire les grandes étapes de manière intuitive, et d'en donner un exemple très simplifié (fig. 3.10).

4. La génération aléatoire d'exemples est une technique simple mais limitée car elle peut demander des connaissances de nature statistique sur la répartition des points dans l'espace d'entrée, et le nombre d'exemples à générer peut être très important dans les applications où l'espace d'entrée a beaucoup de dimensions : par exemple, l'espace perceptuel d'un robot peut avoir 24 dimensions [Reignier 1994a].

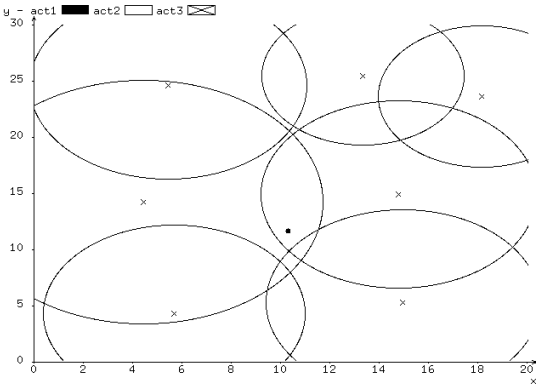


Fig. 3.6 - *Sept prototypes qui recouvrent l'espace d'entrée, le rectangle* $[0, 20] \times [0, 30]$. *Voir la figure de droite pour les décisions qui leur sont associées.*

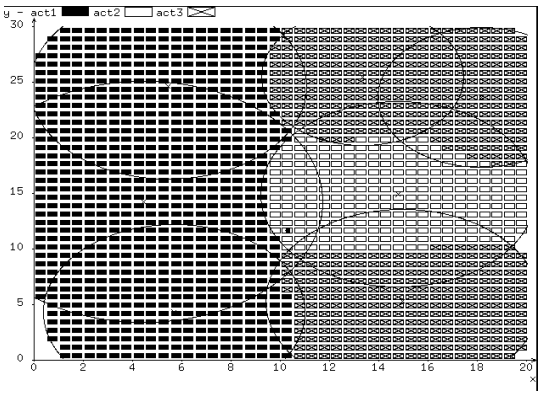


Fig. 3.7 - *L'espace d'entrée a été discrétisé en petits rectangles, et le centre de chacun des petits rectangles a été présenté au réseau : le type de remplissage d'un petit rectangle indique la réponse du réseau. On peut remarquer qu'à certains endroits le réseau ne donne aucune réponse (aucun prototype n'est suffisamment activé).*

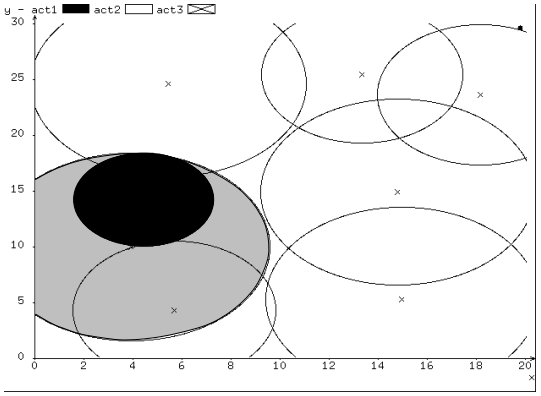


Fig. 3.8 - *Le point* $(4, 10)$ *a été présenté au système hybride, qui a donné comme diagnostic* $act1$. *Mais l'expert a corrigé cette décision, et a imposé* $act2$ *au réseau. On voit clairement ci-dessus la différentiation subie par le prototype en noir, qui a été réduit de manière à exclure le point* $(4, 10)$. *On voit également qu'un nouveau prototype a été créé (celui qui est en gris) pour assimiler le point* $(4, 10)$ *et sa décision* $act2$.

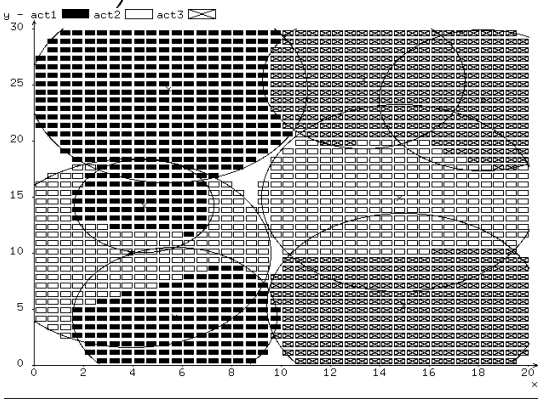


Fig. 3.9 - *Les réponses du réseau après cette différentiation. À plusieurs nouveaux endroits le réseau ne rend aucune réponse, et il donne* $act2$ *sur une zone plus importante que ce que l'on pouvait imaginer. Bien sûr l'effet est ici un peu caricatural, et pourrait être réduit si l'on prenait un petit rayon lors de la création d'un nouveau prototype. Mais cela aurait l'inconvénient de créer beaucoup de prototypes.*

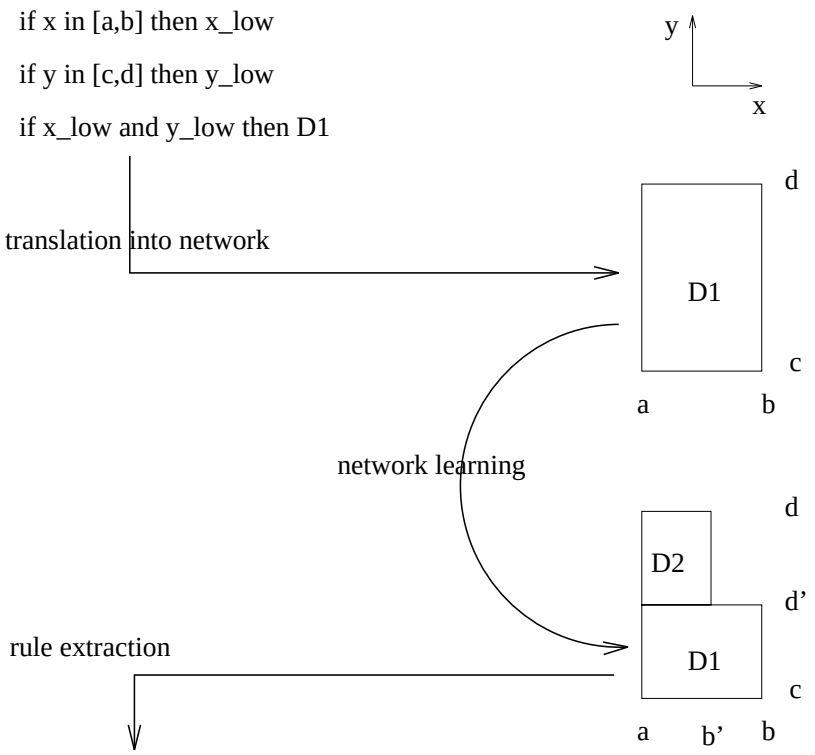


Fig. 3.10 - *Exemple de compilation d'une base de règles sous forme de réseau à base d'hyper-rectangles, puis d'extraction de règles une fois que le réseau a été entraîné sur un jeu d'exemples. Les règles extraites sont un raffinement des règles initiales, et la procédure d'extraction a tenté de réutiliser les symboles de la base de règles initiales.*

Étape 1 : description des régions d'activation La première difficulté de l'extraction de règles à partir d'un réseau à base d'hyper-rectangles (et aussi à base d'hyper-sphères d'ailleurs) est d'expliciter la région d'activation d'un neurone de la couche cachée, alors que cette région n'est définie qu'implicitement par les caractéristiques de tous les neurones de la couche cachée et par le mécanisme de compétition entre ces unités. Par contre la région d'influence d'une unité cachée est un hyper-rectangle et serait facile à expliciter. D'où l'idée d'adapter les mécanismes d'apprentissage de façon à garantir que la région d'activation soit égale à la région d'influence, en interdisant les intersections entre régions d'influence d'unités associées à des classes différentes. Ces mécanismes n'avaient pas été programmés dans la version initiale de SYNHESYS, si bien que nous y avons apporté quelques modifications décrites en annexe A.

À chaque hyper-rectangle $[\alpha_1, \beta_1] \times ... \times [\alpha_n, \beta_n]$ il est alors facile d'associer la règle « **Si** $x_1 \in [\alpha_1, \beta_1]$ **et** ... **et** $x_1 \in [\alpha_n, \beta_n]$ **alors** D_i », où D_i est la décision correspondant à l'hyper-rectangle. Toutefois l'extraction de règles ne peut pas s'arrêter là, car de telles règles ne sont pas structurées et font de plus systématiquement intervenir toutes les dimensions de l'espace d'entrée. Elles sont de trop bas niveau pour être réellement utiles. Les étapes suivantes ont donc pour but de simplifier et structurer ces règles.

Étape 2 : suppression des dimensions non pertinentes Ici, on cherche à supprimer dans les règles les tests sur les dimensions de l'espace d'entrée qui ne sont pas **pertinentes** pour une décision. Par exemple, si les valeurs pouvant être prises dans la j-ième dimension sont forcément comprises entre $VALMIN_j$ et $VALMAX_j$, la présence dans une règle d'un test $x_j \in [VALMIN_j, VALMAX_j]$ n'apporte aucune information, et dans ce cas la dimension j est dite non pertinente pour la conclusion de la règle.

Ce travail n'est pas fait sur les règles, mais directement sur les hyper-rectangles. La procédure consiste grosso modo à « étirer » au maximum chaque hyper-rectangle selon chaque dimension où c'est possible (voir [Giacometti 1992] p. 266 pour une définition rigoureuse de cet étirement ou plutôt extension). Ces extensions étant faites, on peut aisément détecter les hyper-rectangles dont une ou plusieurs faces recouvrent entièrement l'intervalle de valeurs autorisées dans une dimension, qui sont alors non pertinentes. La génération des règles correspondantes est alors très simple. Soulignons que SYNHESYS compare les dimensions pertinentes apparues durant ce processus et celles données par l'expert (implicitement sous forme de règles). Ceci constitue une information intéressante : par exemple, le système peut détecter que les dimensions x et y doivent être prises en compte pour décider de l'appartenance d'un point à une certaine classe alors que l'expert pensait que seule x suffisait.

Cette deuxième étape permet donc d'obtenir des règles qui ne font pas systématiquement intervenir toutes les dimensions de l'espace d'entrée, contrairement à celles que l'on aurait pu extraire à partir des hyper-sphères.

Soit un réseau à base de prototypes possédant quatre unités cachées. Ces unités définissent quatre concepts :

- $C_1 = (\ A_1,\ \{x \in [0,10],\ y \in [0,30]\}\ )$
- $C_2 = (\ A_2,\ \{x \in [10,20],\ y \in [0,10]\}\ )$
- $C_3 = (\ A_3,\ \{x \in [10,20],\ y \in [10,20]\}\ )$
- $C_4 = (\ A_4,\ \{x \in [10,20],\ y \in [20,30]\}\ )$

À partir de ces quatre concepts initiaux, SYNHESYS va définir quatre concepts intermédiaires, par une opération appelée « mise en relation » :

- $C_5 = (\ A_5,\ \{x \in [10,20]\}\ )$ par la mise en relation de $C_2,\ C_3,\ C_4$
- $C_6 = (\ A_6,\ \{y \in [0,10]\}\ )$ par la mise en relation de C_1 et C_2
- $C_7 = (\ A_7,\ \{y \in [10,20]\}\ )$ par la mise en relation de C_1 et C_3
- $C_8 = (\ A_8,\ \{y \in [20,30]\}\ )$ par la mise en relation de C_1 et C_4

Tab. 3.4 - *Un exemple de construction de concepts intermédiaires dans SYNHESYS, d'après Giacometti [1992].*

Étape 3 : structuration de la base de règles Toutefois, la suppression des dimensions non pertinentes n'est pas suffisante, car les règles, bien que simplifiées, sont toutes du même niveau : elles associent directement une décision à une zone de l'espace d'entrée. Il faudrait les structurer pour les rendre plus lisibles, par exemple en « factorisant » des informations communes. Il y a en fait deux situations : soit l'on dispose d'une base de règles initiale, présente dans le module symbolique, soit aucune base n'est disponible. Dans le premier cas, on peut tenir compte des informations fournies par l'expert (les concepts qu'il a employés) pour structurer la base de règles extraites (on parle dans ce cas de **mise en correspondance** des règles données et des règles extraites). Dans le deuxième cas, une méthode a été développée pour obtenir des concepts intermédiaires artificiellement. Informellement, la notion de concept retenue est la suivante : un concept est un couple (A,B), où A est un ensemble de situations et B un ensemble de propriétés nécessaires et suffisantes caractérisant complètement l'ensemble A. Il serait trop long de présenter plus en détail ces méthodes de structuration de la base de règles, aussi nous allons nous limiter à la présentation d'un exemple de construction de ces concepts intermédiaires (tab. 3.4).

3.6 Vers un couplage fort

Nous avons déjà vu deux types de circulation d'informations dans SYNHESYS :

- soit à travers une simple relation d'entrée/sortie entre les deux principaux modules,
- soit à travers des transferts de connaissances entre ces modules.

Un troisième type est également exploré avec SYNHESYS, par l'établissement d'une liaison directe entre les deux modules, permettant des échanges d'informations plus élémentaires.

3.6.1 Description

Pour réaliser cette liaison directe, un neurone est associé à chaque règle directement dans le module symbolique. Chacun de ces neurones, disons des R-neurones, est relié à tous les prototypes de M_C, par des connexions de poids initialement nul.

Lorsque M_S a validé une décision D_C de M_C, les R-neurones associés aux règles utilisées dans cette validation sont activés (leur activation est fixée à 1 tandis que celle des autres reste à 0). L'apprentissage consiste alors à faire en sorte que le poids d'une connexion entre un R-neurone U_R^i et un prototype U_C^j soit :

- d'autant plus renforcé que U_C^j est actif si U_R^i est actif,
- d'autant plus diminué que U_C^j est actif si U_R^i est inactif.

Ce réseau apprend donc à associer à un vecteur contenant les valeurs d'entrée des prototypes (autrement dit leurs similitudes respectives avec l'exemple présenté) un autre vecteur servant à repérer les règles utilisées durant la validation en chaînage arrière.

Ces connexions directes sont utilisées de la manière suivante : lorsqu'une situation est présentée à M_C, les similitudes obtenues au niveau des prototypes sont propagées sur les R-neurones par ces connexions. À chaque R-neurone et donc à chaque règle est ainsi associée une valeur d'autant plus grande que la situation présentée est « proche » de la règle. Ces valeurs sont alors utilisées pour ordonner les règles avant les validations par chaînage arrière ; ainsi les règles ayant les plus grandes valeurs sont examinées en premier par le moteur d'inférence (sinon elles seraient examinées selon leur ordre d'écriture dans la base). Le premier objectif de cette liaison directe est donc une amélioration des performances.

Ce nouveau mécanisme place SYNHESYS aussi dans la catégorie des SHNS à **couplage fort**. En effet il y a partage de structures de données puisque des neurones sont placés dans le module symbolique. Le type d'interactions reste la **coopération**.

3.6.2 Évaluation

Ce mécanisme nous semblant particulièrement original, nous avons cherché à évaluer cette amélioration. Pour cela, nous avons tout d'abord généré aléatoirement quatre bases de règles (la procédure utilisée pour cela est décrite en annexe B). Ensuite, pour chacune de ces bases, nous avons lancé 1 000 validations en chaînage arrière, **sans** le mécanisme de sélection des règles décrit ci-dessus, et nous avons compté dans chaque cas le nombre de règles examinées avant d'aboutir à une validation. En divisant par 1 000, nous avons ainsi obtenu un nombre moyen de règles examinées qui est indiqué dans le tableau 3.5. Nous avons ensuite créé un réseau pour chaque

Base de règles	Nombre moyen de règles examinées sans sélection	avec sélection	Économie réalisée (%)
Base 3 : 102 règles	54.4	20.6	62
Base 4 : 210 règles	96.0	35.3	63
Base 1 : 618 règles	298.5	126.7	58
Base 2 : 2000 règles	—	—	—

Tab. 3.5 - *Résultats de notre évaluation de l'amélioration des performances de M_S utilisé en chaînage arrière. Dans le cas de Base 2, les temps de calculs trop élevés ne nous ont pas permis d'obtenir de résultat. Avec le mécanisme de sélection des règles, le chaînage arrière examine environ 60 % de règles en moins.*

base, et entraîné ces réseaux en générant aléatoirement des exemples en chaînage avant, jusqu'à apprentissage correct d'environ 80 % de ces exemples. Enfin, nous avons recommencé les chaînages arrières **avec** le mécanisme de sélection des règles, et déterminé le nombre moyen de règles examinées (tableau 3.5). Soulignons que la création des réseaux peut demander des temps de calcul importants : dans le cas de notre plus grosse base effectivement testée (618 règles), il a fallu générer 15 000 exemples pour arriver à seulement 70 % de bonnes réponses (au lieu des 80 % espérés), et cela a conduit à la création de 881 prototypes, après plus de 7 heures de temps CPU consommé sur l'ordinateur le plus puissant du LIFIA (SPARC 690 biprocesseurs). Le logiciel utilisait en fin d'exécution 45 Méga-octets de mémoire vive. Cela nous a empêché de faire des tests sur des bases plus importantes. Le tableau 3.5 montre que le mécanisme de sélection permet d'examiner beaucoup moins de règles (environ 60 % en moins) avant d'arriver à une validation.

Toutefois, l'économie ainsi réalisée est compensée par le temps nécessaire pour ordonner les règles en fonction des valeurs de leurs R-neurones. Cela réduit donc un peu l'intérêt de ce mécanisme si on le considère sous le seul angle de l'amélioration des performances. Il est par contre possible que lors d'un chaînage arrière, un tel mécanisme conduise à examiner les règles dans un ordre plus « intelligent » que celui imposé par un ordonnancement arbitraire. Cela pourrait par exemple conduire à poser à l'utilisateur une série de questions paraissant moins arbitraire que ce que font généralement les systèmes experts durant un chaînage arrière.

3.6.3 Couplage fort et explications

Ce mécanisme peut être utilisé pour obtenir des explications que Giacometti [1992] a appelées « par analogie ». Par explication on entend ici la trace des règles qui ont permis de valider par chaînage arrière une décision de M_C. Cela permet d'obtenir une « explication » symbolique d'une décision connexionniste. Toutefois, comme nous l'avons indiqué plus haut, les deux modules peuvent se trouver en situation de conflit, si bien qu'une telle trace n'est alors pas disponible. Comme M_C a donné une décision, les R-neurones de M_S ont effectué leur calcul de valeurs. SYNHESYS construit alors une trace fictive à partir de ces valeurs, en sélectionnant à chaque

niveau parmi les règles envisageables la règle ayant la plus grande valeur. Cette trace fictive est appelée « explication par analogie », bien que « explication par similitude » semble plus approprié (la notion d'analogie sous-entend que l'on fait intervenir deux domaines de connaissances différents [Beaubouche 1994], alors qu'ici on reste dans le même domaine). Ce mécanisme est très intéressant, mais il demanderait une évaluation appropriée pour que l'on cerne mieux ses apports et ses limites.

3.7 Vers l'utilisation de la logique floue

L'intégration de la logique floue [Haton, Bouzid, et al. 1991] dans SYNHESYS a également été étudiée [Jacobsen, Iordanova, et Giacometti 1994], et il existe une version de SYNHESYS comportant un moteur d'inférence adapté à la manipulation de règles floues. Toutefois, le module connexionniste et les processus d'extraction de règles n'ont pas encore été adaptés.

La motivation de cette étude relative à la logique floue provient du fait que les experts en électromyographie qui ont travaillé sur SHADE [Giacometti, Iordanova, Amy, Vila, et al. 1992], une application de SYNHESYS, ont eu des difficultés certaines a déterminer les seuils intervenant dans les règles dites de premier niveau, qui permettent de transformer des variables quantitatives en variables qualitatives. La logique floue peut, dans une certaine mesure, simplifier la détermination de tels seuils[5]. Le module symbolique ayant donc été ainsi adapté, il faut dans un deuxième temps étudier les répercussions sur le reste de SYNHESYS.

En particulier les procédures d'extraction de connaissances doivent être adaptées, car l'un des principes de base de SYNHESYS est d'extraire des règles directement utilisables dans le module symbolique. Jacobsen, Iordanova, et Giacometti [1994] proposent une première procédure d'extraction qui consisterait à associer à chaque hyper-rectangle une règle floue. Cette procédure aurait l'avantage de faire apparaître explicitement dans les règles les intersections entre hyper-rectangles associés à des décisions différentes, et permettrait de plus de gérer rigoureusement ces intersections dans le module symbolique[6].

Toutefois cette première procédure ne permet pas de respecter un autre des principes de base de SYNHESYS, qui est de permettre la mise en correspondance des règles extraites et des règles données par l'expert. Aussi une deuxième méthode a été envisagée par ces auteurs, et son idée générale est de considérer chaque hyper-rectangle comme un sous-ensemble flou de l'espace d'entrée, et de le décomposer lui-même en n intervalles flous, où n est la dimension de l'espace d'entrée. Cela permet de faire apparaître explicitement tous ces intervalles, et devrait faciliter la réalisation d'une procédure de mise en correspondance voisine de celle qui existe déjà

5. Il faut toutefois noter que SYNHESYS, dans sa version de base, peut faciliter cette détermination : en effet, il permet de raffiner des règles grossières données par un expert, et en particulier en affinant les bornes des intervalles. Mais cette possibilité reste à expérimenter.

6. Rappelons que pour le moment, l'apprentissage effectué par le réseau à base d'hyper-rectangles **doit empêcher** de telles intersections, sans quoi des règles contradictoires seront extraites (voir annexe A)

dans SYNHESYS. D'autre part, Jacobsen, Iordanova, et Giacometti [1994] montrent qu'une telle approche permet de considérer tous les calculs effectués par le réseau **durant la phase de reconnaissance d'une situation** comme des opérations sur des ensembles flous, ce qui donne un début de cadre formel à ces calculs. Ces auteurs soulignent également que l'on pourrait aller encore plus loin en adaptant tout le fonctionnement du réseau (donc l'apprentissage aussi) à la gestion d'hyper-rectangles flous. Mais ils n'ont pas encore étudié les répercussions de cette idée, qui n'est pas sans poser de sérieux problèmes théoriques selon nous. En particulier, la deuxième méthode mentionnée ci-dessus revient à traiter les fonctions d'activations des neurones représentant les hyper-rectangles comme des fonctions d'appartenance au sens de la logique floue, telles que seul le centre de l'hyper-rectangle a un degré d'appartenance égal à 1. Or les centres des hyper-rectangles peuvent varier beaucoup durant l'apprentissage, voir même ne jamais se stabiliser. Les fonctions d'appartenance vont donc varier considérablement dans le temps et les conséquences théoriques de cela ne sont pas encore connues.

Les répercussions exactes sur le module connexionniste restent donc à étudier. Pour nous, dans l'état actuel des choses, elles peuvent conduire à une complication supplémentaire de ce module, ce qui ne nous semble pas souhaitable. Mais elles peuvent aussi conduire à des résultats théoriques permettant de mieux maîtriser le fonctionnement du réseau, et à ce titre elles sont à encourager.

3.8 Conclusion

Nous avons regroupé dans ce chapitre les divers composants et mécanismes de SYNHESYS, un SHNS dont les interactions sont de type coopération et qui réalise des couplages étroits et forts (selon les définitions du chapitre 2) :

- les modules symbolique (système expert d'ordre 0^+) et connexionniste (réseau à base de prototypes) ;
- trois schémas d'interactions entre ces deux modules ;
- les mécanismes des transferts de connaissances, qui peuvent être utilisés pour extraire et/ou raffiner des bases de règles ;
- le couplage direct entre les deux modules, permettant une amélioration des performances du chaînage arrière ;
- la version floue de SYNHESYS, qui permettra éventuellement de donner un cadre formel plus rigoureux aux algorithmes d'apprentissage.

À ce stade, notre contribution consiste en :

- la programmation et l'expérimentation du mécanisme d'interdiction des intersections indésirables entre les régions d'activation des hyper-rectangles ; afin d'alléger ce chapitre, ce travail est décrit en annexe A ;
- une évaluation quantitative du couplage direct entre les deux modules ; les résultats sont décrits dans ce chapitre mais les aspects les plus techniques sont

en annexe B ;

- diverses expérimentations de SYNHESYS montrant par exemple le caractère incrémental de l'apprentissage.

Plusieurs directions de recherche sont possibles, notamment la poursuite de la réflexion sur l'intégration de la logique floue. Toutefois, il est indispensable d'exploiter ce qui existe déjà. À ce stade, face à cette abondance de mécanismes, plus ou moins heuristiques, une question se pose : comment les valider et les évaluer? Une manière de le faire est de réaliser des applications de SYNHESYS.

La première application réaliste de SYNHESYS a été SHADE [Giacometti, Iordanova, Amy, Vila, et al. 1992] (Système Hybride d'Aide au Diagnostic en Électromyographie), réalisé principalement par A. Giacometti et I. Iordanova, en collaboration avec le docteur A. Vila du CHRU de Grenoble. De nombreuses bases de règles (obtenues à partir du système expert NEUROP [Besnard, Rialle, Vila, et al. 1991]) et d'exemples ont été construites, et cela a représenté un travail très important. En particulier, les règles de NEUROP étant d'ordre 1, il a fallu les transformer en règles d'ordre 0^+ pour que SYNHESYS puisse les utiliser. D'autre part, les diagnostics des médecins se faisant en plusieurs étapes successives de classification, chacune utilisant les résultats de l'étape précédente, SYNHESYS a dû être adapté pour pouvoir gérer plusieurs réseaux, chaque réseau correspondant à l'expertise d'une étape. Enfin, des problèmes demeurent, car les médecins utilisent souvent des fichiers de taille variable pour décrire un patient, et cela est difficilement pris en compte par un réseau de neurones qui impose un nombre fixe d'entrées.

Bien que le système fonctionne, les médecins travaillant sur le projet ne l'ont pas encore testé de manière systématique.

Quant à nous, nous avons travaillé sur deux applications, l'une indirecte et l'autre directe, de SYNHESYS, décrites dans le chapitre 4, qui nous ont permis d'approfondir cette étude.

Chapitre 4

Applications et bilan de SYNHESYS

Dans ce chapitre, nous présentons deux applications de SYNHESYS à des problèmes réels (sections 4.1 et 4.2), applications qui ont fait progresser notre étude de SYNHESYS.

En effet, la coopération avec le CNET (section 4.1), sur un problème de type industriel, a permis d'étudier les points suivants :

- vérification pratique de certains des avantages et inconvénients des systèmes symboliques et connexionnistes ;
- possibilités d'adaptation de SYNHESYS à la modélisation de l'expertise relative aux problèmes industriels complexes ;
- sélection des mécanismes de SYNHESYS pouvant être réutilisés dans ce type d'application.

Quant à notre deuxième application (section 4.2), en collaboration avec le LAMA (LAboratoire de la Montagne Alpine), elle consistait en une expérimentation beaucoup plus directe de SYNHESYS, et a permis d'étudier principalement la mise en œuvre de l'extraction de règles. Cette application se prolonge actuellement avec une deuxième version de SYNHESYS mieux adaptée à ce problème.

Finalement, nous utilisons ces deux expériences très concrètes pour proposer un premier bilan de SYNHESYS (section 4.3) dans lequel nous faisons le point sur la validation du système, nous proposons une réflexion sur plusieurs problèmes plus fondamentaux (caractère neuronal ou non des réseaux employés, contraintes imposées par l'architecture, importance accordée à l'extraction de règles), et nous concluons par les nombreux apports et originalités de SYNHESYS.

4.1 Proposition d'une architecture hybride pour le test paramétrique

Ce travail a été effectué dans le cadre d'une convention d'un an entre le LIFIA et le CNET [Orsier 1993b]. Cette convention avait pour but d'étudier la faisabilité de la réalisation d'un système expert neurosymbolique pour l'aide au développement de filières de fabrication de circuits intégrés. Pour cela, cette convention prévoyait de développer deux maquettes, l'une symbolique et l'autre connexionniste, et de comparer leurs avantages et inconvénients respectifs, puis de proposer une architecture possible pour un système hybride, à partir de celle de SYNHESYS. Le CNET fournissait l'expertise relative à l'application et le LIFIA son expérience en matière de construction de système hybride.

4.1.1 Contexte de l'étude : le test paramétrique

Cette étude se situait dans un contexte de développement de filières de fabrication de circuits intégrés. En schématisant, les responsables de ces filières cherchent à atteindre des valeurs cibles pour des paramètres mesurés sur les plaquettes sortant des chaînes de fabrication. Les moyens dont ils disposent sont par exemple des réglages de machines ou des essais de nouveaux procédés. Une fois les mesures effectuées (après une phase dite de test, fig. 4.1 et fig. 4.2), il faut les analyser pour juger de l'état de la filière, tirer les leçons des essais effectués et déterminer les prochains essais.

Plus précisément, les paramètres mesurés sont certaines caractéristiques électriques de composants élémentaires, actifs ou passifs, disposés sur les plaques. L'ensemble des valeurs obtenues par le testeur paramétrique permet de mesurer l'état de la filière technologique étudiée par rapport aux spécifications établies par les concepteurs de la filière. Le travail d'analyse est très complexe et ne peut être réalisé que par des experts de la technologie en question. Cette complexité provient notamment de :

- la complexité des relations existant entre les caractéristiques du procédé (par exemple réglages effectués) et les mesures électriques. Il est important de noter que les équations physiques sous-jacentes à ces relations sont généralement connues, mais que certaines ne sont exploitables que par des logiciels de simulation numérique,
- la variabilité du processus de réalisation, jamais stabilisé sur une filière en développement,
- le grand nombre d'étapes technologiques intervenant dans la fabrication d'une plaquette (plus d'une centaine),
- la durée de la fabrication des plaques (plusieurs mois),
- le volume important des informations disponibles, qui doivent être pré-traitées avec des méthodes statistiques.

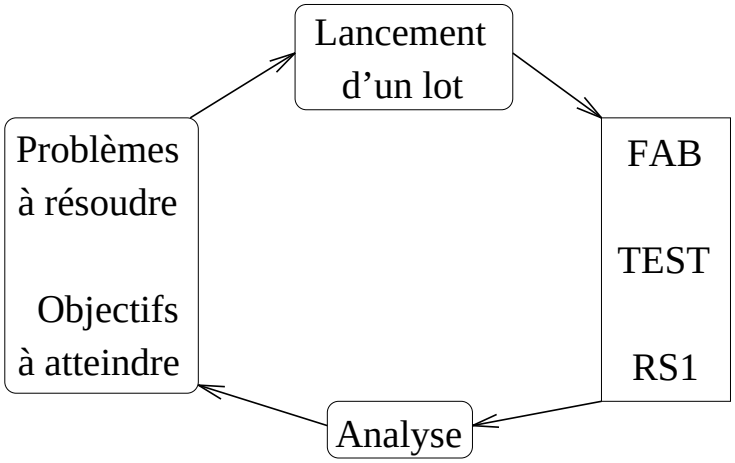


Fig. 4.1 - *Position de notre étude dans le contexte de développement de filières. FAB représente la chaîne de fabrication, TEST le test paramétrique, et RS1 et un logiciel de prétraitement statistique des données mesurées. Notre étude se situait au niveau de l'aide à l'analyse.*

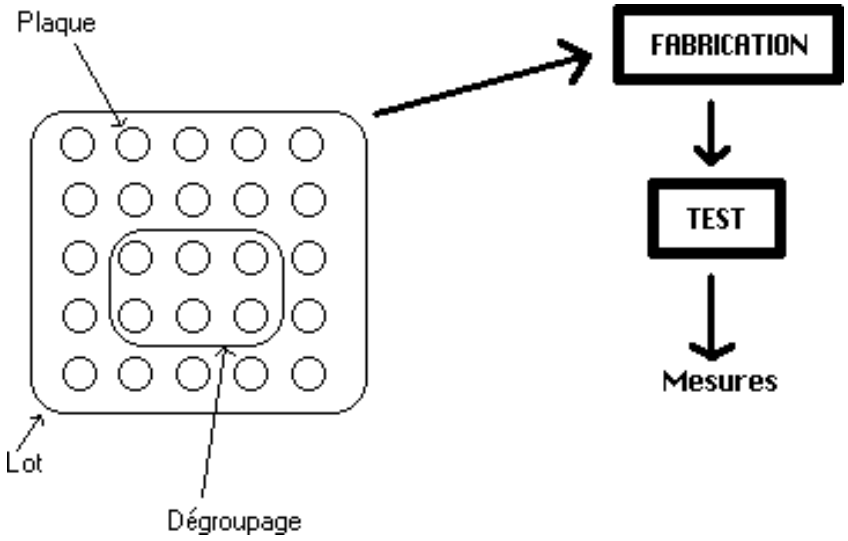


Fig. 4.2 - *Vocabulaire employé par les experts du CNET. Un ensemble de* **plaques** *(ou plaquettes, ou encore* wafers*) et les procédés de fabrication qu'elles subissent constituent un* **lot***. Toutes ces plaquettes ne parcourent pas les mêmes étapes technologiques, car un même lot permet de faire plusieurs essais et réglages. Par abus de langage, nous appellerons* **dégroupage** *un ensemble de traitements donné et le groupe de plaques qui l'ont suivi (le dégroupage est en fait l'opération qui consiste à séparer les plaques pendant la fabrication). Une* **filière** *peut être vue comme un ensemble de lots, réalisés autour d'un thème commun.*

4.1.2 Choix d'un sous-problème

Afin de pouvoir réaliser des maquettes représentatives, il a été nécessaire de définir un sous-problème.

Dans un premier temps, une filière particulière (DEMO 0.7), puis un composant élémentaire ont été sélectionnés par les experts du CNET. Le composant retenu est un transistor particulier qu'ils examinent souvent lors des analyses (transistor de W (largeur de grille) $20\mu m$ et L (longueur de grille) $0.8\mu m$). Plus tard, il a été nécessaire, afin de vérifier des hypothèses, de faire intervenir aussi un transistor plus grand (L = $5\mu m$), car il était souvent utilisé par les experts pour faire des comparaisons avec le premier.

Chacun de ces transistors est caractérisé par les paramètres électriques suivants :

- V_T, tension de seuil,
- I_{DSS}, courant de drain sous le seuil,
- I_{DSAT}, courant de drain en saturation,
- MN pente,
- ΔL, différence entre longueur de grille dessinée sur masque et longueur (de canal) mesurée électriquement.

Ces cinq paramètres permettent aux experts d'avoir une première idée de l'état du transistor, et on peut en première approximation considérer que le transistor fonctionne normalement (un point de fonctionnement est atteint) si ces paramètres sont corrects.

De même, les étapes technologiques intervenant dans la fabrication sont trop nombreuses pour permettre la réalisation d'une maquette. Aussi, les experts ont sélectionné les principales étapes influençant les paramètres ci-dessus :

- ajustement des tensions de seuil,
- oxyde de grille,
- formation de la grille,
- LDD,
- formation des espaceurs,
- drain-source N,
- formation du $TiSi_2$.

Enfin, pour terminer la description du sous-problème, il a été nécessaire de préciser nos objectifs, car nos discussions avec les experts du CNET ont montré que la phase d'analyse est trop complexe pour être traitée dans son ensemble par une simple maquette. En effet, cette phase comporte plusieurs aspects intimement liés : le diagnostic de problèmes de fabrication, la détection de ces problèmes, la confrontation avec les mesures et analyses des lots antérieurs, la consultation intelligente de ces volumineuses données antérieures, la préparation des futurs lots, l'optimisation des

réglages effectués, etc.

La seule détection d'un éventuel problème de fabrication demandera ainsi de prendre en compte la description des traitements suivis, leurs différences avec les traitements des autres dégroupages du lot ou avec des lots antérieurs, la préparation des futurs lots. De plus l'analyse d'un lot ne se fait pas de manière locale (dans ce cas, seules les informations relatives à ce lot seraient prises en compte), mais prend en compte un environnement et une histoire.

L'un des objectifs initiaux de la convention était le diagnostic de problèmes de fabrication, mais nos interlocuteurs du CNET Meylan étaient convaincus du peu de chances de réaliser une maquette capable d'intéresser les utilisateurs potentiels. Ce type de diagnostic ne leur semblait bien adapté qu'aux chaînes de fabrication bien stabilisées, lorsque la plupart des problèmes qui peuvent se poser sont répertoriés et que l'on dispose de mesures effectuées durant la fabrication. Or notre étude se plaçait dans le contexte de filières en développement (et donc dans lesquelles des portions importantes des traitements peuvent évoluer), et peu de données de fabrication fiables étaient réellement disponibles. D'autre part, l'expérience des concepteurs de deux systèmes de diagnostics de *wafers*, PIES [Pan et Tenenbaum 1986] et AESOP [Dishaw et Pan 1989], montre le bien-fondé de ce choix : ces deux systèmes montrent qu'il est impossible de pratiquer ce type de diagnostic de problèmes de fabrication avec des connaissances superficielles, et qu'il faut parvenir à intégrer dans le système de diagnostic de nombreux résultats de simulation numérique, à défaut de pouvoir travailler directement sur des modèles physiques.

Par contre, l'analyse des besoins de nos interlocuteurs du CNET a permis de dégager plusieurs objectifs pour un outil de résolution de problèmes. Ces experts étaient notamment intéressés par :

1. la représentation des lots et des connaissances qui en ont été extraites par les experts,
2. la représentation des filières (spécifications, traitements),
3. leur consultation, en vue de la préparation de nouveaux lots,
4. la confrontation des lots antérieurs avec le lot courant, afin de détecter des anomalies, soit dans le lot courant, soit dans les connaissances du système,
5. la prédiction de valeurs de paramètres, à partir de spécifications données pour une filière.

4.1.3 Réalisation de deux maquettes

Bien sûr, ces objectifs n'étaient pas tous envisageables dans le court terme. Nous avons choisi de travailler sur les points 1 à 4, notamment dans la maquette symbolique.

Maquette symbolique

La convention prévoyait d'utiliser un générateur de système expert commercial. Nous avons choisi le générateur SMECI [ILOG 1992], en raison de sa richesse fonctionnelle. Il s'agit d'un véritable environnement de développement offrant, entre autres, :

- une représentation de connaissances à base de catégories, de prototypes et d'objets,
- des règles de production d'ordre 1, organisées en bases de règles, elles-mêmes associées à des tâches ;
- un moteur d'inférence gérant un agenda de tâches ;
- un environnement de développement graphique multi-fenêtres, comprenant des éditeurs dédiés à SMECI, ainsi que l'environnement de développement d'interfaces graphiques Aïda,
- la possibilité de communiquer avec des logiciels écrits en Fortran ou C (important pour communiquer avec les divers outils statistiques utilisés par le CNET).

SMECI s'est révélé parfaitement adapté à la description des connaissances relatives au problème, qui étaient très structurées. Par contre, le fait qu'il s'agisse d'un gros système utilisé sur station de travail a empêché les experts de l'expérimenter seuls à cause de l'investissement qu'il fallait faire (apprendre quelques notions d'Unix et de Lisp). Pour cette raison nous avons développé des interfaces spécialisées.

En utilisant les capacités de représentation de connaissances par objets de ce générateur, nous avons écrit un ensemble de classes (environ 190) permettant de représenter l'ensemble des données généralement utilisées par les experts (lots, plaques, dégroupages, traitements suivis, etc.) que nous avons ensuite instanciées soit automatiquement à partir de données prétraitées par RS1, soit à la main (notamment la description des traitements). La maquette comporte ainsi environ 1700 instances facilement consultables et modifiables soit avec les interfaces de SMECI, soit avec des interfaces spécialisées que nous avons développées (voir par exemple la figure 4.3). Des détails sont donnés en annexe D.

Nous avons enfin écrit quelques règles permettant de vérifier si des relations inter-paramètres attendues par les experts étaient bien vérifiées par les données stockées dans le système.

La réalisation de la maquette symbolique a permis de vérifier certaines des difficultés d'un système symbolique (paragraphe 1.2.4), comme l'acquisition des connaissances et la prise en compte de données imprécises (les paramètres électriques mentionnés ci-dessus ne sont pas caractérisés par une valeur mais par une médiane et un écart-type). Nous avons pu également vérifier que les règles que nous écrivions étaient souvent interdépendantes, et aussi que parfois elles devenaient très techniques et trop éloignées des connaissances qu'elles modélisaient (en raison notamment de la nécessité de tenir compte des imprécisions). Bien sûr la maquette symbolique souffre aussi de difficultés plus générales comme la fragilité et l'absence de capacité d'adaptation. Par contre nous avons apprécié la facilité de prototypage, de mise au point

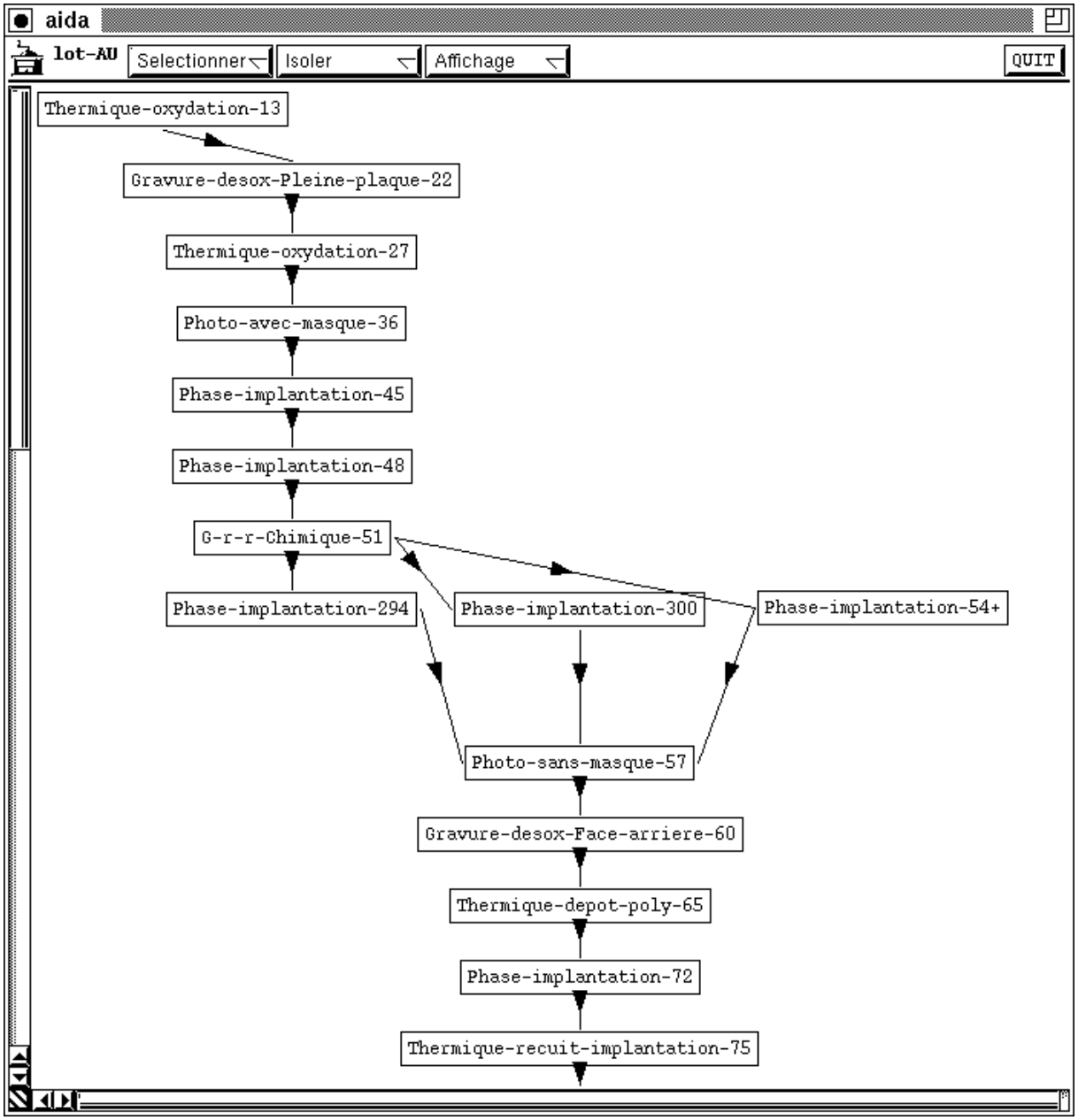


Fig. 4.3 - *L'une des interfaces spécialisées que nous avons développées pour la maquette symbolique réalisée pour le CNET. Cette interface permet d'afficher les dégroupages d'un lot et leurs parties communes. les menus permettent d'appeler diverses fonctions que nous avons réalisées, comme sélectionner un dégroupage particulier.*

et de développement, ainsi que la richesse fonctionnelle d'un générateur de systèmes experts commercial.

Maquette connexionniste

Principe : Les réseaux de neurones permettant de faire de la classification étaient bien adaptés à la résolution de certains des aspects du problème posé.

Nous avons utilisé les réseaux à prototypes de SYNHESYS pour faire de la classification de plaques à partir de leurs valeurs de paramètres électriques, mais des réseaux multicouches auraient tout aussi bien convenu pour une telle tâche, comme d'autres réseaux classifieurs d'ailleurs.

Dans notre modélisation du problème, exprimée sous forme de modèle symbolique (voir annexe D), chaque plaque est caractérisée par huit paramètres électriques. Chacun des paramètres est mesuré en 21 points sur une plaque, et l'on dispose de la médiane et de l'écart-type de ces mesures. Nous avons utilisé un réseau à prototypes dont les entrées sont les huit médianes, et nous avons entrainé le réseau à associer à chaque plaque (définie par les huit médianes) son lot. Il s'agit bien sûr d'une classification grossière, mais non dépourvue d'intérêt, car il est possible qu'une plaque ne suive pas exactement le traitement prévu (par exemple en raison d'erreurs de manipulation de la part des opérateurs en salle blanche). Aussi quand une plaque a des caractéristiques suspectes, les experts tentent de voir si elle n'a pas suivi par erreur un autre ensemble de traitements. Le réseau entraîné n'a toutefois pas été testé.

Bilan : La principale difficulté rencontrée était de devoir se limiter à un problème très précis comme la classification de plaques, alors que le problème posé était beaucoup plus global. La maquette connexionniste ne pouvait traiter que des étapes particulières d'un raisonnement plus global, nécessairement géré au niveau symbolique. Par conséquent, notre étude, en accord avec les experts, a principalement porté sur le niveau symbolique. La maquette connexionniste aurait éventuellement pu servir à montrer que les réseaux de neurones étaient mieux adaptés que le système symbolique au traitement des aspects statistiques du problème (valeurs manquantes ou aberrantes, grand nombre de valeurs).

4.1.4 Conclusion

La réalisation de ces deux maquettes a assez bien mis en valeur les avantages et inconvénients respectifs des mondes symboliques et connexionnistes, la plus grande difficulté rencontrée étant celle de l'acquisition des connaissances. En effet, l'expertise en test paramétrique n'est pas du tout formalisée, et les experts n'ont pas l'habitude de la communiquer. Aussi, il a été relativement facile de définir des structures de données permettant de décrire les objets sur lesquels porte une expertise, mais il a été impossible d'obtenir une méthode d'analyse des données du test paramétrique que nous puissions implémenter en SMECI. Cela était d'autant plus difficile dans

notre étude que nous étions confrontés à trois experts pas toujours d'accord sur les objectifs ni sur les méthodes.

Cette étude nous a mieux permis de cerner le champ d'application de SYNHESYS qui est celui de la classification de vecteurs de caractéristiques numériques ou booléennes. Ce système n'est pas bien adapté à la résolution de problèmes industriels complexes où les données sont souvent structurées et l'expertise complexe (elle exige par exemple de faire des calculs, éventuellement avec d'autres logiciels, en cours de résolution), mais SYNHESYS n'avait pas été prévu pour ce type de problèmes.

Les deux causes de l'inadéquation de SYNHESYS à ce type de problèmes sont le faible pouvoir de représentation de son module symbolique et le choix architectural qui consiste à imposer la même entrée et la même sortie aux deux modules (§4.3.2). Le remplacement direct du module symbolique par un module plus puissant n'est d'autre part pas possible : en effet, les principaux mécanismes de SYNHESYS dépendent de la forme des règles utilisées dans le module symbolique. Un tel remplacement exige donc la redéfinition complète de ces mécanismes.

À l'issue de cette étude, deux types d'architecture sont possibles pour ce type de problèmes industriels, où l'expertise est fort complexe :

- une **architecture de type sous-traitance**, avec un couplage étroit, permettant de continuer à travailler sur un problème assez global. Dans cette architecture le système symbolique serait chargé de représenter les connaissances structurées et de piloter le raisonnement, tandis que des réseaux classifieurs seraient entraînés sur des problèmes très précis puis appelés comme des procédures par le système symbolique. Au vu des systèmes présentés dans le chapitre 2, un tel système a de bonnes chances de réussite, à condition de pouvoir mettre au point une démarche représentable sous forme symbolique. Par contre une telle architecture serait très différente de celle de SYNHESYS et il serait difficile de réutiliser des mécanismes de SYNHESYS ;
- l'**architecture de SYNHESYS**, qui permettrait de se focaliser sur des problèmes plus restreints du type de la classification des plaques. Bien que plus restreint, ce type de problème est tout aussi intéressant et permettrait d'exploiter toutes les possibilités de SYNHESYS comme l'extraction et le raffinement de règles. De tels mécanismes sont intéressants dans les situations où l'expertise n'est pas immédiatement disponible : en effet, par des étapes successives de raffinement il est possible d'arriver à une base de règles satisfaisante.

Dans la section suivante, nous présentons une étude consécutive à celle-ci, et qui a justement consisté en l'application de l'**architecture de SYNHESYS** à un autre problème.

4.2 Application de SYNHESYS en géographie alpine

4.2.1 Introduction

Ce travail a débuté dans le cadre d'une convention entre le LIFIA et le LAMA (LAboratoire de la Montagne Alpine), qui appartient à l'Institut de Géographie Alpine de l'université Joseph Fourier. Le but de cette convention était pour les géographes d'obtenir une assistance dans l'utilisation de techniques de résolution de problèmes, notamment de systèmes hybrides neurosymboliques, et le but pour le LIFIA était d'obtenir des problèmes réalistes pour tester les architectures de type SYNHESYS (et aussi de disposer d'experts suffisamment disponibles, ce qui est souvent le plus difficile).

Cette coopération avec le LAMA a ensuite été étendue à la Chambre d'Agriculture de l'Isère, afin de disposer de nouvelles données collectées par la Chambre et de l'expertise de ses ingénieurs. Cette nouvelle convention a été notamment financée par la Région Rhône-Alpes. Ces travaux sont encore en cours.

4.2.2 Le problème

Nos interlocuteurs du LAMA s'intéressent au problème général de la dynamique de l'évolution des terres agricoles en zone de montagne, et cherchent à mettre en évidence les divers facteurs (bio-physiques, fonciers, humains, etc.) qui participent à cette dynamique. Ils s'intéressent en particulier au problème de la déprise agricole, mot qui a diverses acceptions : « repli agricole », « exode rural », « abandon et sous-exploitation », « progression des ligneux » [Josselin et Orsier 1993].

Comprendre les facteurs qui concourent à la déprise agricole est particulièrement important pour les collectivités locales et certaines institutions agricoles de la région Rhône-Alpes. Pour celles-ci l'enjeu est de favoriser la mise en place de modes d'exploitation novateurs mais viables, intégrant à la fois des pratiques agricoles productives et une gestion cohérente de l'environnement montagnard.

Étant donné la complexité des phénomènes étudiés, des outils informatiques sont indispensables : systèmes d'informations géographiques (SIG), techniques statistiques, intelligence artificielle. Les techniques statistiques restent incontournables mais leurs résultats demandent souvent un important travail d'interprétation, et les interprétations sont parfois subjectives et contestées (deux personnes différentes peuvent parvenir à des conclusions opposées à partir de la même analyse). L'intelligence artificielle (les systèmes experts principalement, mais aussi les systèmes multi-agents) est utilisée par les géographes depuis quelques années seulement.

Dans le cadre de notre convention, les géographes étaient surtout intéressés par les méthodes d'extraction et de raffinement de règles offertes par SYNHESYS :

- l'extraction, afin de comparer les règles obtenues par un système informatique

et les règles données par des experts ;

- le raffinement, pour essayer d'améliorer les règles données par les experts en tenant compte d'exemples de décisions prises par les agriculteurs.

Nous avons travaillé en deux étapes :

1. application de SYNHESYS à un premier jeu de données fourni au LAMA par le CEMAGREF, dans une optique d'extraction de règles (paragraphe 4.2.3). Cette première étape a surtout mis en valeur les limites, par rapport à ce type de données, de la version initiale de SYNHESYS, appelée SYNHESYS V1 par la suite ;
2. application d'une nouvelle version de SYNHESYS (SYNHESYS V2) à un deuxième jeu de données fourni par la Chambre d'Agriculture, dans une optique de raffinement de règles. SYNHESYS V2 est en cours de réalisation au LIFIA, et devrait tirer parti des remarques et recommandations que nous avons faites suite à notre première application. Étant donné que nous n'avons pas travaillé directement sur SYNHESYS V2 et que cette deuxième étape est en cours, elle ne sera pas présentée dans ce mémoire.

4.2.3 Extraction de règles (SYNHESYS V1)

Données

Le premier jeu de données avait été fourni au LAMA par le CEMAGREF. Ce fichier décrit 400 parcelles de terrain appartenant à 8 communes iséroises de moyenne ou haute montagne considérées comme représentatives de la diversité des massifs. Chaque parcelle était utilisée il y a 15 ans. Chacune est décrite par 59 variables hétérogènes de par leurs types (qualitatives ou quantitatives, continues ou discrètes) et par leur niveau de collecte (parcelle, exploitation contenant la parcelle, commune contenant l'exploitation). Ce fichier a trois caractéristiques importantes :

- les parcelles ne sont que très rarement contigües et de toute manière leur localisation n'a pas été conservée dans le fichier ;
- les données ont été collectées par enquêtes verbales, si bien que leur véracité n'est pas garantie ;
- les valeurs exactes des variables quantitatives ne sont pas connues car elles ont été saisies par classes durant l'enquête : par exemple l'âge de l'agriculteur n'est pas connu précisément, on sait seulement que ce dernier est dans la tranche d'âge 20-35ans.

Ce premier jeu de données était surtout destiné à tester SYNHESYS et mettre au point nos méthodes de travail. Aussi le fichier ci-dessus a été considérablement simplifié, en tenant compte d'études statistiques faites par le LAMA qui ont permis de sélectionner les variables les plus importantes. Le fichier utilisé pour les tests de SYNHESYS contenait les 400 parcelles décrites seulement par 5 variables dont une

variable décision, EVU. EVU représente la variation de l'utilisation de la parcelle en 15 ans, et possède 5 modalités :

REPR : reprise, c'est-à-dire passage par exemple de pré de fauche ou de prairie entretenue à culture, ou de prairie non entretenue à pré de fauche, ou de friche à prairie entretenue,
EMPR : emprise, c'est-à-dire stabilité d'utilisation,
DEPFA : faible déprise, c'est-à-dire passage à un niveau d'utilisation « inférieur » : culture à pré de fauche, pré de fauche à prairie entretenue, etc.,
DEPMO : déprise moyenne, c'est-à-dire passage à deux stades d'utilisation « inférieurs » : culture à prairie entretenue, etc.,
DEPFO : forte déprise, c'est-à-dire passage à trois stades d'utilisation « inférieurs » : culture à prairie non entretenue, etc.

Quant aux quatre variables descriptives, il s'agit de :

1. SAU, surface agricole utile de l'exploitation, avec pour modalités :

   <15ha : inférieure ou égale à 15 ha,
   15-30ha : de 15 à 30 ha,
   >30ha : plus de 30ha,
   expl=0 ; pas d'exploitation de la parcelle.

2. AGS, âge de l'exploitant et succession potentielle :

   20-35ans : de 20 à 35 ans,
   35-50ans : de 35 à 50 ans,
   >50/as : plus de 50 ans avec successeur,
   >50/ss : plus de 50 ans sans successeur,
   expl=0 : pas d'exploitant sur la parcelle.

3. REG, régime d'activité :

   !agri : uniquement agriculteur,
   da : double-actif,
   ret/nagr : retraité ou non-agriculteur,
   explo=0 : pas d'exploitant.

4. FVL, type de faire-valoir sur la parcelle :

   direct : la parcelle appartient à l'exploitant,
   bail : location avec bail,
   probverb : probablement location orale, achats d'herbe ou location gratuite,
   explo=0 : pas d'exploitant.

Comme on l'a vu ci-dessus, le problème a été ramené à l'extraction de règles reliant quatre variables décrivant une situation et une variable décision. SYNHESYS est donc potentiellement un outil approprié. Deux problèmes sont toutefois apparus à l'issue de cette étude préliminaire :

– les variables sélectionnées sont **qualitatives**, soit qualitatives **ordinales**, soit qualitatives **nominales**. Or SYNHESYS a été pour le moment uniquement

prévu pour le traitement de données booléennes (VRAI étant codé par 0, et FAUX par 1, par exemple) et réelles. Il faut donc essayer de se ramener à de telles données par un codage approprié, comme d'ailleurs en analyse de données [Bouroche et Saporta 1987] ;

- les variables sélectionnées ne permettent pas de discriminer les parcelles. Cela veut dire que pour un quadruplet de valeurs décrivant une parcelle, on pourra avoir plusieurs décisions différentes. Cela ne pose pas de problèmes aux géographes qui ont l'habitude de travailler avec de telles données, mais le comportement de SYNHESYS n'avait jamais été testé dans une telle situation.

Dans les sections suivantes, nous présentons les résultats de l'application de SYNHESYS à ce jeu de données.

Un premier codage

Une première solution au problème du codage consisterait à associer à chacune des modalités d'une variable donnée un entier distinct. Par exemple dans le cas de FVL, direct serait représenté par 1, bail par 2, probverb par 3, et expl=0 par 4. Plusieurs problèmes apparaissent toutefois. Tout d'abord, la distance euclidienne généralement utilisée dans SYNHESYS n'est pas bien adaptée. En effet, avec le codage ci-dessus, direct est implicitement plus proche de bail que de probverb. On a donc modifié la sémantique des données. Ce premier problème apparaît parce que FVL est qualitative nominale. La distance euclidienne donnerait des résultats plus corrects intuitivement dans le cas de variables qualitatives ordinales, comme AGS[1]. Mais il reste quand même un deuxième problème plus fondamental : les opérations algébriques qui seront faites après les calculs de distance dans le module connexionniste de SYNHESYS (modifications des prototypes, opérations sur les hyper-rectangles durant l'extraction de règles) n'ont **aucun sens**. Par exemple, dans le cas de AGS, la moyenne des entiers représentant >50/as et 35-50ans n'a aucune signification. Ce premier codage est donc inexploitable.

Plus généralement, ces premières remarques conduisent à poser le problème de la définition d'une distance adaptée au problème. Par exemple, en supposant que l'on soit capable de coder correctement les variables, il est peu probable que chaque variable ait la même importance. Il serait donc sans doute judicieux de pondérer chaque variable. Or c'est justement le genre d'information que les experts souhaiteraient obtenir d'un système informatique. De plus ils ne veulent surtout pas donner ce type d'information pour ne pas fausser les résultats par des considérations subjectives.

Un deuxième codage

Une deuxième solution de codage est couramment utilisée en analyse des données [Bouroche et Saporta 1987] : au lieu de représenter une variable par ses modalités,

1. Mais AGS pose en plus un problème particulier, car ses deux dernières modalités codent plus d'information que l'âge de l'agriculteur, et tiennent compte de la présence ou non d'un successeur susceptible de reprendre l'exploitation.

on la représente à l'aide de **variables indicatrices** supplémentaires. Par exemple, SAU sera représentée par quatre indicatrices pouvant prendre les valeurs 0 ou 1. On se ramène donc au traitement de variables booléennes. Il faut donc regarder comment SYNHESYS traite exactement ces variables (il est possible de déclarer que les variables sont continues ou booléennes dans la base de règles du module symbolique). Prenons un exemple. Soient deux variables booléennes x et y, une variable décision D de modalités $act1$, $act2$, $act3$, et faisons apprendre au système les exemples suivants :

x	y	D
0	1	$act1$
0	0	$act2$
1	0	$act3$
1	1	$act1$

On obtient alors les rectangles de la figure 4.4, puis on extrait les règles correspondantes (tab. 4.1). On voit donc que SYNHESYS traite les variables booléennes comme des variables continues pouvant varier entre 0 et 1. Le système ne tient donc pas réellement compte de la nature booléenne des données et l'utilisateur doit « interpréter » les règles obtenues pour revenir à des variables booléennes, ce qui n'est pas très pratique (et peut être éventuellement difficile) mais permet d'utiliser quand même le système pour faire quelques essais permettant de mieux cerner ses possibilités.

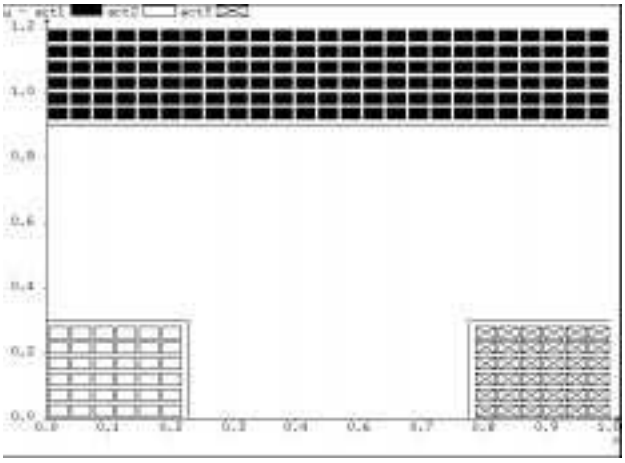


Fig. 4.4 - *Rectangles obtenus par SYNHESYS dans le cas de l'exemple de deux variables booléennes. Intuitivement, les résultats sont satisfaisants : un seul rectangle suffit en effet à regrouper les deux exemples de décision $act1$. Toutefois on voit que les variables booléennes sont traitées exactement comme des variables continues, ce qui semble avoir certains avantages : un exemple légèrement bruité, comme 0.1 0.1 par exemple, serait correctement classé par le système. Mais c'est lors de certaines extractions de règles que les variables booléennes poseront des problèmes (voir fin de la section 4.2.3).*

Nous avons donc codé nos données avec des variables indicatrices (quatre pour SAU, cinq pour AGS, quatre pour REG, quatre pour FVL, soit 17 variables d'entrée, et

```
REGLE R0
Si x in [0.8,1.0] et y in [0.0,0.2] Alors act3
FINREGLE R0
REGLE R1
Si x in [0.0,0.2] et y in [0.0,0.2] Alors act2
FINREGLE R1
REGLE R2
Si y in [0.8,1.0] Alors act1
FINREGLE R2
```

Tab. 4.1 - *Règles extraites par SYNHESYS dans le cas de variables booléennes. Le système traite les variables booléennes comme des données continues pouvant varier entre 0 et 1, ce qui laisse à l'utilisateur un travail d'interprétation : par exemple, il doit être capable de traduire* x *in [0.0,0.2] par « x est faux ». On voit aussi sur cet exemple l'intérêt du mécanisme d'élimination des variables non pertinentes : la variable* x *n'apparaît pas dans la règle R2.*

cinq pour la variable décision[2]).

Résultats du deuxième codage

Toutefois, avec ce codage, les premiers essais ont été décevants : après seulement une dizaine de présentations du fichier d'exemples, plusieurs centaines de prototypes sont obtenus, avec un pourcentage de bonnes réponses de moins de 50 %.

Nous avons alors supposé que cela provenait des exemples contradictoires présents dans le jeu d'exemples. Pour confirmer cette hypothèse, nous avons alors cherché à identifier le problème sur un cas simplifié. Nous avons pris les exemples du tableau 4.2, codés avec des variables indicatrices. Dans ce cas nous pouvons d'ailleurs facilement déterminer les règles qui pourraient être extraites. Avec cet exemple, nous obtenons des résultats intéressants (voir tableau 4.2) puisqu'il est possible de retrouver les règles extraites manuellement. Dans un deuxième temps, nous avons « perturbé » cet exemple simplifié, en introduisant des « clashs » [Cremilleux 1991], c'est-à-dire des exemples contradictoires. En ajoutant au fichier d'exemples initial les clashs (C D DEC2 et C E DEC1), on retrouve le même problème qu'avec le fichier du LAMA : après 300 présentations, on a déjà 172 hyper-rectangles[3] et un taux de bonnes réponses de seulement 62 %. Cela nous conduit donc à penser que les problèmes d'apprentissage rencontrés avec le fichier proviennent de la présence de clashs dans ce fichier, et nous avons confirmé cette hypothèse (voir annexe C).

Une extraction de règles

Comme nous l'expliquons en annexe C, nous avons construit, de manière arbitraire, un fichier de 51 exemples, sans clashs, à partir du fichier original de l'IGA, afin de

2. En fait les variables décisions sont toujours codées de cette manière dans SYNHESYS.
3. De plus, 552 hyper-rectangles avaient déjà été « tués ».

Variables représentées par leurs modalités			Mêmes variables représentées par des variables indicatrices		
X	Y	Z (décision)	$x_1x_2x_3$	y_1y_2	z_1z_2
A	D	DEC1	0 0 1	1 0	1 0
A	E	DEC1	0 0 1	0 1	1 0
B	D	DEC2	0 1 0	1 0	0 1
B	E	DEC2	0 1 0	0 1	0 1
C	D	DEC1	1 0 0	1 0	1 0
C	F	DEC2	1 0 0	0 1	0 1

Règles extraites manuellement :
R_1 : Si A alors DEC1
R_2 : Si B alors DEC2
R_3 : Si C & D alors DEC1
R_4 : Si C & E alors DEC2

```
Règles extraites par SYNHESYS :
RS0 :  Si x1 in [0.0,0.2] et x2 in [0.8,1.0]    Alors dec2
       et x3 in [0.0,0.2]
RS1 :  Si x1 in [0.8,1.0] et x2 in [0.0,0.2]    Alors dec2
       et x3 in [0.0,0.2] et y1 in [0.0,0.2]
       et y2 in [0.8,1.0]
RS2 :  Si x1 in [0.0,0.8] et x2 in [0.0,0.2]    Alors dec1
RS3 :  Si x2 in [0.0,0.2] et x3 in [0.0,0.2]    Alors dec1
       et y1 in [0.8,1.0] et y2 in [0.0,0.2]
```

Tab. 4.2 - *Un exemple simplifié présenté à SYNHESYS. Le fichier d'exemples est facilement appris après 300 présentations. Les règles extraites doivent être interprétées. Les prémisses de la règle RS0 peuvent être vues comme décrivant la présence de la deuxième modalité de la variable X, si bien que RS0 est en fait R_2. De même, RS1 est en fait R_4. Par contre RS2 et RS3 posent plus de problèmes car toutes les variables indicatrices n'apparaissent pas dans les prémisses. Dans un premier temps, RS2 peut être traduite en « Si x1 est faux et x2 est faux alors dec1 », ce qui revient à « Si x3 est vrai alors dec1 » à cause de la relation implicite entre les x_i. Finalement RS2 correspond à R_1. En suivant le même raisonnement, RS2 correspond à R_3. Cette démarche est toutefois empirique et peu satisfaisante.*

terminer cette étude par un essai concret d'extraction de règles. À partir du fichier de 51 exemples, très facilement appris (taux de bonnes réponses de 99 %, 24 prototypes), SYNHESYS a extrait 35 règles, dont les dix premières ont été reproduites en annexe C. Il faut alors essayer d'interpréter ces règles pour revenir aux modalités initiales, comme nous l'avons déjà fait ci-dessus sur de petits exemples. Toutefois de nouveaux problèmes apparaissent. Si à partir de l'expérience des petits exemples ci-dessus, on est à peu près sûr que « $x_i \in [0.0, 0.2]$ » peut être interprété en « x_i est faux » et que « $x_i \in [0.8, 1.0]$ » peut être interprété en « x_i est vrai », que dire de « $x_i \in [0.2, 1.0]$ » (règle R7 en annexe) ou de « $x_i \in [0.0, 0.8]$ » (règle R2) ? Il faudrait là une connaissance très précise de ce qui a conduit le système à fournir ces intervalles, et cela dépend à la fois du réseau de neurones et de la procédure d'extraction.

4.2.4 Conclusion et perspectives

On atteint là les limites de ce qu'on peut faire avec SYNHESYS V1 sur des données booléennes ou des données qualitatives nominales codées par variables indicatrices. De plus, le « bruit » présent dans les données fournies par les géographes est assez particulier, puisqu'il s'agit la plupart du temps de « clashs », c'est-à-dire d'exemples identiques mais appartenant à des classes différentes. Il s'agit en fait d'un problème de classification en des classes non-disjointes, alors que SYNHESYS ne peut traiter que des classes disjointes. Nous n'avons pas poursuivi plus loin l'interprétation de ces règles extraites, et nous pensons qu'il vaut mieux :

- soit utiliser des logiciels existants et adaptés à ce type de données,
- soit développer une nouvelle version de SYNHESYS permettant de résoudre les divers problèmes que nous avons soulevés, le principal étant la prise en compte d'autres types de données que les seules données quantitatives. Un problème théorique important se pose alors : les **généralisations** effectuées par les réseaux à base de prototypes sont **numériques** : ce sont des opérations algébriques effectuées sur des attributs numériques. Prendre en compte d'autres types d'attributs impose une réflexion de fond sur des généralisations non-numériques. Deux approches sont possibles : soit se ramener à des généralisations numériques par des transformations adaptées, à supposer qu'elles existent ; soit effectuer des généralisations en fonction du type de données (voir par exemple [Bisson 1993] dans le domaine de l'apprentissage automatique).

Le développement [4] de cette nouvelle version est en cours au LIFIA. Toutefois, travailler avec des utilisateurs, dans une optique de transfert de technologie, impose des contraintes et notamment celle de respecter des délais. Afin de faire progresser ce travail sur l'extraction de règles de décision, nous avons proposé d'utiliser des logiciels d'induction d'arbre de décision, comme ID3 [Quinlan 1986] et ses descendants.

4. Il faut à la fois modifier l'algorithme d'apprentissage et la méthode d'extraction de règles pour tenir compte du type de données. L'inconvénient est toutefois de compliquer encore plus le module connexionniste de SYNHESYS et finalement de sortir encore plus du cadre « neuronal » initial.

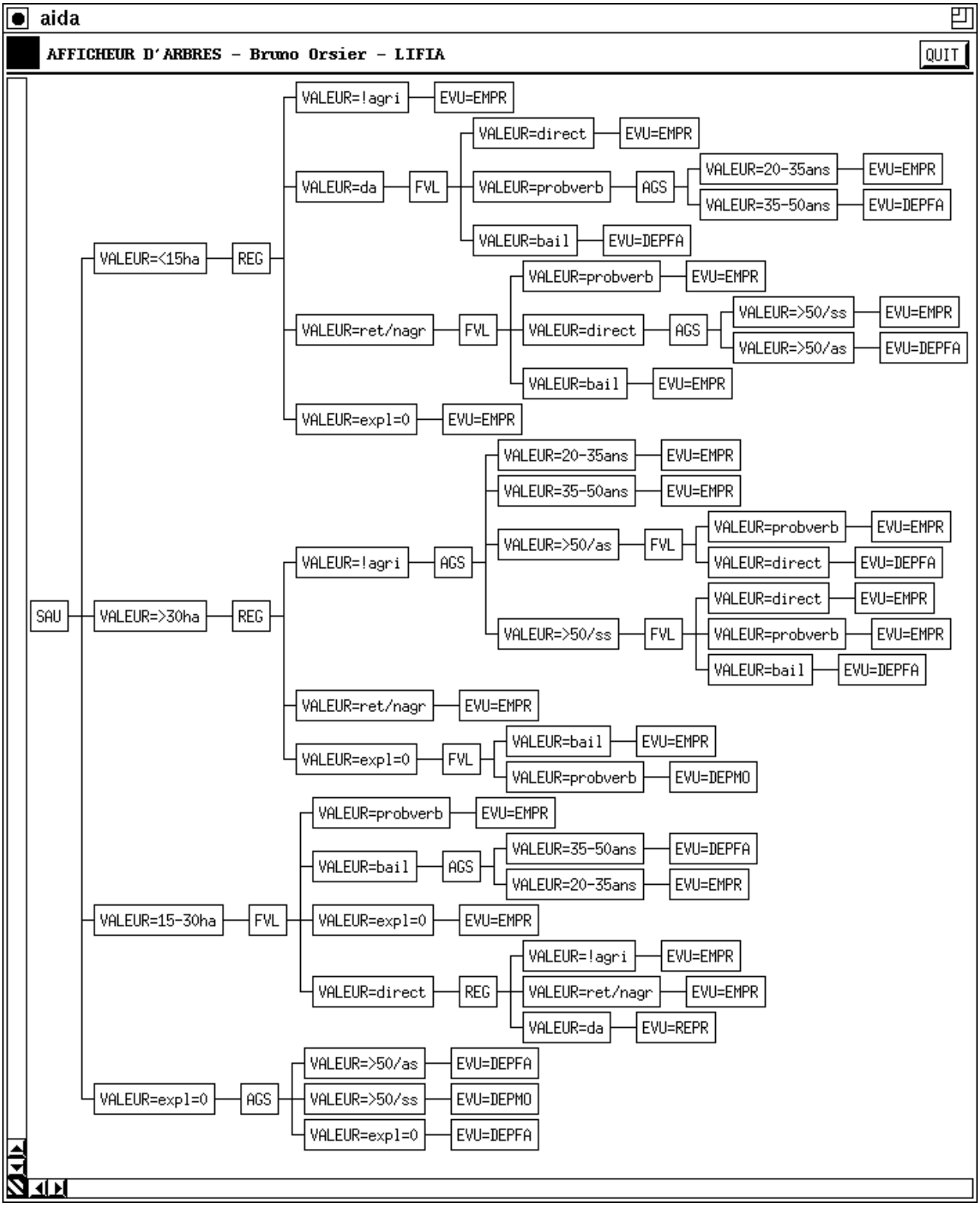


Fig. 4.5 - *Arbre de décision induit par ID3 à partir du fichier d'exemples sans bruit. En mettant cet arbre à plat, on pourrait en déduire une trentaine de règles d'ordre 0.*

Afin de montrer l'intérêt de tels algorithmes, nous avons programmé la version de base de ID3 pour l'appliquer au fichier de 51 exemples. Le résultat est donné par la figure 4.5, et il est très encourageant : l'arbre est assez facile à exploiter (mais nous ne prétendons pas qu'il ait une quelconque valeur du point de vue de l'application) et on peut facilement en déduire des règles d'ordre 0 si on le souhaite. Remarquons que le nombre de règles serait de 30 (nombre de feuilles de l'arbre), ce qui est du même ordre de grandeur que le nombre de règles extraites par SYNHESYS. Toutefois, la version de base de ID3 n'accepte pas de données bruitées, à moins d'utiliser des versions plus performantes. Nous avons choisi ARBRE [Cremilleux 1991], car il est bien adapté à l'induction dans le domaine de l'incertain (en particulier il prend en compte les « clashs »), il comporte des fonctions perfectionnées d'élagage des arbres de décisions, et il est disponible sur Macintosh, ce qui le rend facilement utilisable par des non-informaticiens. ARBRE a ensuite été utilisé au LAMA [Josselin 1995].

4.3 Bilan de SYNHESYS

Grâce à ce travail très concret sur des applications, nous sommes en mesure d'affiner le bilan de SYNHESYS. Du point de vue du concepteur de systèmes hybrides, le premier intérêt des problèmes réels est de permettre une certaine validation (section 4.3.1). Un deuxième intérêt est le recul que prend le concepteur au fil des contacts avec les utilisateurs ; ce recul permet de distinguer plusieurs questions plus fondamentales pour la conception de nouveaux systèmes hybrides (section 4.3.2). Nous concluons ce bilan par les apports de SYNHESYS (section 4.3.3).

4.3.1 Faisons le point sur la validation

Qu'entendons nous exactement par validation? Pour nous, au sens large, cela veut dire montrer :

- que le système fonctionne bien (il n'y a pas de « bugs », il se comporte selon ses spécifications),
- qu'il rend des services, par exemple en résolvant des problèmes non résolus jusqu'alors, ou bien encore en permettant de développer des applications suffisamment réalistes,
- que les méthodes heuristiques employées sont suffisamment performantes (une validation quantitative est indispensable). Ceci est d'autant plus important quand ces méthodes font intervenir des paramètres que l'utilisateur doit être capable de régler.

Valider SYNHESYS, aux sens ci-dessus, est indispensable pour tirer parti de tout le travail qui a déjà été effectué. Dans le chapitre précédent, nous avons déjà contribué à cette validation, en montrant des aspects positifs ou négatifs du système. La validation a été poursuivie dans ce chapitre, grâce aux applications, principalement celle réalisée en coopération avec le LAMA.

Nous résumons ici notre contribution. D'une part, nous avons apporté quelques modifications et corrections mineures : cet aspect de notre travail apparaît surtout en annexe A. D'autre part, **nous avons évalué le gain de performance** offert par le couplage direct entre les deux modules : il s'agit là d'une validation quantitative d'une méthode heuristique.

En ce qui concerne les aspects moins positifs de notre validation, nous avons donné des exemples de **non-convergence de l'algorithme d'apprentissage** dans le cas des réseaux à base d'hyper-rectangles : cette convergence[5] n'est donc pas prouvée (alors qu'elle l'est dans le cas des réseaux à base d'hyper-sphères parce qu'ils s'appuient sur la loi de Grossberg [Azcarraga 1993]). Ces exemples ont été trouvés grâce aux données réelles fournies par le LAMA et à nos expérimentations avec la base des iris (annexe A).

Nous avons également signalé que l'**incrémentalité** de l'apprentissage peut avoir des effets pervers dont nous avons donné un exemple concret. Cette incrémentalité peut donc être délicate à gérer, et il nous semble indispensable de développer des mécanismes appropriés. Ce point que nous avons soulevé sera d'ailleurs pris en compte dans un nouveau système hybride couplant CBR et réseaux à prototypes [Malek et Labbi 1995].

Enfin, nous étions particulièrement intéressés par l'étude de la **qualité**[6] **des règles extraites**, et c'était l'un de nos objectifs principaux lorsque nous avons recherché des applications concrètes et réalistes, ainsi que des experts de ces applications. En effet on ne sait pas actuellement si les règles extraites sont réellement exploitables par un expert. Les difficultés peuvent par exemple provenir par exemple du nombre important de règles extraites, ou encore du grand nombre de paramètres numériques qui interviennent dans les règles (les bornes des intervalles). Il ne nous est pas possible actuellement de conclure cette étude de qualité, puisqu'il faut maintenant attendre la réalisation de la nouvelle version de SYNHESYS.

La validation d'un système de taille importante comporte donc de nombreux aspects parmi lesquels nous avons fait un choix. Au moins trois autres points auraient mérité une étude approfondie :

le choix de la distance : tous les calculs de similitude reposent actuellement sur la distance euclidienne, éventuellement pondérée. Est-elle satisfaisante dans tous les cas ? Ne faut-il pas avoir une distance adaptée à chaque application ? Comment fait-on quand on a des données hétérogènes, ou quand des données sont manquantes ? On retombe en fait sur des problèmes classiques de l'analyse de données.

5. On parle ici de la stabilisation de l'architecture du réseau : stabilisation du nombre de prototypes et de leur répartition dans l'espace.

6. La notion de qualité peut être précisée grâce aux travaux de Perez, Hall, Romaniuk, et Lilkendey [1992] qui ont étudié spécialement ce problème, et se sont penchés plus particulièrement sur la précision des règles induites, la qualité des règles induites (jugée par deux experts et aussi par comparaison avec les règles d'un système expert existant), et la flexibilité du système, et cela pour des algorithmes symboliques (ID3, GID3, CN2), connexionniste (QuickProp) et hybride (SC-net). Leur conclusion est d'ailleurs qu'aucun système n'est parfait, et que dans tous les cas les règles obtenues doivent être révisées par des experts.

la « sensibilité » des règles extraites par rapport aux divers paramètres du système : les procédures d'extraction de règles reposent sur près d'une dizaine de paramètres numériques, il serait intéressant d'étudier l'influence de ces paramètres sur les règles extraites, pour voir si on obtient des règles très différentes ou non. Le choix de la distance peut aussi avoir une influence sur les règles extraites.

la comparaison avec d'autres systèmes : un SHNS a généralement pour ambition de résoudre certains des problèmes rencontrés par les systèmes symboliques. Par conséquent il est nécessaire d'évaluer précisément ses apports par rapport à des systèmes symboliques accomplissant le même travail. Dans le cas de SYNHESYS, il faudrait faire des comparaisons avec des systèmes symboliques d'induction et/ou de raffinement de règles, et éventuellement avec d'autres méthodes permettant d'améliorer les performances d'un système expert en chaînage arrière.

4.3.2 Des problèmes plus fondamentaux

Outre les problèmes relativement techniques mentionnés ci-dessus, notre étude nous a conduit également à examiner des problèmes que nous jugeons plus fondamentaux.

Les réseaux de SYNHESYS sont-ils des réseaux de neurones ?

Cette question mérite d'être posée, surtout au sujet du réseau à base d'hyper-rectangles. En effet, certaines des opérations portant sur les hyper-rectangles sont assez complexes. Par exemple durant l'absorption l'hyper-rectangle ne doit pas trop empiéter sur ses voisins[7], ou encore l'heuristique utilisée pour la différentiation se base sur les voisins de l'hyper-rectangle à différencier pour choisir quelle face reculer. De même l'assimilation suppose qu'on ne crée pas non plus un hyper-rectangle empiétant sur ses voisins. C'est aussi le cas du nouveau critère permettant de choisir entre différentiation et accommodation (et cela concerne les deux réseaux). Ces opérations que nous appellerons **globales** (parce qu'elles concernent tous les prototypes) sont destinées soit à interdire les intersections entre régions d'activations, soit à réduire le nombre de prototypes. Ces opérations sont faciles à programmer sur un réseau de neurones simulé, comme celui de SYNHESYS, sur un ordinateur séquentiel : elles consistent en de simples parcours de listes.

Toutefois, l'un des grands intérêts des réseaux est leur parallélisme potentiel : ils sont censés être facilement réalisables sur des machines parallèles. Or ces opérations globales compromettent cet avantage, parce qu'elles rendent ces réseaux difficiles à réaliser sous forme de circuits intégrés, où il faut se contenter d'unités très simples effectuant des sommations et des applications de fonctions de transfert. Finalement,

7. mais il n'y a pas de notion de voisinage dans l'algorithme ni dans l'architecture du réseau, si bien qu'il faut examiner TOUS les hyper-rectangles pour chaque opération relative à aux voisins d'un hyper-rectangle.

ces opérations globales compromettent l'adéquation algorithme/architecture qui est l'une des caractéristiques essentielles des réseaux de neurones.

Une réponse souvent faite à notre objection est « ce n'est pas grave, ma méthode sert juste à entraîner un réseau qui sera **ensuite** réalisé sur une machine parallèle pour être exploité sans apprentissage ». Mais ce type de réponse ne nous satisfait pas dans le cas de réseaux incrémentaux justement censés pouvoir apprendre durant toute leur utilisation.

Cette constatation sur les opérations globales nous conduit à nous interroger sur la définition que l'on doit retenir pour un réseau de neurones, et les caractéristiques minimales que l'on doit en exiger. En effet, les remarques ci-dessus ne nous conduisent pas à dire : « les réseaux de SYNHESYS ne sont pas des réseaux », mais plutôt à nous demander quelles sont les critères permettant de parler de réseaux de neurones.

Ce point est d'autant plus crucial que certains réseaux, notamment ceux à base de prototypes, sont très proches de méthodes statistiques ou symboliques (par exemple Salzberg [1991] décrit une méthode d'apprentissage utilisant des hyper-rectangles mais pas de neurones). Ainsi SYNHESYS peut être vu, dans une large mesure, comme un système combinant deux modules **symboliques**, le seul mécanisme vraiment neuronal étant la liaison directe entre les deux modules. Toutefois, ceci n'est pas incohérent avec la réflexion qui a conduit à la réalisation de SYNHESYS, car le but était la modélisation hybride de connaissances expertes. Dans ce contexte, peu importe qu'il y ait ou pas des réseaux de neurones dans le système[8].

Par contre, cela pose un problème pour nous car :

- nous souhaitons intégrer les bonnes caractéristiques des mondes symboliques et connexionnistes (c'est notre choix de recherche initial) ; pour cela il faut utiliser des systèmes clairement connexionnistes,
- notre travail sur des applications concrètes nous a amené à rencontrer des chercheurs d'autres disciplines, connaissant d'autres méthodes (statistiques, apprentissage automatique), et nous avons constaté que ces réseaux fort peu neuronaux sont difficilement crédibles.

Il nous semble donc indispensable de donner une définition assez stricte de ce que l'on accepte comme réseau de neurones. L'absence d'une telle définition a pour effet de laisser le champ libre à toutes les heuristiques et améliorations possibles, mais ne prenant pas en compte les contraintes que pose une éventuelle réalisation matérielle. Nous pensons que pour le moment le respect de l'adéquation algorithme/architecture est une définition suffisamment contraignante pour un réseau de neurones. Par conséquent, nous posons une première contrainte pour la conception de nouveaux systèmes

8. Cependant, ce cadre neuronal apporte à SYNHESYS plusieurs inconvénients :

   - il impose des contraintes de programmation importantes (il faut simuler neurones et connexions, ce qui est lourd et peu efficace),
   - il ne favorise pas l'évolution du système vers des mécanismes d'apprentissage plus puissants (prise en compte de contre-exemples, de données structurées, de distances mieux adaptées).

   De plus ce cadre n'est pas réellement respecté dans SYNHESYS (les opérations sur les prototypes sont globales).

hybrides :

Contrainte 4.1 *utiliser des réseaux de neurones respectant l'adéquation algorithme architecture.*

Le choix de l'architecture

Dans SYNHESYS, chacun des deux modules effectue exactement le même travail : classer le même vecteur d'entrée. Ils utilisent pour cela des connaissances différentes et complémentaires, mais de même niveau car portant sur les mêmes objets. Ce choix de base dans la conception du système a donc conduit à l'architecture de la figure 3.1. Ce choix architectural a toutefois une forte influence sur la classe d'applications que peut traiter le système hybride : la classification de vecteurs de caractéristiques numériques et booléennes. D'autre part, il conduit, dans une certaine mesure, les deux modules à se limiter l'un l'autre. Si SYNHESYS profite des points forts de ses deux modules, il souffre tout autant de certaines de leurs faiblesses respectives :

- le système a un faible pouvoir de représentation de connaissances, ce qui est dû à l'utilisation de règles d'ordre 0^+ dans le module symbolique. Cela a notamment conduit à un gros travail de traduction des règles d'ordre 1 de NEUROP pour réaliser SHADE, comme nous l'avons signalé plus haut. La restriction du système à ces règles est due en partie à la volonté de pouvoir mettre en correspondance les règles de l'expert et les règles extraites à partir du réseau : le réseau et la procédure d'extraction exercent donc une forte contrainte sur le module symbolique ;
- à cause du réseau, le système n'accepte que des entrées sous la forme de vecteurs de caractéristiques numériques et booléennes. Ceci a également posé des problèmes dans le cas de SHADE, pour la prise en compte de certains fichiers de description de patients (lorsque ces fichiers étaient de taille variable), et dans le cas de l'application en géographie alpine ;
- le réseau est extrêmement **spécialisé** sur un type de situation donnée. Cela pose problème quand on doit enchaîner plusieurs raisonnements portant sur des situations de type différent : il faut alors gérer plusieurs réseaux différents, ce qui complique l'architecture initiale. Ce problème est également apparu dans SHADE ainsi que dans l'application CNET ;
- le système symbolique impose sa fragilité dans les schémas d'interactions utilisant la validation en chaînage arrière.

SYNHESYS n'évite donc pas complètement le risque que courent les systèmes hybrides : l'héritage de points faibles des composants (chaque module imposant certaines de ses moins bonnes caractéristiques au système tout entier).

Un tel choix architectural peut se justifier dans certaines applications, mais a des conséquences très importantes. En particulier, il n'est pas nécessairement le meilleur moyen d'exploiter les points forts des deux composants. Avant de concevoir un nouveau système hybride, il faudra donc réfléchir à la répartition des tâches entre les composants du système et décider si les connaissances qu'ils manipulent doivent

être du même niveau ou pas. D'où la nouvelle contrainte pour la conception de systèmes hybrides :

Contrainte 4.2 *définir une architecture évitant l'héritage des points faibles.*

Importance de l'extraction de règles

L'extraction de règles de décision est le mécanisme central de SYNHESYS. Toutefois elle contraint les deux modules à manipuler des connaissances très voisines, sans quoi elle ne serait pas possible. Cette forte contrainte empêche notamment les modules d'évoluer indépendamment : ainsi on ne peut pas remplacer le module symbolique de SYNHESYS par un module plus puissant (règles d'ordre 1, objets) sans revoir toute la procédure d'extraction. De plus il reste très difficile d'extraire d'un réseau de neurones autre chose que des règles d'ordre 0 ; des travaux actuels visent à extraire diverses structures symboliques [Crucianu 1994], mais sont pour le moment peu utiles pour résoudre des problèmes pratiques.

L'extraction de règles à partir d'un réseau de neurones reste de toute manière délicate et plus coûteuse que d'autres méthodes plus simples. À cet égard, la comparaison entre NEITHER [Baffes et Mooney 1993] et KBANN [Towell 1992] est très intéressante. KBANN transforme une base de règles en réseau multicouches, puis entraîne ce réseau à l'aide de la rétropropagation, et enfin extrait de nouvelles règles à partir de ce réseau. Il s'agit en fait uniquement de raffinement de règles car contrairement à SYNHESYS on doit absolument partir d'une base de règles initiale. Par contre NEITHER est un système de raffinement de règles purement symbolique. Les deux systèmes utilisent le même type de règles, et ont été testés sur le même problème [Baffes et Mooney 1993]. Les résultats, mesurés par le pourcentage de bonnes classifications sur le jeu d'essais, sont quasiment identiques pour les deux systèmes, mais le système symbolique est beaucoup plus direct et beaucoup plus rapide, et produit des règles moins complexes.

Bien sûr, l'extraction de règles n'est pas toujours une fin en soi, et il s'agit souvent de mieux comprendre (d'expliquer) le fonctionnement d'un réseau. Mais d'autres formes d'explication sont possibles, et peut être plus adaptées : en effet les réseaux de neurones intéressent maintenant sérieusement les mathématiciens, et l'on peut espérer dans un proche avenir des résultats intéressants sur le comportement des réseaux. Les caractérisations mathématiques seront probablement plus adaptées dans le cas des grands réseaux de neurones, dont la réalisation est l'une des voies de recherche les plus intéressantes pour l'avenir.

Enfin, centrer un SHNS sur l'extraction de règles a l'inconvénient important de rendre sa validation (et par conséquent sa valorisation) complexe, puisqu'il est nécessaire de le comparer avec les nombreux autres systèmes d'extraction (ou induction) de règles et d'arbres de décision.

Par conséquent, nous imposons la contrainte suivante pour la conception de SHNS :

Contrainte 4.3 *proposer des alternatives à l'extraction de règles.*

Une version plus faible de cette contrainte est :

Contrainte 4.4 *proposer des architectures où l'extraction de règles n'est pas indispensable.*

Nous chercherons surtout à respecter cette contrainte faible, car le respect de la contrainte forte est certainement un sujet de recherches à part entière.

4.3.3 Originalités et apports de SYNHESYS

Le premier mérite de SYNHESYS est d'exister sous la forme d'un système informatique réellement utilisable (il dispose même d'une interface graphique très commode) ; on a vu dans le chapitre 2 que c'est loin d'être le cas de tous les systèmes hybrides. Il a d'autre part été conçu relativement indépendamment d'une application particulière, ce qui permet de l'appliquer à divers problèmes. De plus un apport important de SYNHESYS est de soulever ces problèmes fondamentaux mentionnés plus haut, et dont l'étude doit précéder toute nouvelle conception de système hybride.

De plus même si l'architecture choisie impose des contraintes importantes, elle permet de résoudre au moins partiellement certains des problèmes des composants symboliques et connexionnistes (modulo la remarque ci-dessus sur le caractère peu neuronal des réseaux employés) :

amélioration des performances avec l'expérience : elle est rendue possible grâce à la liaison neuronale directe ;

adaptation : grâce à l'extraction de règles le module symbolique peut être adapté et amélioré ;

dégradation progressive : les réseaux de SYNHESYS peuvent donner des bonnes réponses pour des exemples non appris ;

explication du fonctionnement du réseau : grâce à l'extraction de règles on peut obtenir une description symbolique de ce fonctionnement.

4.3.4 Conclusion et perspectives

Nous avons examiné SYNHESYS d'un point de vue critique qui nous parait indispensable pour bien valoriser un tel système, notamment par des publications permettant de montrer l'importance et la qualité de l'ensemble du travail effectué. Les principaux éléments de notre bilan apparaissent dans [Orsier, Amy, Rialle, et Giacometti 1994], mais il est souhaitable de poursuivre ce travail de valorisation.

Il est clair qu'il reste des problèmes à résoudre, notamment la validation qui représente sans doute autant de travail que la réalisation du système. Mais cette analyse critique ne doit pas faire oublier les nombreux apports et originalités du système. En particulier, la méthode d'extraction/raffinement de règles basée sur des hyperrectangles n'a pas d'équivalent dans la littérature, à notre connaissance. D'autre

part, la réalisation d'un couplage fort est également originale, car il existe très peu d'exemples de ce type de couplage (chapitre 2).

La réalisation de ce système montre aussi qu'il est possible d'offrir de nombreux mécanismes complémentaires (compilation, raffinement, extraction de règles ; diagnostic ; amélioration des performances) au sein d'un même système et sur la base d'une architecture très cohérente. C'est l'un des apports principaux des systèmes hybrides. Là encore SYNHESYS se caractérise par le nombre et la variété de mécanismes qu'il propose. Le chapitre 2 permet de voir que SYNHESYS est l'un des systèmes les plus complets à l'heure actuelle car la plupart des autres systèmes servent à étudier un seul problème très précis (comme l'extraction de règles par exemple).

En conclusion, tous ces mécanismes ont pu être développés sans inventer de nouveau paradigme de représentation de connaissances, il a suffi de faire cohabiter et coopérer deux paradigmes existants.

En ce qui concerne les perspectives pour SYNHESYS, plusieurs directions de recherches apparaissent à l'issue de notre étude :

- réaliser une nouvelle version de SYNHESYS adaptée à une plus large gamme de problèmes. Dans ce cas le caractère neuronal du module qui traite les exemples continue à n'avoir que peu d'importance. Dans ce cas, il serait intéressant de continuer à améliorer l'algorithme d'apprentissage et la procédure d'extraction de règles, sans se restreindre à un cadre neuronal peut-être limité[9] ;
- poursuivre la validation du système actuel,
- essayer de simplifier le réseau à base d'hyper-rectangles, en utilisant des mécanismes plus simples, moins globaux, de manière à se replacer dans un cadre neuronal.

Nous abordons un nouveau problème dans le chapitre suivant, la conception d'un SHNS multi-niveaux, afin d'ouvrir de nouvelles perspectives pour les SHNS.

9. Se placer dans un autre cadre comme l'apprentissage automatique ou le RBC permettrait de disposer de plus de mécanismes, comme l'apprentissage à partir de contre-exemples, peu étudié dans le cadre neuronal.

Chapitre 5

Conception d'un SHNS multi-niveaux : NESSY3L

Dans ce chapitre nous proposons une architecture, appelée NESSY3L (*NEuroSymbolic SYstem with 3 Levels*), respectant, dans une certaine mesure, les contraintes déterminées dans le chapitre précédent, et offrant de nouvelles perspectives et directions de recherche.

L'idée de base consiste à intercaler un niveau neurosymbolique entre un niveau purement neuronal et un niveau symbolique (section 5.1). Nous présentons ensuite l'adaptation de cette architecture à un problème précis, l'évitement d'obstacles par un robot mobile (section 5.2). Puis nous décrivons de manière précise la réalisation de cette architecture et son intégration dans le simulateur de robot MOLUSC du LIFIA (section 5.3), ainsi que plusieurs expériences et améliorations (section 5.4). Enfin nous donnons des éléments de discussion de cette architecture (section 5.5).

5.1 Architecture proposée

5.1.1 Motivations pour NESSY3L

Revenons un moment sur un aspect injustement négligé des SHNS : l'interface entre les modules symboliques et neuronaux. Cet aspect est un peu négligé en raison du caractère ad-hoc des systèmes déjà réalisés : les concepteurs programment une interface au moment où c'est nécessaire, pour un usage très précis. Seuls les environnements de développement de SHNS conduisent à une réflexion plus générale sur les interfaces :

- dans [Wilson et Hendler 1993], le problème de la réalisation de cette interface est essentiellement vu comme un problème de traduction entre les mondes symboliques et connexionnistes. La proposition de ces auteurs est d'utiliser la programmation par objets, parce qu'elle fournit un cadre bien adapté au développement de méthodes de traduction génériques et spécialisées ;

- dans le projet MIX, nous avons proposé que cet aspect d'interface/traduction soit explicitement pris en compte au niveau du langage de développement multi-agents ADL. L'intérêt d'une prise en compte explicite est justement d'éviter les réalisations ad-hoc et de contraindre le développeur à réfléchir sur l'interface. Suite à cette proposition, un sous-ensemble du langage CKRL[1] a été retenu et a été intégré dans ADL [González, Velasco, et al. 1994].

Dans les SHNS que nous avons étudiés, l'interface peut être de deux types :

- traduction des sorties d'un module en entrées pour un autre module : par exemple, les activations des unités de sortie d'un réseau sont converties en faits symboliques ;
- traduction des connaissances représentées dans un module en connaissances exploitables dans un autre module : par exemple, l'extraction de règles symboliques à partir d'un réseau.

Nous avons vu dans les chapitres précédents les inconvénients importants de l'extraction de règles, procédure fort lourde par rapport aux avantages que l'on peut en retirer actuellement. Aussi, nous allons explorer dans ce chapitre une nouvelle architecture qui rendra moins nécessaire ce type d'interface.

Une seconde motivation, à plus long terme, est de concevoir une architecture qui puisse servir de support à la réalisation de couplages forts. Le couplage fort, comme nous l'avons indiqué dans le chapitre 2, est potentiellement très puissant mais il est très peu utilisé pour le moment, en raison de la difficulté de conception et de réalisation de systèmes fortement intégrés. Nous montrerons que notre architecture peut servir de support à de futurs couplages forts.

5.1.2 Principe général de cette architecture

Le principe de base est très simple, et est présenté par la figure 5.1. Contrairement au schéma traditionnel des SHNS de la figure 5.2, nous proposons d'insérer entre les niveaux connexionnistes et symboliques un (ou plusieurs éventuellement) niveau(x) intermédiaire(s).

Les principaux avantages que nous attendons de cette architecture sont :

- la simplicité des interfaces : en effet, au lieu d'avoir une interface complexe, on a plusieurs interfaces plus simples ;
- le relâchement des contraintes imposées par le schéma traditionnel : par exemple dans SYNHESYS l'extraction de règles impose aux modules de traiter des connaissances très proches ;
- une meilleure répartition des tâches : dans une telle architecture, on utilise mieux les points forts des composants connexionnistes et symboliques ;

1. *Common Knowledge Representation Language*. CKRL a été conçu dans le cadre d'un autre projet Esprit (2154).

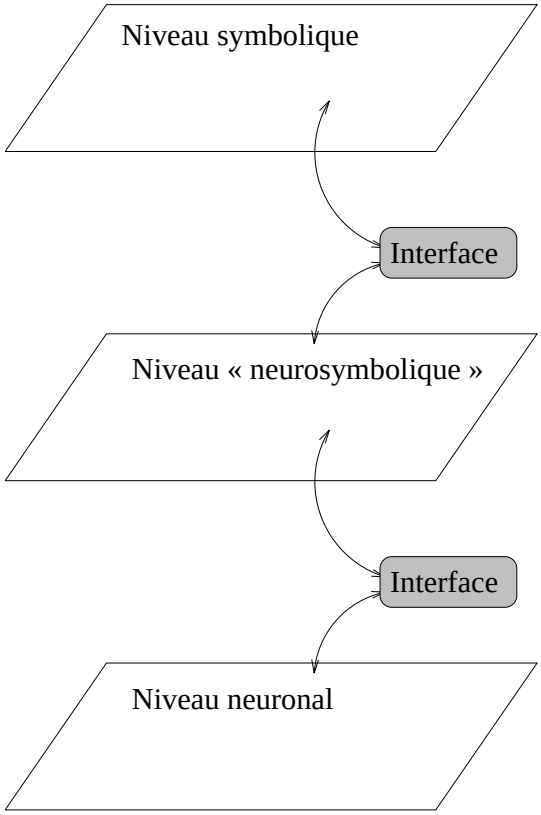


Fig. 5.1 - *Architecture NESSY3L proposée dans ce chapitre. L'interface entre les niveaux symboliques et neuronaux est décomposée en plusieurs interfaces simples grâce à la présence d'un niveau intermédiaire.*

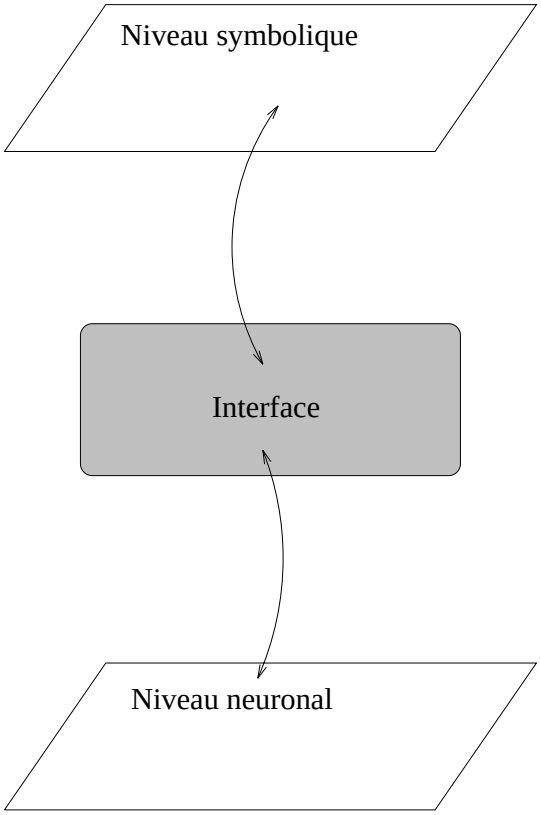


Fig. 5.2 - *Architecture traditionnelle d'un SHNS. L'interface peut être très complexe.*

– le respect des trois contraintes que nous avons énoncées dans le chapitre précédent :

1. utiliser des réseaux de neurones respectant l'adéquation algorithme architecture ;
2. éviter l'héritage des points faibles ;
3. proposer des architectures où l'extraction de règles n'est pas indispensable.

Bien que cette proposition soit surtout motivée par des considérations relevant de l'informatique, il est intéressant de situer une telle architecture par rapport aux travaux en Sciences Cognitives menés notamment par l'équipe Réseaux d'Automates du LIFIA. Notre architecture s'insère en fait très bien dans la hiérarchie des différents niveaux des activités cognitives proposée par Amy [1991] (fig. 5.3).

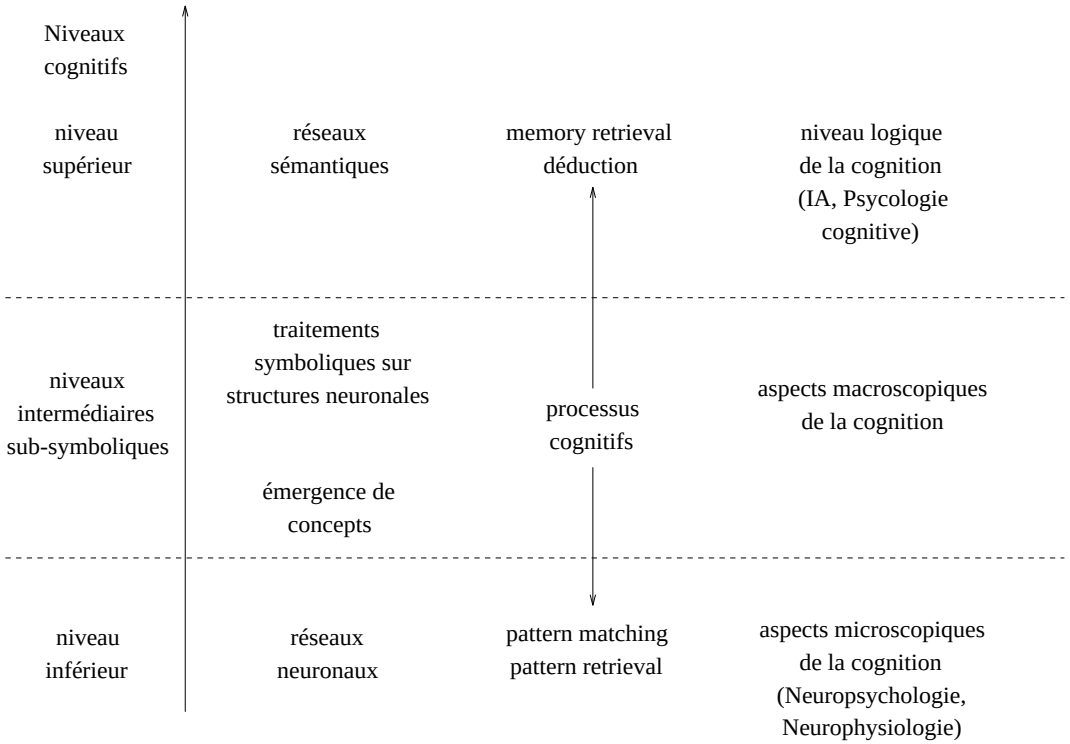


Fig. 5.3 - *D'après Amy [1991]. Les différents niveaux des activités cognitives. En face de chaque niveau sont indiqués les types des principales tâches relevant de ce niveau.*

Le système SYNHESYS était surtout concerné par les niveaux inférieur (traitements de type associatif – utilisation de réseaux de neurones) et supérieur (raisonnement logique – utilisation de systèmes symboliques). L'architecture que nous proposons permettra de réfléchir sur les niveaux intermédiaires qui sont difficiles à concevoir.

Bien entendu, il reste à définir un niveau intermédiaire. Nous nous intéressons surtout à ce que B. Amy appelle « traitements symboliques sur structures neuronales » (fig. 5.3) ; l'« émergence de concepts » est pour le moment laissée de côté. Or justement, nous avons fait remarquer dans le chapitre 2 la grande variété et l'intérêt des

travaux relatifs à l'approche purement connexionniste de l'intégration neurosymbolique. Nous pensons que ces travaux peuvent être utilisés pour réaliser un tel niveau intermédiaire.

Toutefois, il n'est pas possible à ce stade de nos réflexions de privilégier tel ou tel travail, en particulier parce que l'application visée[2] aura certainement une grande importance. Mais nous pensons surtout aux nombreux travaux qui visent à réaliser une base de règles sous forme d'un réseau de neurones [Sun 1991, Towell 1992, Labbi 1993, Fu 1994] ; dans l'application que nous proposons ci-dessous, nous utiliserons un réseau versatile [Labbi 1993] pour réaliser une base de règles. Les travaux qui visent à représenter des structures de données récursives et de taille variable (tableaux, listes, arbres, etc.) dans des systèmes connexionnistes [Pollack 1990] présentent également un intérêt, peut-être pour réaliser les niveaux intermédiaires supérieurs ; toutefois nous n'avons pas exploré cette direction de recherche.

5.2 Adaptation de NESSY3L à une application appropriée

5.2.1 Motivations pour la robotique mobile

Le travail que nous avons effectué dans les chapitres précédents nous conduit à prendre en compte plusieurs éléments pour choisir une application appropriée.

Parmi les diverses argumentations en faveur des systèmes hybrides, nous trouvons particulièrement intéressante celle de Tirri [1991]. Rappelons que cet auteur propose d'utiliser des réseaux de neurones pour prendre en compte des données sensorielles continues, et ainsi d'apporter une solution aux difficultés que rencontrent les systèmes symboliques faces à de telles données. Dans ce contexte on peut utiliser par exemple les capacités d'approximation des réseaux multicouches et les capacités d'auto-organisation des réseaux de Kohonen ou des réseaux à base de prototypes.

Ce premier élément nous a conduit à envisager la robotique mobile [Reignier 1994b] comme domaine d'application pour notre système hybride, en raison du grand nombre de données numériques à traiter. De plus, ces données provenant de capteurs sont généralement bruitées, et là aussi les réseaux de neurones trouvent un emploi naturel.

D'autre part, les possibilités de réalisation de réseaux de neurones sous forme de circuits intégrés vont se multiplier dans les prochaines années, et la robotique mobile sera un champ d'application privilégié car il est nécessaire d'embarquer les systèmes de traitement dans un espace réduit. Il est donc important pour les systèmes hybrides d'être présents dans ce domaine.

2. Rappelons que notre but est de résoudre des problèmes concrets, pas de proposer un modèle théorique de la cognition. Le lecteur intéressé par la notion d'organisation des niveaux cognitifs pourra se reporter à [Grumbach 1994].

De plus, l'une des difficultés importantes que nous avons rencontrées dans les applications précédentes (avec le CNET et le LAMA) était le manque d'expérience des utilisateurs par rapport aux systèmes hybrides. En effet travailler avec des utilisateurs ne connaissant pas les réseaux de neurones, voire pas les systèmes symboliques non plus, demande un temps important de formation, de définition du vocabulaire, de correction des a priori. Dans le même ordre d'idées, les systèmes hybrides sont difficiles à justifier quand les limites de leurs composants n'ont pas été vérifiées directement dans le domaine de l'application. Or ces difficultés tombent d'elles-mêmes en robotique, car les roboticiens sont familiarisés avec de nombreuses techniques (voir par exemple l'état de l'art très complet de Reignier [1994b]) et même avec les SHNS [Pomerleau, Gowdy, et Thorpe 1991].

Enfin, de manière générale, les roboticiens sont très intéressés par les couplages numérique/symbolique, et les SHNS peuvent apporter des solutions originales.

5.2.2 Choix d'un problème particulier

La robotique mobile est toutefois un domaine très vaste, dans lequel il faut résoudre des problèmes de haut niveau comme la planification de chemins, et des problèmes de plus bas niveau comme le traitement de données perceptives. Étant donné que nous voulons exploiter les capacités des réseaux de neurones à traiter un grand nombre de données numériques, nous avons choisi de nous focaliser sur les problèmes de bas niveau où l'on traite les données issues de capteurs. Un problème type est alors l'**évitement d'obstacles**, et nous allons traiter ce problème précis dans le reste de ce chapitre.

Du point de vue de la robotique, le cadre de travail est alors la **navigation réactive**. Schématiquement, il s'agit de réaliser une application $f : S \longrightarrow A$, où S est l'**espace perceptif** (espace de toutes les mesures de capteurs possibles), et A est l'**espace des actions** (espace de toutes les commandes possibles).

Toutes les expériences mentionnées dans le reste du chapitre ont été effectuées avec le simulateur de robot MOLUSC [Reignier 1994b], développé au LIFIA. Il est certain que travailler avec des données simulées comporte des limites [Dedieu et Bessière 1994], mais cela permet de faire facilement de nombreuses expériences, le temps de mettre au point un premier prototype sans danger : le robot réel simulé par MOLUSC pèse en effet 250 kg, et son utilisation demande certaines précautions.

Concrètement, nous disposons des mesures provenant de 24 capteurs ultra-sons, répartis tout autour du robot (fig. 5.4), qui donnent une estimation des distances entre le robot et les obstacles par mesure du temps de vol. Une caractéristique importante des mesures par ultra-sons est leur relative mauvaise qualité ; en effet, de nombreux phénomènes peuvent perturber les mesures. Entre autres, la vitesse du son dans l'air dépend de l'humidité et de la température, et la réflexion des ondes ultra-sonores varie selon le matériau de l'obstacle. Cette mauvaise qualité est prise en compte par MOLUSC qui bruite artificiellement les mesures qu'il fournit. MOLUSC est donc un peu plus qu'un simple simulateur, mais sa modélisation du

bruit ne correspond toutefois pas à la réalité. Aussi des expériences sur le robot réel seront nécessaires pour valider complètement notre travail.

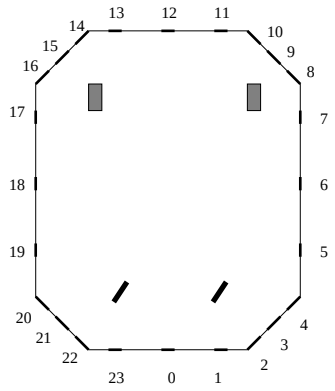


Fig. 5.4 - *Répartition des 24 capteurs ultrasons autour du robot.*

En ce qui concerne les commandes, nous devons générer des couples (vl, va) où vl est la vitesse linéaire désirée, et va est la vitesse angulaire désirée.

Enfin, pour réaliser un système symbolique, il faut disposer de connaissances expertes. Nous allons pour cela réutiliser une partie du travail effectué par Reignier [1994b], qui a utilisé la logique floue et différentes méthodes pour traiter le problème de la navigation réactive. Il sera d'ailleurs intéressant de comparer notre architecture avec ces différentes approches.

5.2.3 Adaptation de l'architecture

Notre architecture générale doit maintenant être adaptée au problème que nous voulons résoudre. Plusieurs points doivent être précisés :

- proposition d'un niveau intermédiaire,
- répartition des tâches entre les niveaux,
- définition des interfaces intermédiaires,
- utilisation des données capteurs.

En ce qui concerne la répartition des tâches, nous avons indiqué ci-dessus que l'un des intérêts des réseaux était de pouvoir traiter les données sensorielles ; par conséquent, le rôle du niveau neuronal sera d'effectuer de tels traitements que nous préciserons ci-dessous (il s'agit principalement de la reconnaissance de situations perceptives). Le niveau symbolique peut avoir plusieurs rôles :

- effectuer diverses tâches de haut niveau, comme la planification du chemin à suivre, la prise en compte de règles du type code de la route (voir par exemple [Hassoun 1994]). Un niveau symbolique réaliste pourrait être réalisé[3] à partir

3. Un tel travail pourrait être effectué dans le cadre d'un sujet de DEA, voire de thèse, via une collaboration entre l'équipe « Réseaux d'automates » et les roboticiens du LIFIA.

des travaux effectués par les roboticiens du LIFIA [Hassoun 1994, Reignier 1994b] ;

- résoudre les éventuelles difficultés rencontrées par les niveaux inférieurs, dans le cadre d'une démarche de développement incrémentale :

  1. réaliser le niveau n ;
  2. déterminer, via des expériences et/ou une analyse théorique, les limites de l'architecture composée des niveaux $1, ..., n$;
  3. réaliser un niveau $n+1$ permettant de repousser ces limites.

  Bien qu'un peu empirique, cette démarche a l'intérêt de proposer une solution pragmatique au problème de la répartition des tâches entre composants d'un SHNS.

Comme niveau intermédiaire, nous proposons d'utiliser les réseaux non-isolés que nous avons présentés dans le chapitre 1. Ces réseaux, que nous qualifierons de versatiles dans le reste du chapitre, sont particulièrement bien adaptés :

- ils ont une faible période d'autonomie[4], et sont donc bien adaptés au traitement de données évolutives comme les mesures effectuées par les capteurs [Labbi 1993] ;
- ils sont un modèle théorique du comportement de l'expert qui modifie son diagnostic en temps réel, à la suite de changements dans son contexte de décision (par exemple des modifications dans les données) [Labbi 1993]. Par conséquent ils sont particulièrement intéressants pour la robotique mobile, où les décisions peuvent être remises en cause à tout instant (apparition d'un obstacle mobile par exemple) ;
- ils peuvent coder des règles de décision [Labbi 1993], et peuvent donc bien constituer le niveau intermédiaire que nous proposons ci-dessus ;
- ils sont parfaitement bien caractérisés d'un point de vue mathématique, et aucune autre explication de leur comportement n'est a priori nécessaire ;
- ils offrent des possibilités originales de couplage avec le niveau symbolique : par exemple le niveau symbolique pourrait aussi imposer des changements de contexte, ou encore régler divers paramètres des réseaux versatiles (coefficient de fatigue de telle ou telle unité, poids d'une connexion, etc.).

Ce niveau intermédiaire permet de produire les décisions (vl, va) qui seront envoyées aux effecteurs (fig. 5.5).

5.3 Réalisation de NESSY3L/ROB

Nous appelons NESSY3L/ROB l'application de NESSY3L à la robotique.

4. La période d'autonomie d'un tel système est le temps durant lequel le système ne prend pas en compte ses entrées (pour mettre à jour son état interne par exemple).

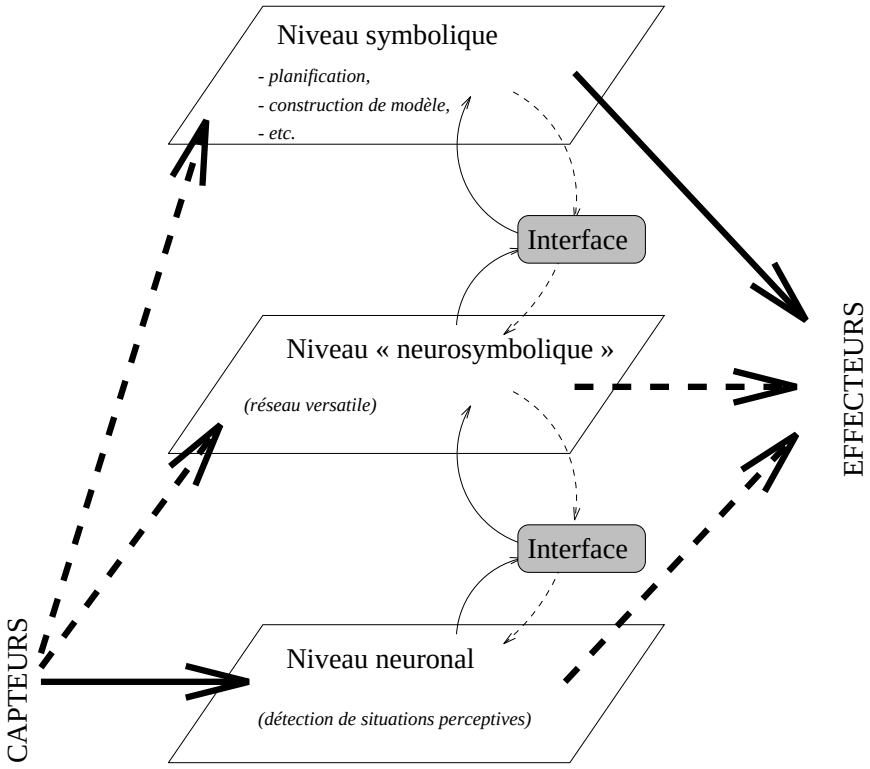


Fig. 5.5 - *Adaptation de notre architecture générale au problème de la navigation réactive. Les flèches en pointillés indiquent des transferts potentiels d'informations, tandis que les flèches en trait plein indiquent les transferts qui existent à l'heure actuelle dans notre système.*

Nous décrivons dans cette partie la réalisation informatique des deux niveaux inférieurs de NESSY3L/ROB, ainsi qu'un niveau symbolique à l'état embryonnaire. Notre prototype a été greffé sur le simulateur MOLUSC en développant des outils de communication appropriés.

Le détail de NESSY3L/ROB est présenté par la figure 5.6.

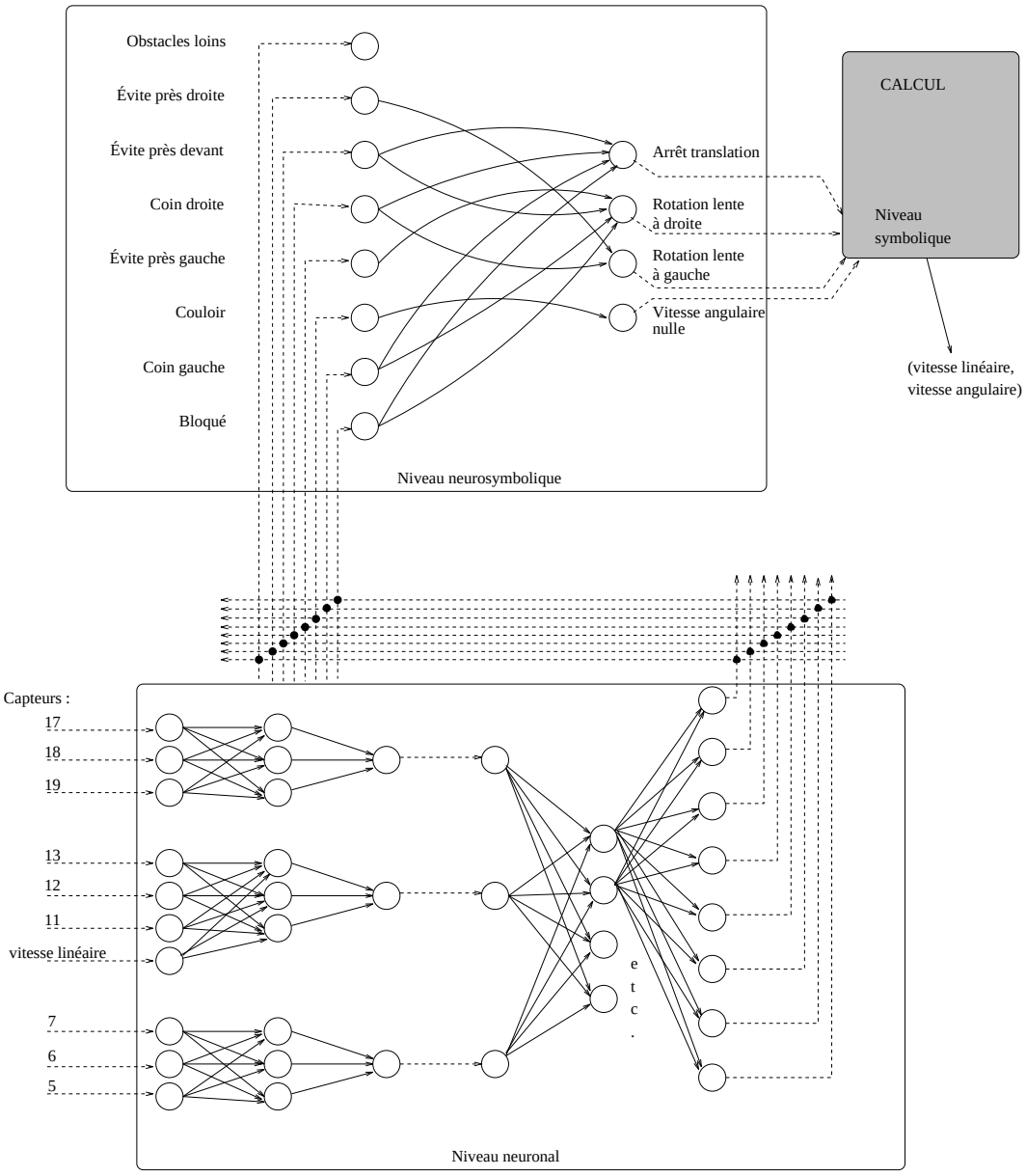


Fig. 5.6 - *Détail de notre système NESSY3L/ROB. La figure montre notamment les trois niveaux : neuronal pur, neurosymbolique, symbolique pur. Sur la figure, etc. signifie que la couche cachée et la couche de sortie du réseau 3x4x8 sont complètement interconnectées.*

Nous utilisons ici deux aspects du travail de Reignier [1994b] : d'une part, nous allons chercher à détecter les huit situations de la figure 5.7, situations que P. Reignier a utilisées comme entrées de son système flou ; d'autre part, nous allons chercher à

représenter sous forme d'un réseau versatile les règles d'évitement d'obstacles que P. Reignier a associé à ces huit situations. Il s'agit d'une sous-utilisation de notre architecture[5], mais il est nécessaire dans un premier temps de réaliser un prototype simple, afin d'affiner nos idées et de mettre au point quelques outils.

À l'exception du réseau versatile, tous les réseaux décrits ci-dessous ont été réalisés avec le simulateur SNNS [Zell et al. 1993]. Ce simulateur se présente sous trois formes principales :

- un logiciel graphique de création, visualisation, entraînement et test d'une grande variété de réseaux,
- un logiciel permettant d'effectuer toutes les opérations d'entraînement et de test en différé. Ceci est très utile car il est courant de consommer plusieurs heures de temps CPU sur une station de travail pour entraîner un réseau,
- un ensemble de fonctions (en langage C) utilisables par un programmeur qui désirerait intégrer certaines de ces fonctionnalités dans un autre logiciel.

L'ensemble de fonctions n'est pas encore documenté dans SNNS. Aussi, l'exemple d'utilisation que nous avons développé, pour intégrer les réseaux de SNNS dans MOLUSC, constitue une base à partir de laquelle d'autres programmeurs exploitent ces fonctions.

5.3.1 Détecteurs de huit situations perceptives

Nous désirons détecter les huit situations de la figure 5.7. En les observant, on voit qu'il suffit d'être capable de détecter les obstacles sur un côté et sur le devant du robot. Dans cette section, nous décrivons tout d'abord la manière dont nous avons réalisé deux réseaux de neurones qui serviront de détecteurs, puis nous exploitons les sorties de ces deux réseaux à l'aide d'un troisième réseau.

À partir de ce choix, notre architecture se spécialise fortement, mais cela est inévitable si l'on souhaite résoudre un problème pratique. Toutefois l'architecture pourra évoluer assez facilement, car elle est modulaire : ses composants peuvent être remplacés par d'autres plus performants.

Détection des obstacles à l'avant

Nous avons tout d'abord réalisé un détecteur des obstacles situés à l'avant, à l'aide d'un réseau multicouches. Ce réseau prend en entrée :

- les valeurs des 3 capteurs situés à l'avant (v_1, v_2, v_3),
- la valeur de la vitesse linéaire du robot $(vitesse)$.

5. Cette architecture permet par exemple de prendre en compte le temps, alors que le système flou en question ignore totalement cette dimension. En particulier, nous parlerons de l'utilisation de réseaux récurrents du type SRN dans les perspectives.

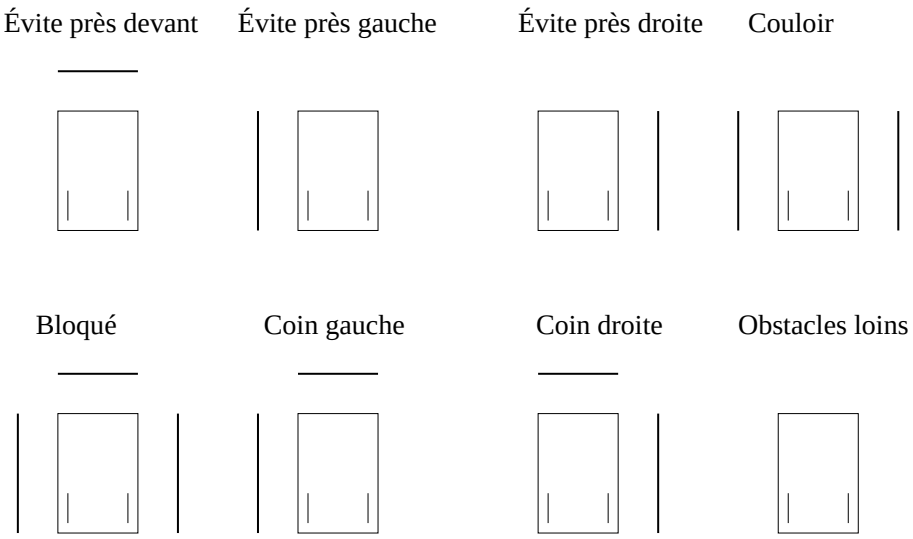


Fig. 5.7 - *D'après Reignier [1994b]. Les huit situations perceptives que nous voulons détecter à l'aide de réseaux de neurones.*

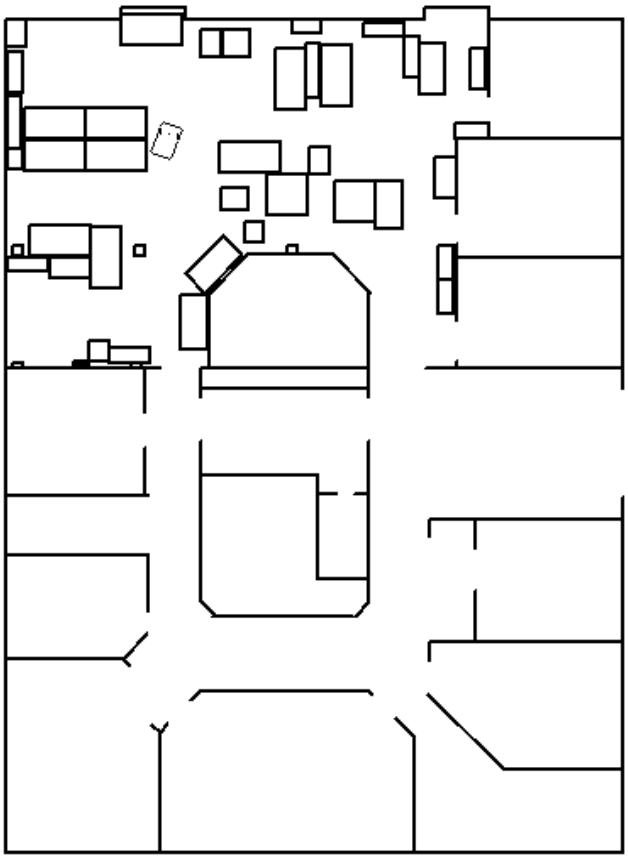


Fig. 5.8 - *L'environnement que nous avons utilisé dans le simulateur MOLUSC pour entraîner et tester nos réseaux de neurones. Cet environnement représente le rez-de-chaussée du LIFIA. On distingue le robot en haut et à gauche de la figure.*

Ces valeurs sont normalisées. Le réseau doit répondre 0 s'il n'y a pas d'obstacle et 1 s'il y a un obstacle. Normalement, il faudrait donner « à la main » la bonne réponse pour chaque entrée, mais cela est relativement délicat en pratique : un opérateur doit indiquer la bonne réponse en appuyant sur certaines touches du clavier quand il reconnaît certaines situations, et il doit prendre grand soin de fournir des réponses cohérentes. Aussi, pour gagner du temps (notre but est ici de réaliser rapidemment un prototype), nous avons réalisé une simple fonction d'évaluation permettant de donner la « bonne » réponse :

$$f(v_1, v_2, v_3, vitesse) = \begin{cases} 0 \text{ si } min(v_1, v_2, v_3) > k_1 * vitesse^2 + k_2 \\ 1 \text{ sinon} \end{cases}$$

avec $k_1 = 100 s^2/m$ et $k_2 = 0.5m$ dans nos simulations.

Cette fonction peut bien sûr être améliorée, par exemple en calculant exactement la distance de sécurité nécessaire à l'arrêt (il faut alors tenir précisément compte de la manière dont les profils de vitesse sont gérés par le simulateur).

L'utilisation de cette fonction ne nuit pas à la généralité de notre approche, et est simplement destinée à gagner du temps. Notons que l'intérêt de l'apprentissage est justement de faciliter la modélisation, en permettant utiliser des jeux d'exemples au lieu d'identifier de telles fonctions. En effet l'utilisation de telles fonctions d'évaluation est difficile, car il faut d'une part déterminer le type de fonctions, et ensuite régler les paramètres comme k_1 et k_2.

Grâce à cette fonction et aux possibilités de téléopérations offertes par MOLUSC, nous avons produit un fichier de 348 exemples qui nous a permis d'entraîner le réseau multicouches de la figure 5.9. Le robot a été téléopéré dans la partie inférieure de l'environnement de la figure 5.8.

Après apprentissage avec la règle de rétropropagation[6], 99.4% des exemples ont été appris. Nous avons effectués trois tests de la capacité de généralisation de ce réseau en téléopérant le robot dans d'autres parties de l'environnement de la figure 5.8. Ces trois tests ont donné les résultats suivants :

	Nombre d'exemples présentés	Pourcentage de réponses correctes
test1	198	94.4%
test2	363	83.5%
test3	515	87.5%

À ce stade, nous considérons que ces pourcentages sont suffisamment élevés. De toute manière, il est impossible d'obtenir 100% de réponses correctes puisque l'on travaille avec des données bruitées. De plus notre ensemble d'apprentissage n'est peut-être pas suffisamment représentatif de toutes les situations possibles.

6. En fait, nous avons souvent utilisé l'heuristique suivante : utiliser la rétropropagation avec moment jusqu'à ce que l'erreur ne diminue plus, puis utiliser RPROP (*Resilient Propagation*) [Zell et al. 1993] ; RPROP permet alors une réduction substantielle de l'erreur.

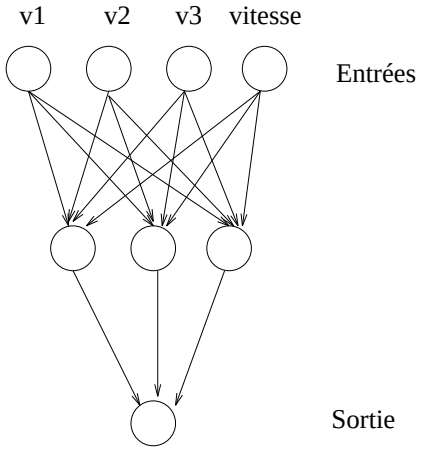


Fig. 5.9 - *Architecture du réseau permettant de détecter les obstacles situés devant le robot. Chaque unité a pour fonction de transfert la fonction SNNS*

$$Logistic(x,\theta) = \frac{1}{1+e^{-(x+\theta)}},$$

où θ est un biais propre à chaque unité. Une architecture encore plus simple aurait éventuellement pu convenir.

Détection des obstacles à gauche

Dans un deuxième temps nous avons cherché à réaliser un détecteur pour les obstacles situés sur le côté gauche du robot.

Pour entraîner ce deuxième détecteur (3 entrées, 3 unités cachées, 1 sortie), nous n'avons pas utilisé de fonction d'évaluation, et nous avons indiqué pour chaque exemple la bonne sortie[7]. Ceci a été fait sans trop de difficultés et a permis d'obtenir un fichier de 541 exemples, mais il n'est pas possible de garantir que tous les exemples du jeu d'apprentissage sont corrects à cause des inévitables erreurs de l'opérateur qui indique les bonnes réponses.

Une fois l'apprentissage terminé, on obtient 95% d'exemples appris, ce qui est très satisfaisant compte tenu de la présence d'exemples incorrects. Sur un jeu de test on obtient :

	Nombre d'exemples présentés	Pourcentage de réponses correctes
test1	624	91%

Combinaison de ces deux détecteurs

Nous sommes maintenant en mesure de détecter les huit situations perceptives. En effet le détecteur à gauche peut aussi être utilisé à droite, et avec trois détecteurs

7. Pour cela nous avons légèrement modifié MOLUSC, de telle sorte que l'appui sur une touche du clavier permette d'inverser une variable booléenne codant la bonne réponse.

on peut identifier les huit situations. On pourrait utiliser un ensemble de règles du type :

```
SI est-proche(détecteur-avant,0)
ET est-proche(détecteur-gauche,0)
ET est-proche(détecteur-droite,0)
ALORS OBSTACLES-LOINS

SI est-proche(détecteur-avant,0)
ET est-proche(détecteur-gauche,0)
ET est-proche(détecteur-droite,1)
ALORS ÉVITE-PRÈS-DROITE
```

Mais cela implique de définir arbitrairement des seuils pour préciser la notion de proximité. En effet, les détecteurs ne sont pas réellement binaires[8] et donnent des valeurs comprises entre 0 et 1. Aussi il est plus cohérent avec notre approche de réaliser un nouveau réseau de neurones approchant la fonction qui permet de passer des 3 sorties des détecteurs aux 8 situations. Le réseau réalisé comporte 3 entrées, 4 unités cachées, et 8 sorties, et a été entraîné avec les 8 exemples d'associations.

Nous sommes donc maintenant capables de détecter les huit situations perceptives.

5.3.2 Réseau versatile

Quelques détails sur la programmation de ces réseaux sont donnés en annexe B.2. Ici nous présentons seulement le réseau que nous avons effectivement intégré dans MOLUSC, et qui nous sert à représenter des règles de décision.

Construction du réseau

Ces règles sont les suivantes[9] [Reignier 1994b] :

Évite près devant : on arrête le mouvement de translation du robot et on demande une rotation lente à droite.

Évite près gauche : on ne fournit pas de consigne pour la vitesse de translation et on demande une rotation lente vers la droite de manière à s'éloigner de l'obstacle.

Évite près droite : de même, on ne fournit pas de consigne pour la vitesse de translation et on demande une rotation lente sur la gauche.

Couloir étroit : pas de consigne pour la vitesse linéaire. On demande une vitesse angulaire nulle pour rester dans l'axe du couloir.

8. C'est un choix de conception. En effet nous ne souhaitons pas imposer une discrétisation dès le premier niveau, puisque les niveaux supérieurs sont capables de traiter les données continues.

9. Nous sommes conscients de la simplicité de ces règles, et il sera bien sûr nécessaire d'exploiter les bases de règles plus complètes proposées par Reignier [1994b], une fois que notre premier prototype sera au point.

blocage : on arrête le mouvement de translation du robot et on demande une rotation lente vers la droite (de manière soit à réaliser un demi-tour, soit à apercevoir une issue par changement de point de vue résultant de la rotation).

Coin gauche : on arrête le mouvement de translation du robot et on demande une rotation lente à droite.

Coin droit : de même, on arrête le mouvement de translation du robot et on demande une rotation lente à gauche.

Quatre décisions sont donc possibles : arrêt translation, rotation lente à droite, rotation lente à gauche, vitesse angulaire nulle. Par conséquent, le réseau versatile comportera huit entrées (les huit situations perceptives) et quatre sorties (les quatre décisions).

Une fois l'architecture déterminée, il faut trouver (empiriquement comme nous l'avons signalé page 42) les valeurs des coefficients c_i, d_{ij} et w_{ik}.

Pour le moment, nous ne souhaitons pas privilégier une décision par rapport aux autres, donc nous imposons la même valeur aux poids des connexions inhibitrices. Ainsi, la matrice d'inhibition, nécessairement symétrique, est :

$$(d_{ij}) = \begin{pmatrix} 0 & d & d & d \\ d & 0 & d & d \\ d & d & 0 & d \\ d & d & d & 0 \end{pmatrix}$$

avec $d = 0.2$ dans nos expérimentations numériques. Cette matrice pourra éventuellement être modifiée ultérieurement pour permettre à telle ou telle décision de l'emporter plus rapidement.

Les c_i sont liés aux d_{ij} par l'importante relation

$$\min c_i > \max \sum_{j=1}^{n} |d_{ij}|$$

qui garantit la convergence globale du réseau. Là aussi, nous ne souhaitons pas particulièrement prilégier une décision, si bien que nous prendrons comme vecteur de fatigue :

$$(c_i)^T = (c,\ c,\ c,\ c)$$

avec $c = 1$ dans nos expérimentations numériques, ce qui respecte la condition ci-dessus.

Il reste à déterminer les poids w_{ik}, nécessairement positifs ou nuls, des connexions entre la couche d'entrée et la couche de sortie. Nous prenons comme point de départ la règle empirique suivante[10] :

$$w_{ik} = \frac{1}{\text{Nombre de connexions arrivant sur l'unité } i}$$

10. Elle a déjà été utilisée par A. Labbi dans ses divers exemples.

Il faut maintenant préciser le nombre de connexions arrivant sur chaque unité de sortie : cela revient à coder sous forme de réseau les règles ci-dessus, ce qui donne le réseau de la figure 5.10. Grâce à ce codage et à la règle empirique ci-dessus, on obtient la matrice des poids de la figure 5.11.

Quelques expériences

Nous avons réalisé quelques tests du réseau décrit ci-dessus, indépendamment du simulateur de robot.

La figure 5.12 correspond au scénario suivant :

- à $t = 0\ s$, $X(0)$ est le vecteur nul, et tous les obstacles sont loins (l'unité d'entrée correspondante a 1 pour sortie, et toutes les autres ont 0 pour sortie) ; aucune décision ne l'emporte sur les autres et le réseau converge vers un état d'équilibre ;
- à $t = 1\ s$, un obstacle se présente sur la droite ; par conséquent, la décision de tourner à gauche commence à l'emporter sur les autres ;
- à $t = 2\ s$, un obstacle apparaît devant ; par conséquent, les activations correspondant à l'arrêt de la translation et à la rotation sur la droite commencent à augmenter, et inhibent progressivement la décision précédente qui était de tourner à gauche ;
- à $t = 3\ s$, un coin se présente sur la droite ; la décision de tourner à gauche reprend alors le dessus ;
- à $t = 4\ s$, un obstacle apparaît à gauche ; peu à peu la décision de tourner à droite l'emporte ;
- enfin à $t = 5\ s$, on se trouve dans un couloir ; la décision d'obtenir une vitesse angulaire nulle est alors prise ; on laisse se prolonger cet état pendant deux secondes afin de bien observer la convergence.

Le deuxième scénario (fig. 5.13) présente des caractéristiques voisines, mais :

- on laisse le réseau converger durant une période plus longue que dans le premier scénario ($2\ s$ au lieu de $1\ s$), afin de mieux illustrer cette convergence ;
- on a repris certains événements du premier scénario, afin de montrer que les mêmes états d'équilibre étaient atteints bien que les conditions initiales soient fort différentes (voir par exemple l'apparition d'un couloir, d'obstacles loins, d'un obstacle à droite sur la figure 5.13).

Ces deux scénarios illustrent bien le mécanisme de changement de décision quand surviennent des changements dans les données sensorielles. Ils montrent aussi que la propriété théorique de convergence globale est bien respectée par notre programme, et que la convergence se produit en un temps relativement court, de l'ordre de la seconde.

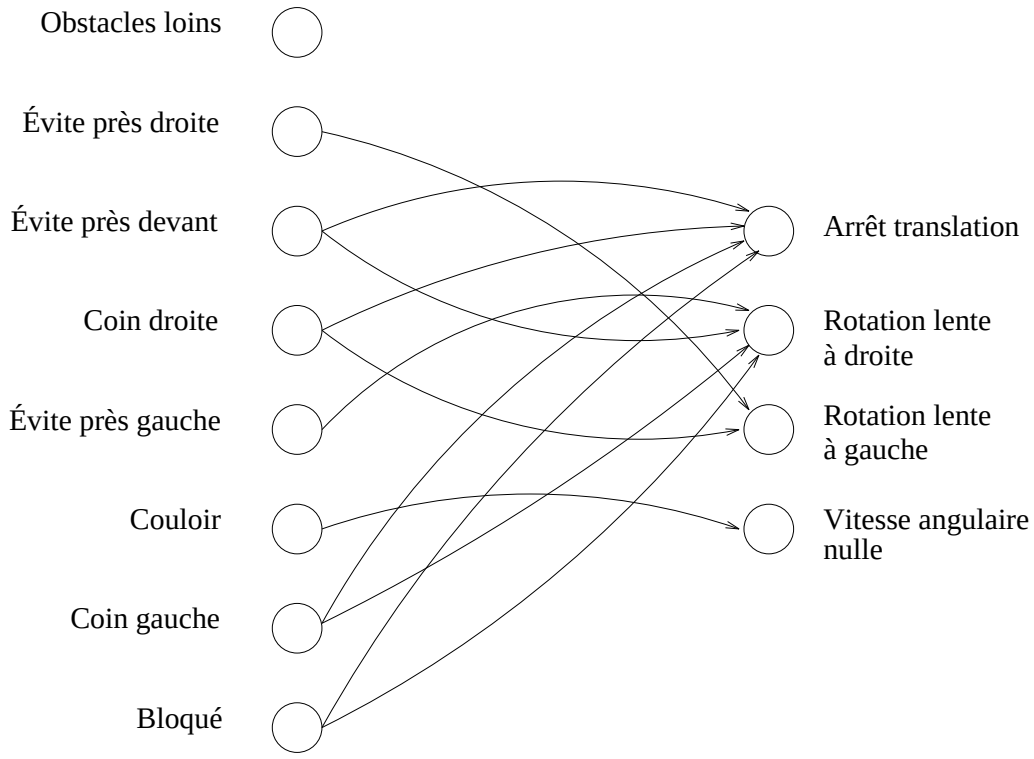


Fig. 5.10 - *Connexions entre la couche d'entrée et la couche de sortie de notre réseau versatile. L'unité « Obstacles loins » n'a pas d'influence directe sur les décisions. N.B. : il y a aussi des connexions inhibitrices entre les unités de sortie, mais elles ne sont pas indiquées sur la figure.*

	Bloqué	Coin gauche	Couloir	Évite près gauche	Coin droite	Évite près devant	Évite près droite	Obstacles loins
Arrêt translation	*p1*	*p1*	*0*	*0*	*p1*	*p1*	*0*	*0*
Rotation lente à droite	*p2*	*p2*	*0*	*p2*	*0*	*p2*	*0*	*0*
Rotation lente à gauche	*0*	*0*	*0*	*0*	*p3*	*0*	*p3*	*0*
Vitesse angulaire nulle	*0*	*0*	*p4*	*0*	*0*	*0*	*0*	*0*

Fig. 5.11 - *Matrice des poids w_{ik} des connexions ci-dessus. Dans les expérimentations numériques nous avons pris :*

$$p1 = 0.25,\ p2 = 0.25,\ p3 = 0.5, p4 = 1$$

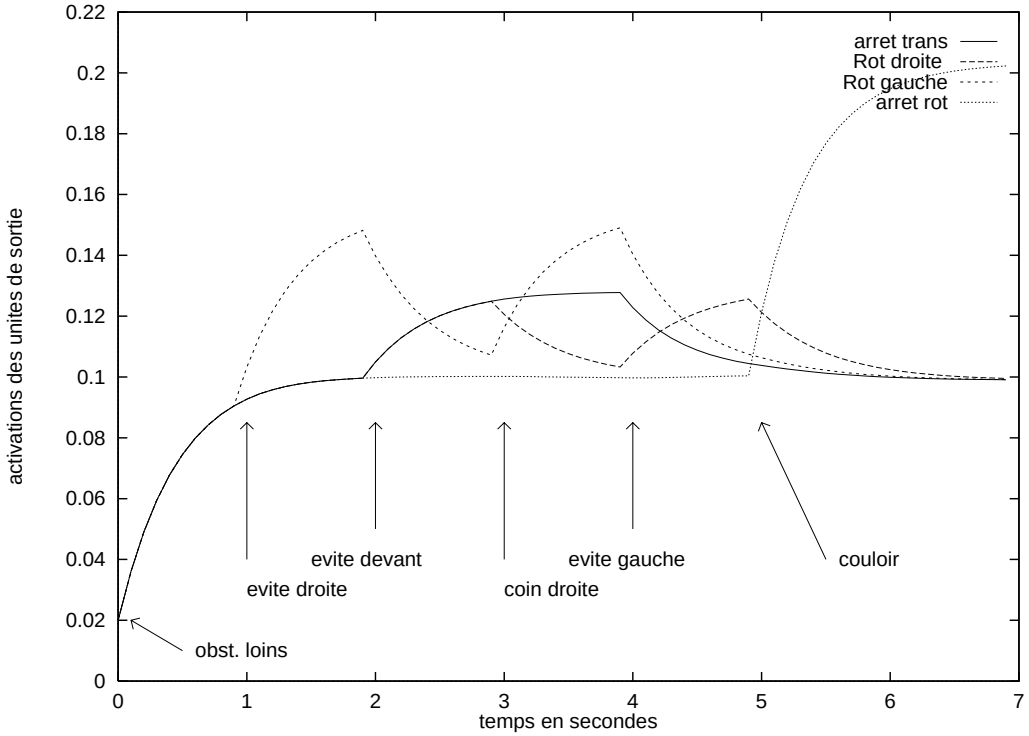


Fig. 5.12 - *Résultats du premier scénario d'utilisation du réseau versatile. Commentaires dans le texte.*

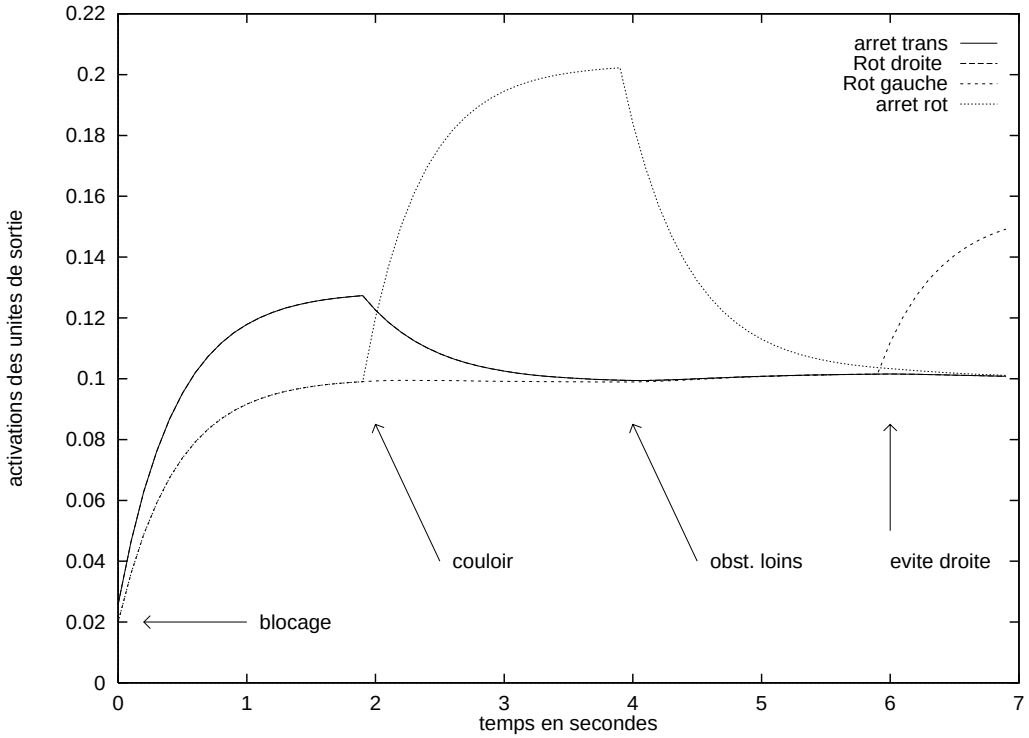


Fig. 5.13 - *Résultats du deuxième scénario d'utilisation du réseau versatile. Commentaires dans le texte.*

5.3.3 Intégration dans MOLUSC

Cette intégration pose principalement deux problèmes. Le premier problème concerne les aspects purement informatiques de l'intégration des réseaux développés avec SNNS, et le deuxième concerne l'utilisation du réseau versatile pour piloter effectivement le robot.

Intégration des réseaux SNNS

L'inconvénient de la bibliothèque de fonctions SNNS dont nous avons parlé plus haut est que toutes ces fonctions ne peuvent travailler que sur **un seul réseau à la fois** : le réseau et ses caractéristiques sont des variables globales. Par conséquent, il n'est pas possible dans un programme utilisant ces fonctions d'exploiter plusieurs réseaux en même temps.

Étant donné que nous voulons utiliser les divers réseaux ci-dessus, nous avons adopté la stratégie suivante :

1. écrire un programme C prenant un nom de réseau en paramètre ; ce programme (`appel_reseau`) charge ce réseau, puis se met en attente d'une lecture sur son entrée standard ; pour chaque lecture effectuée, le programme initialise les unités d'entrée du réseau avec les valeurs lues, puis effectue la propagation des activations, et enfin écrit les sorties du réseau sur la sortie standard ;
2. développer une petite bibliothèque de fonctions permettant à n'importe quel programme C sous Unix de communiquer via des tubes (*pipes*) avec un autre programme utilisant ses entrées/sorties standards ; cette bibliothèque est décrite en annexe B.3 ;
3. utiliser dans MOLUSC ces fonctions de communication.

Note : l'utilisation dans MOLUSC des réseaux versatiles ne pose par contre aucun problème, étant donné que nous avons développé notre propre bibliothèque de fonctions (annexe B.2), qui permet éventuellement d'utiliser plusieurs réseaux versatiles.

Utilisation effective du réseau versatile

Afin de vérifier le bon fonctionnement de l'ensemble des réseaux, nous avons téléopéré[11] le robot pendant un certain temps dans l'environnement de la figure 5.8, et tracé les activations des quatre unités de sortie sur la figure 5.14. Cette figure montre simplement que l'on retrouve bien les divers états d'équilibre observés sur les figures 5.12 et 5.13.

Cette vérification élémentaire étant faite, un nouveau problème se pose : quelles commandes (vl, va) faut-il envoyer au robot ? Ou encore, quel sens peut-on donner

11. Rappelons que dans tout ce chapitre, nous parlons uniquement du robot **simulé**. Aucune expérience n'a été faite sur le robot réel.

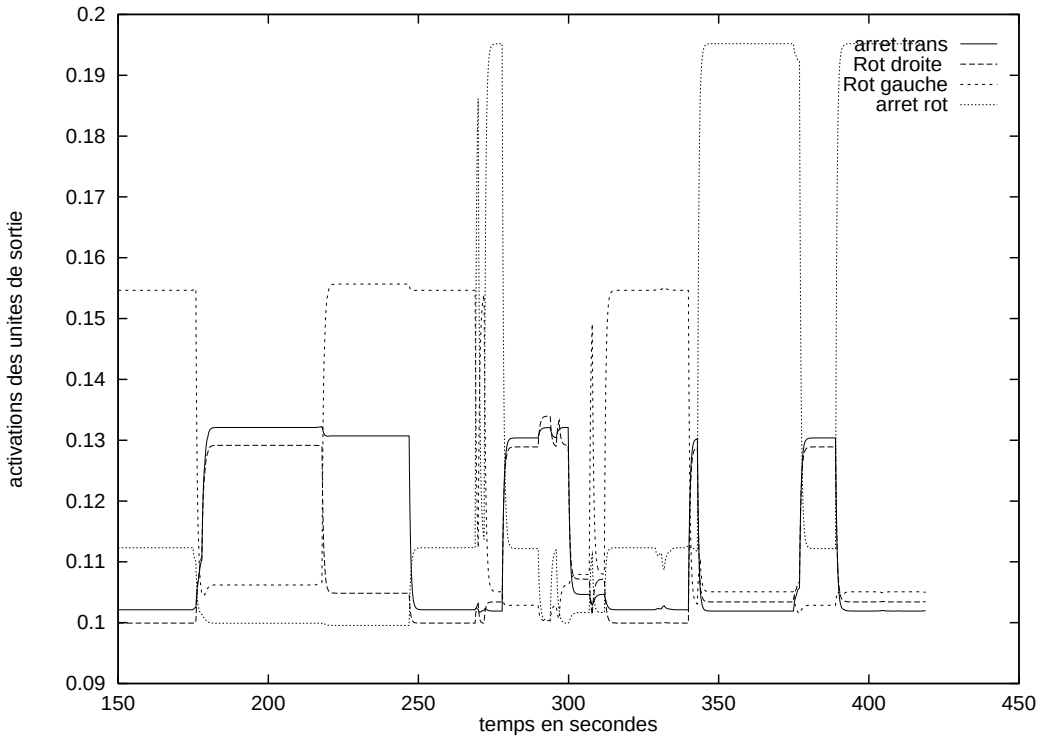


Fig. 5.14 - *Activations des unités de sortie du réseau versatile, comme sur les figures précédentes. Mais ici, les entrées sont obtenues à partir des données capteurs.*

aux activations des quatre unités de sortie du réseau versatile? L'étude théorique de Labbi [1993] ne donne aucune réponse.

Une première méthode (niveau neuronal) : Une première solution consiste à utiliser un réseau pour associer à ces activations des valeurs (vl, va) ; pour cela, on peut téléopérer le robot, et enregistrer les valeurs des activations ainsi que les couples (vl, va) donnés par l'opérateur. Mais nos expérimentations ont montré que l'opérateur avait beaucoup de mal à donner des réponses consistantes : en effet, pour chaque situation perceptive, de nombreux couples (vl, va) sont possibles. Par conséquent, le jeu d'exemples ainsi obtenu comporte beaucoup d'incohérences et ne permet pas de réaliser un apprentissage satisfaisant.

Pour chaque situation, un certain choix de décisions existe, et ce choix est difficile à effectuer au niveau neuronal, surtout avec des réseaux approximateurs de **fonctions**. Aussi il est préférable de reporter la prise de décision au niveau symbolique, où il est possible par exemple de choisir après un raisonnement.

Une deuxième méthode (niveau symbolique) : Une deuxième solution consiste à transformer directement ces activations en valeurs (vl, va). Nous avons choisi la méthode suivante :

calcul de vl **:** $vl(t) = k_{vl}^{scale} \cdot (k_{vl}^{max} - U_0(t))$

$$\text{où} \begin{cases} U_0 & = & \text{activation de l'unité arrêt translation} \\ k_{vl}^{max} & = & \max U_0(t) \\ & & (\approx 0.13 \text{ dans nos expérimentations}) \\ k_{vl}^{scale} & = & \text{coefficient arbitraire} \\ & & (10 \text{ dans nos expérimentations}) \end{cases}$$

$$\textbf{calcul de } va: \; va(t) = \begin{cases} 0 & \text{si} \quad U_3(t) = \max_{i=1}^{3} U_i(t) \\ -k_{va}^{scale} \cdot U_1(t) & \text{si} \quad U_1(t) = \max_{i=1}^{3} U_i(t) \\ k_{va}^{scale} \cdot U_2(t) & \text{si} \quad U_2(t) = \max_{i=1}^{3} U_i(t) \end{cases}$$

$$\text{où} \begin{cases} U_1 & = & \text{activation de l'unité rotation à droite} \\ U_2 & = & \text{activation de l'unité rotation à gauche} \\ U_3 & = & \text{activation de l'unité arrêt rotation} \\ k_{va}^{scale} & = & \text{coefficient arbitraire} \\ & & (100 \text{ dans nos expérimentations}) \end{cases}$$

En effectuant ces calculs, on se place en fait à un **niveau symbolique**, certes rudimentaire, mais notre système est un simple prototype. Ceci illustre bien la démarche de développement incrémentale dont nous avons parlé ci-dessus : après avoir constaté une limite des niveaux inférieurs (difficulté de produire va et vl par apprentissage), nous tentons de la résoudre à un niveau supérieur.

Notons que l'utilisation de telles fonctions ne permet pas de prendre la bonne décision dans tous les cas, comme nous le verrons ci-dessous (situations d'indécision). Par conséquent il sera nécessaire d'enrichir le niveau symbolique.

Dans la suite de ce chapitre, les valeurs (vl, va) ainsi calculées seront appelées *vitesses demandées* par opposition aux *vitesses obtenues*, qui sont celles réellement fournies par le simulateur qui tient compte d'un certain nombre de contraintes cinématiques.

Une troisième méthode (niveau neuronal) : Une troisième méthode, que nous n'avons pas expérimentée, consiste à éviter la modélisation ci-dessus, qui est délicate (choix des fonctions et de leurs paramètres), en revenant au niveau neuronal pour utiliser un mécanisme d'apprentissage. En effet, il est possible de reprendre l'idée de la première méthode, mais sans utiliser la téléopération pour donner les bonnes réponses au réseau. Une meilleure solution consisterait à identifier la vitesse angulaire et la vitesse linéaire désirée pour **chaque état d'équilibre** et aussi pour **chaque transition** entre états d'équilibres.

5.4 Expériences

Nous avons réalisé plusieurs expériences avec notre architecture intégrée dans le simulateur MOLUSC, afin de déterminer les points forts et les points faibles de notre approche.

5.4.1 Expérience 1 : coin inférieur gauche du LIFIA

Dans un premier temps, nous avons laissé le robot parcourir le coin inférieur gauche de l'environnement de la figure 5.8. La figure 5.15 montre la trajectoire suivie par le robot. On constate notamment que :

- le robot montre une forte propension à parcourir des trajectoires circulaires ; cela découle directement de la valeur du paramètre k_{va}^{scale}. En changeant cette valeur, on obtient d'autres types de trajectoires, mais sa valeur doit rester relativement élevée ; sinon, il est possible que le robot ne tourne pas suffisamment vite face à un obstacle ;
- le robot ne cherche pas particulièrement à sortir de la pièce : cela est parfaitement normal, car on ne lui a donné aucun but particulier ; le seul comportement qui lui a été imposé est d'**éviter les obstacles**.

La figure 5.16 permet d'analyser plus finement le comportement du robot. Elle présente les activations des quatre unités de sortie du réseau versatile.

Cette figure montre notamment la forme très particulière de l'activation de l'unité arrêt translation : comme la vitesse linéaire demandée (fig. 5.17) dépend directement de cette activation, le robot passe périodiquement d'une vitesse linéaire d'environ 0.3 m/s à une vitesse linéaire quasi-nulle. Ce comportement est dû au détecteur d'obstacles situé à l'avant du robot, et qui tient compte de la vitesse du robot : dès qu'elle est trop importante (au sens de la fonction d'évaluation que nous avions utilisée pour l'entraînement du réseau correspondant), le robot détecte un obstacle, ce qui provoque la demande d'une vitesse linéaire nulle ; une fois que la vitesse est devenue suffisamment faible, le détecteur ne détecte plus d'obstacle, et le robot recommence à avancer, et le phénomène se répète dès que la vitesse est suffisamment importante. Nous verrons ci-dessous comment corriger ce phénomène qui peut être dommageable pour un robot réel.

Ceci explique par exemple le changement de décision après 4 s : tout en tournant légèrement à gauche, le robot à une vitesse linéaire importante et considère très rapidement qu'un obstacle se trouve devant lui ; d'autre part, comme il a légèrement tourné sur la gauche, il détecte à la fois un obstacle sur la gauche et la droite, si bien qu'il se considère en situation de blocage. Par conséquent, il lui faut, d'après les règles de décision, s'arrêter et tourner à droite, ce qu'il fait comme l'indique la première impulsion négative de la figure 5.17. Ayant légèrement tourné à droite, il ne détecte plus qu'un obstacle sur la droite, et par conséquent, c'est la décision de tourner à gauche qui l'emporte pendant une quinzaine de secondes : la vitesse angulaire effectivement obtenue augmente alors progressivement (fig. 5.17) tandis que la vitesse linéaire continue à augmenter et diminuer périodiquement (fig. 5.18).

Ensuite vers $t = 14s$, le robot se considère à nouveau en situation de blocage, d'où la victoire de l'unité arrêt translation. Brève victoire, car dès que la vitesse linéaire a diminué, aucun obstacle n'est détecté à l'avant, et le robot se considère alors comme étant dans un couloir (obstacles à droite et à gauche), si bien que l'unité arrêt rotation l'emporte.

Le fonctionnement du détecteur des obstacles à l'avant explique aussi le comportement que l'on observe entre 35 et 60 s : une rotation prolongée à droite, mais avec une faible vitesse angulaire. Le robot détecte en fait périodiquement la situation Obstacles loins (d'où rotation à gauche) et la situation Obstacle devant (d'où rotation à droite).

Pourquoi le robot tourne-t-il à gauche dans la situation Obstacles loins ? Rappelons que l'unité Obstacles loins n'a pas d'influence directe sur les décisions. Elle a une influence indirecte, car quand elle est activée, toutes les autres unités d'entrée du réseau versatile sont en principe inactives, ce qui a pour conséquence de provoquer la convergence du réseau vers un état où toutes les unités de sortie ont la même activation. Quand le réseau est dans cet état, la décision prise dépend uniquement de la programmation du calcul de va indiqué page 162. Cette décision prise implicitement est un inconvénient de notre système actuel, mais nous avons voulu coder directement les règles d'évitement d'obstacles proposées page 155. De plus la correction est immédiate : il suffit d'ajouter une connexion entre les unités Obstacles loins et arrêt rotation par exemple ; il faut aussi modifier les poids des autres connexions arrivant sur cette unité, car ils sont fonction du nombre du connexions.

Ce comportement du détecteur avant n'est pas anormal : il effectue la tâche qui lui a été enseignée. Aussi, pour supprimer ces variations périodiques de la vitesse linéaire, il vaut mieux chercher tout d'abord à corriger la fonction qui permet de calculer la vitesse linéaire. Nous avions pris :

$$vl(t) = k_{vl}^{scale} \cdot (k_{vl}^{max} - U_0(t))$$

Afin d'obtenir une augmentation moins rapide de $vl(t)$, nous avons choisi une fonction du carré de $U_0(t)$) :

$$vl(t) = k_{vl}^{scale'} \cdot (k_{vl}^{max} - U_0(t))^2$$

avec $k_{vl}^{scale'} = 100$. Étant donné que la vitesse linéaire généralement obtenue est alors plus faible que celle de la première expérience, il a été aussi nécessaire de réduire le coefficient k_{va}^{scale} de moitié (sinon le robot suit une trajectoire circulaire de faible rayon).

On obtient alors la nouvelle trajectoire de la figure 5.19. Les figures 5.20, 5.21 et 5.22 montrent que le phénomène de variation périodique de vl est partiellement corrigé : durant de longues périodes on obtient des vitesses relativement stables.

5.4.2 Expérience 2 : couloir

La trajectoire obtenue dans cette deuxième expérience est montrée par la figure 5.23.

Dans cette expérience, le robot détecte bien les obstacles, mais il touche certains car il ne tourne pas assez rapidement. La figure 5.24 permet de comprendre pourquoi : on remarque que la décision de tourner à gauche (provoquée par la détection d'un obstacle devant et d'un obstacle à droite) est aussitôt contrecarrée par la décision de

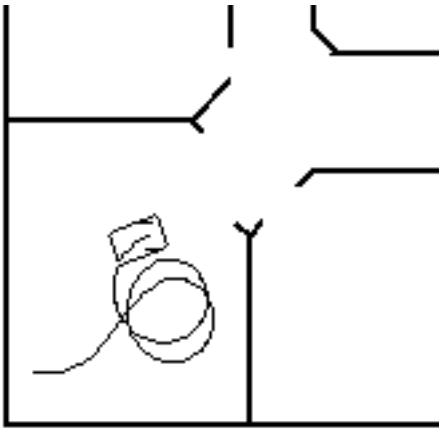


Fig. 5.15 - *Trajectoire suivie par le robot, durant 6 minutes.*

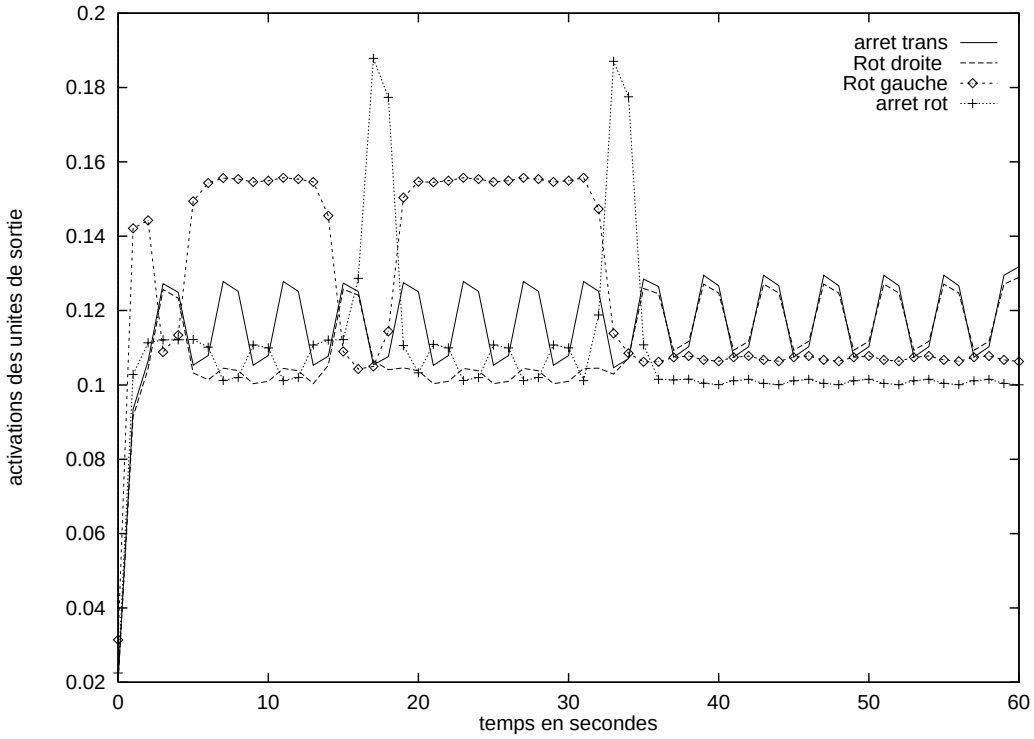


Fig. 5.16 - *Activations des unités de sortie du réseau versatile, durant la première minute de l'expérience ci-dessus, au cours de laquelle le robot s'est déplacé jusqu'avant le petit plateau horizontal qui précède une forte rotation à droite.*

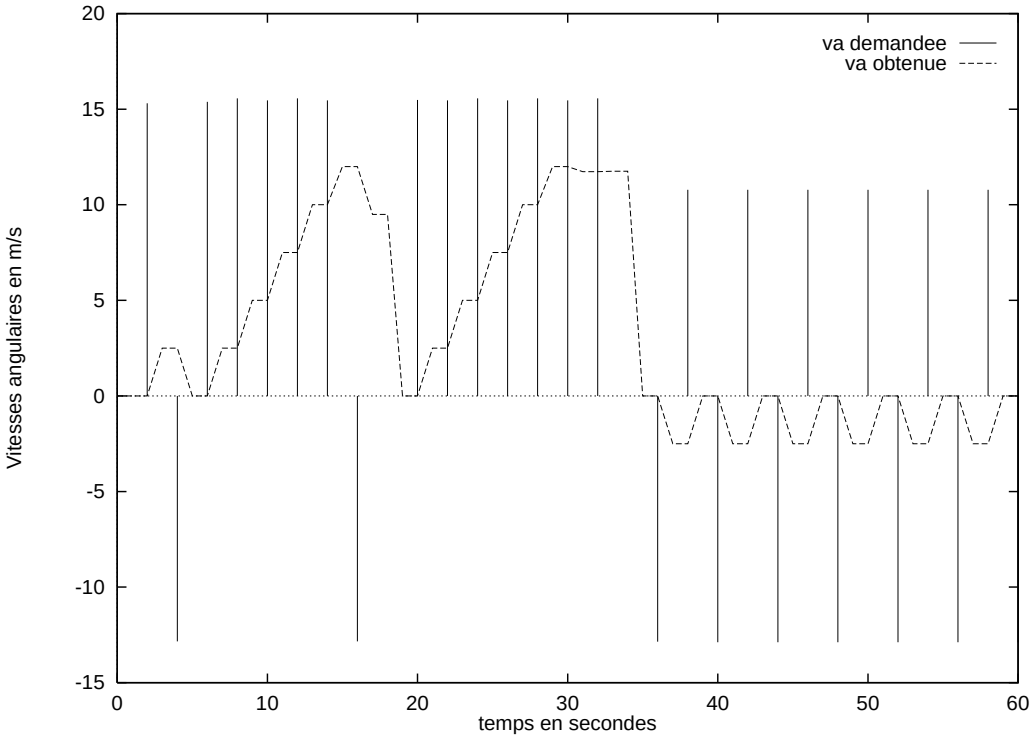


Fig. 5.17 - *Vitesse angulaire demandée (tracée sous forme d'impulsions) et vitesse angulaire obtenue, durant la première minute de l'expérience de la figure 5.15. Commentaires dans le texte.*

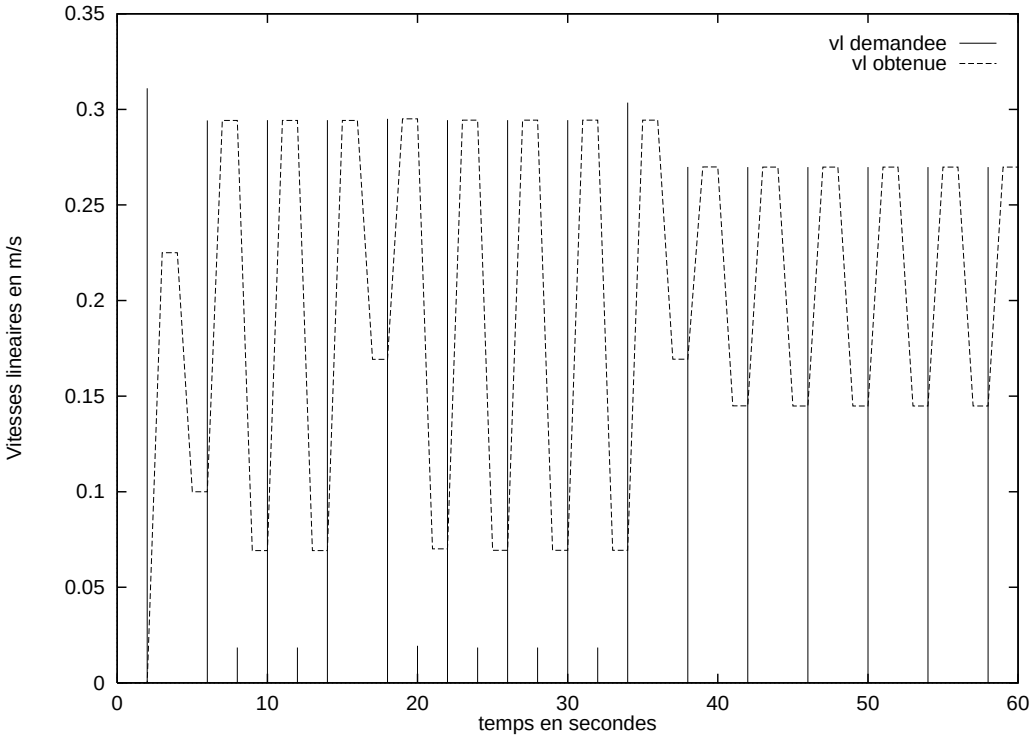


Fig. 5.18 - *Vitesse linéaire demandée (tracée sous forme d'impulsions) et vitesse linéaire obtenue, durant la première minute de l'expérience de la figure 5.15. Commentaires dans le texte.*

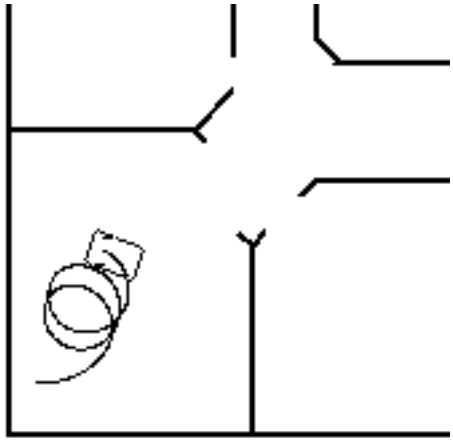


Fig. 5.19 - *Trajectoire suivie par le robot, durant 10 minutes, avec la nouvelle méthode de calcul de* $vl(t)$ *:*

$$vl(t) = k_{vl}^{scale'} \cdot (k_{vl}^{max} - U_0(t))^2$$

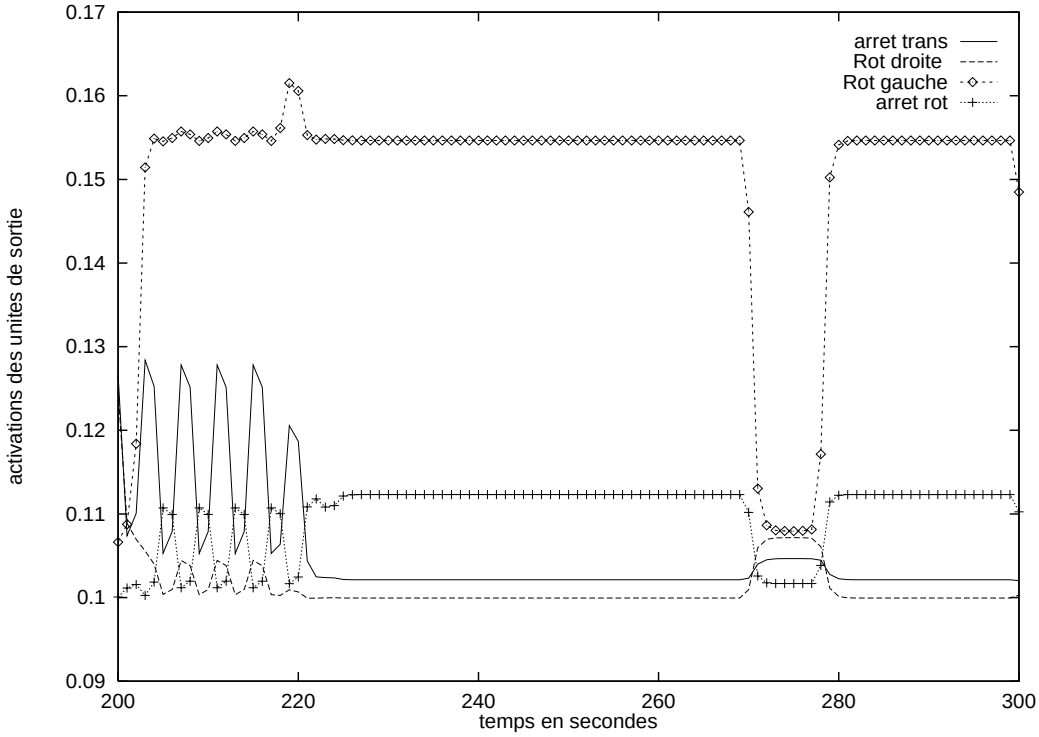


Fig. 5.20 - *Activations des unités de sortie du réseau versatile, durant une partie de l'expérience ci-dessus.*

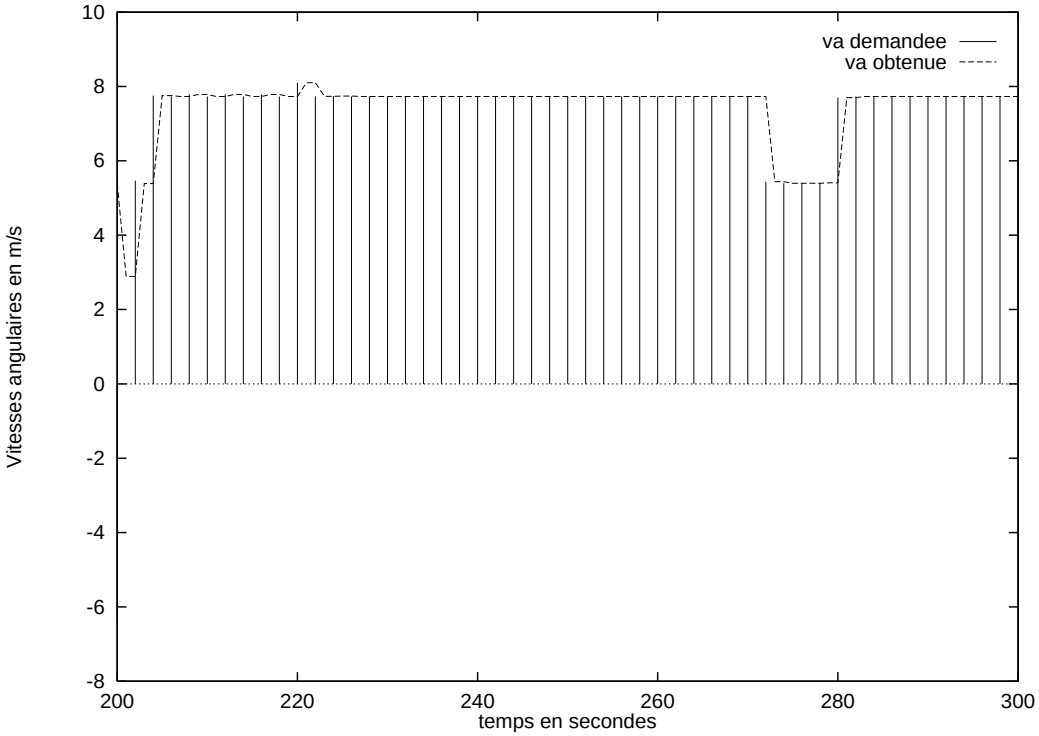


Fig. 5.21 - *Vitesse angulaire demandée (tracée sous forme d'impulsions) et vitesse angulaire obtenue, durant une partie de l'expérience de la figure 5.19. Commentaires dans le texte.*

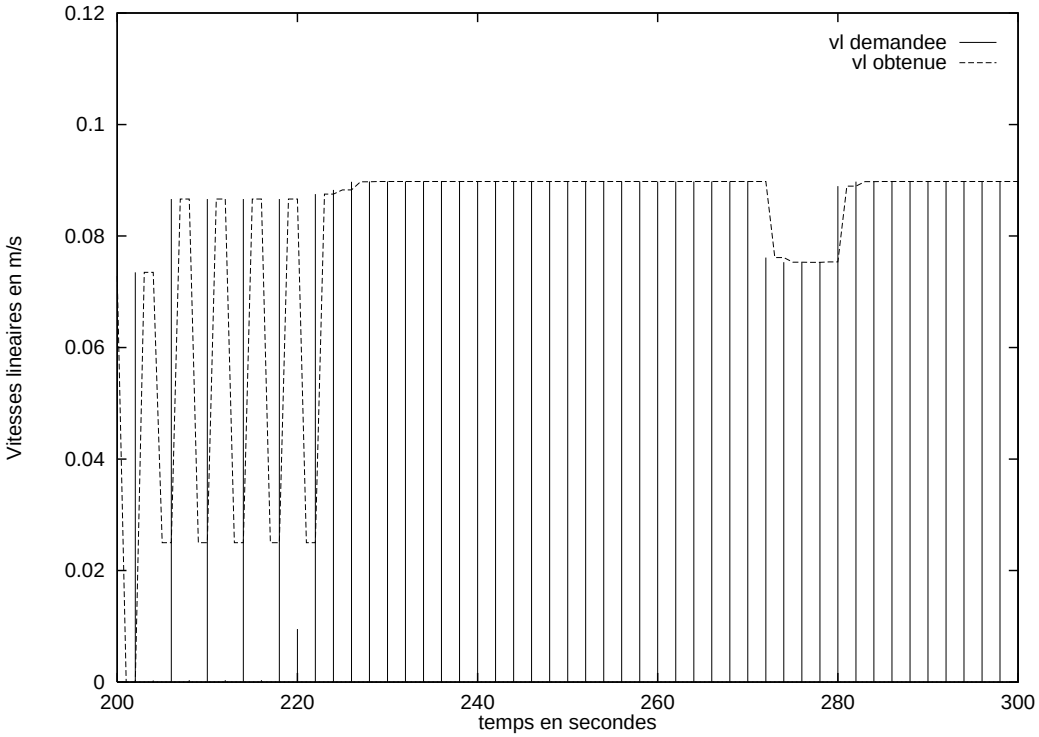


Fig. 5.22 - *Vitesse linéaire demandée (tracée sous forme d'impulsions) et vitesse linéaire obtenue, durant une partie de l'expérience de la figure 5.19. Commentaires dans le texte.*

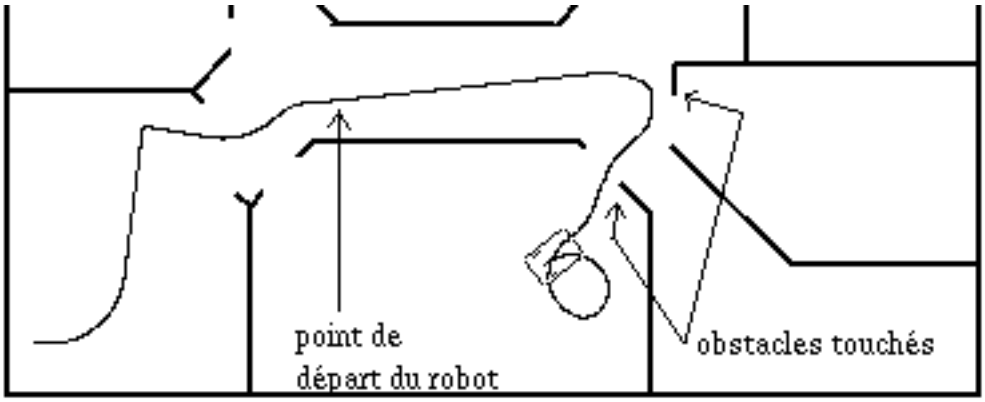


Fig. 5.23 - *Trajectoire suivie par le robot, durant 28 minutes, avec la nouvelle méthode de calcul de $vl(t)$. Nous avons téléopéré le robot jusqu'à l'entrée du couloir (marque « point de départ »), puis notre système a pris le contrôle.*

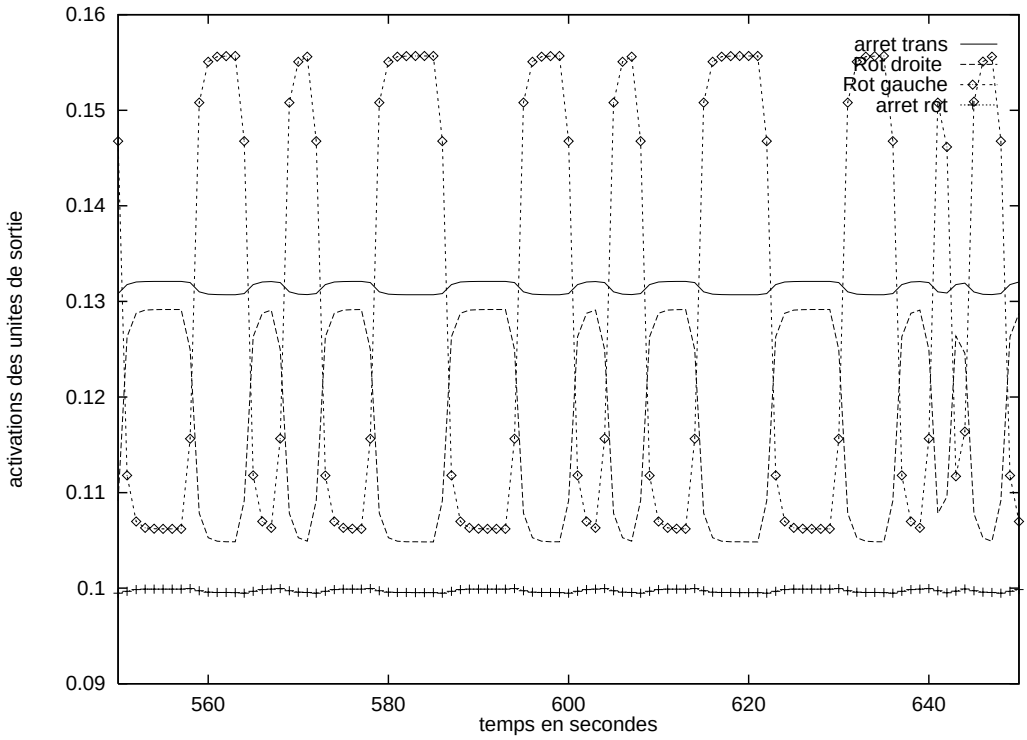


Fig. 5.24 - *Activations des unités de sortie du réseau versatile, durant une partie de l'expérience ci-dessus (rotation à droite au bout du couloir).*

tourner à droite (provoquée par la détection d'un obstacle devant). En effet, lorsque le robot détecte un obstacle devant, il commence à tourner à droite ; à ce moment il détecte aussi un obstacle à droite, et par conséquent commence à tourner à gauche, jusqu'à ce qu'il ne détecte plus d'obstacle à droite. Et ainsi de suite.

Une fois de plus, ce comportement est normal. Il met simplement en évidence l'incomplétude des règles de la page 155, qui n'envisagent pas toutes les situations. Une meilleure règle pour la situation **Évite près devant** serait : « on arrête le mouvement de translation du robot, et on demande une rotation lente à droite, sauf si l'on est déjà en train de tourner à gauche, auquel cas on continue à tourner à gauche ».

5.4.3 Analyse et améliorations

Ces premières expériences montrent que NESSY3L/ROB fonctionne, et donne déjà des résultats exploitables. Une analyse des problèmes rencontrés est maintenant nécessaire, avant tout nouveau développement, et nous donnons ci-dessous quelques premiers éléments. De plus, une analyse théorique est souhaitable, mais fait partie des perspectives de notre travail.

Prise en compte de Obstacles loins

Nous avons vu ci-dessus que le robot tournait en rond quand les obstacles étaient loins, et expliqué pourquoi (les unités de sortie du réseau versatile ont la même activation). Deux améliorations sont en fait possibles :

- ajouter une connexion supplémentaire dans le réseau versatile, permettant ainsi d'influencer la sortie vitesse angulaire nulle ; nous avons effectué cette correction et obtenu de bons résultats (le robot cesse de tourner en rond) ;
- détecter, au niveau symbolique, cette situation d'**indécision** (aucune décision ne l'emporte), et prendre une décision (ce qui revient à favoriser une des unités de sortie du réseau versatile) après un raisonnement symbolique.

Calcul de la vitesse linéaire

Nous avons vu ci-dessus le comportement de l'activation de l'unité Arrêt translation, qui dépendait fortement des réponses du capteur des obstacles situés à l'avant. Mais ce comportement dépend aussi de la manière dont nous utilisons l'activation pour calculer la vitesse linéaire. En effet, dans la méthode ci-dessus une accélération par défaut est également commandée : dans MOLUSC, la vitesse est demandée au moyen d'une commande *move*, qui exige aussi une accélération. Dans l'esprit du système flou de Reignier [1994b], nous utilisons l'accélération maximale permise (0.9 m/s^2), et cela explique le comportement assez brusque du robot.

Cette brusquerie peut aussi être grandement réduite en effectuant un pilotage plus « fin » du robot. Au lieu de donner seulement la commande *vl* et une accélération linéaire par défaut, comme ci-dessus, nous avons calculé aussi une accélération linéaire

à partir de l'activation de Arrêt translation. Notre idée consiste à fournir une accélération maximale quand Arrêt translation a une activation faible, une accélération nulle quand cette activation est moyenne, une accélération négative quand l'activation est forte. Là aussi nous avons obtenu de bons résultats, mais la définition des seuils relatifs à l'activation est délicate, et cette difficulté provient de l'absence de sens précis des activations du réseau versatile. Le risque est aussi de vouloir « faire dire » trop de choses à cette l'unité Arrêt translation.

Conflits entre tourner à gauche et tourner à droite

Nous avons vu dans l'expérience du couloir, ci-dessus, le problème dû à la succession de situations perceptives du type Coin droite et Évite près devant, qui conduisait le robot à percuter des obstacles. Ce problème s'explique par la priorité donnée arbitrairement à la décision de tourner à droite. De même dans le cas de la situation Bloqué[12]. Le niveau symbolique pourrait être exploité pour résoudre ce problème, mais une solution simple consiste à relier Évite près devant aux deux unités Rotation lente à gauche et Rotation lente à droite, avec les mêmes poids pour ne pas favoriser l'une des décisions.

Dans le cas où Évite près devant est détectée, deux cas sont alors possibles :

- l'une des unités Rotation lente à droite et Rotation lente à gauche est en train de gagner la compétition ; cette unité continue alors à gagner (alors qu'avec la première version de NESSY3L/ROB, Rotation lente à droite aurait systématiquement gagné) ;
- aucune de ces unités ne gagnait ; dans ce cas elles reçoivent la même influence des unités d'entrée, et leurs activations restent identiques ; il s'agit alors d'un nouvel état d'**indécision**, à traiter au niveau symbolique.

Inhibitions entre Arrêt translation et les unités concernant les rotations

Avec la première matrice d'inhibitions que nous avons choisie, une relation d'inhibition existe entre ces unités, et conduit à une compétition entre ces quatre décisions. Cela pose le problème du sens que l'on donne à cette compétition. En effet, si l'une des unités concernant les rotations tend à gagner, elle affaiblira Arrêt translation, ce qui revient à augmenter la vitesse du robot (et indirectement à contribuer au comportement brusque du robot). De plus, le comportement de l'opérateur téléopérant le robot est plutôt de ralentir durant les rotations. Aussi il serait plus judicieux d'adopter une dynamique favorisant le ralentissement du robot quand Rotation lente à droite ou Rotation lente à droite gagne. Or cela est possible car les travaux théoriques sur le réseau versatile n'interdisent pas aux d_{ij} d'être négatifs.

Nous avons réalisé quelques expériences qui ont permis de vérifier que l'on gardait bien la propriété de convergence, mais le choix d'une bonne valeur pour d_{12} et d_{13}

12. Mais nous ne pouvons pas traiter correctement cette situation avec NESSY3L/ROB, car il n'est pas possible, en l'état actuel, de commander une marche arrière.

s'est révélé délicat, et nous avons, pour le moment, préféré supprimer toutes les connexions entre Arrêt translation et les trois autres unités.

État actuel du réseau versatile

Ces diverses améliorations ont conduit à modifier le réseau versatile (fig. 5.25), et ses matrices des poids (fig. 5.26) et d'inhibitions :

$$(d_{ij}) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & d & d \\ 0 & d & 0 & d \\ 0 & d & d & 0 \end{pmatrix} \text{ où } d = 0.2$$

Problèmes non résolus

Cette nouvelle version pose encore quelques problèmes :

- dans un long couloir étroit, lorsque le robot démarre dans une direction d'angle faible avec l'axe du couloir, le robot finit par percuter un mur ; ce comportement est normal en l'état actuel de NESSY3L/ROB : tant que l'on perçoit à la fois un obstacle à gauche et un obstacle à droite, le robot va tout droit. Ce problème est dû au détecteur situé à l'avant, qui n'a pas été entraîné sur des exemples de telles situations ;
- le robot ne détecte pas les obstacles ponctuels (par exemple, un pied de chaise) : en effet, aucun obstacle de ce type n'a été présenté dans les divers jeux d'exemples ;
- le détecteur des obstacles à l'avant continue de poser d'autres problèmes, du type de ceux que nous avons évoqué plus haut. Nous nous proposons de recommencer son apprentissage, sans utiliser de fonction d'évaluation (et donc en demandant à l'opérateur d'indiquer les bonnes réponses).

Il paraît donc prioritaire d'améliorer les détecteurs du niveau neuronal, et il est notamment envisageable de remplacer les réseaux à rétropropagation que nous avons utilisé par des réseaux plus performants, du type RBF.

5.5 Discussion et perspectives

Notre architecture NESSY3L doit maintenant être examinée de deux points de vue :

- celui des SHNS,
- celui de l'application.

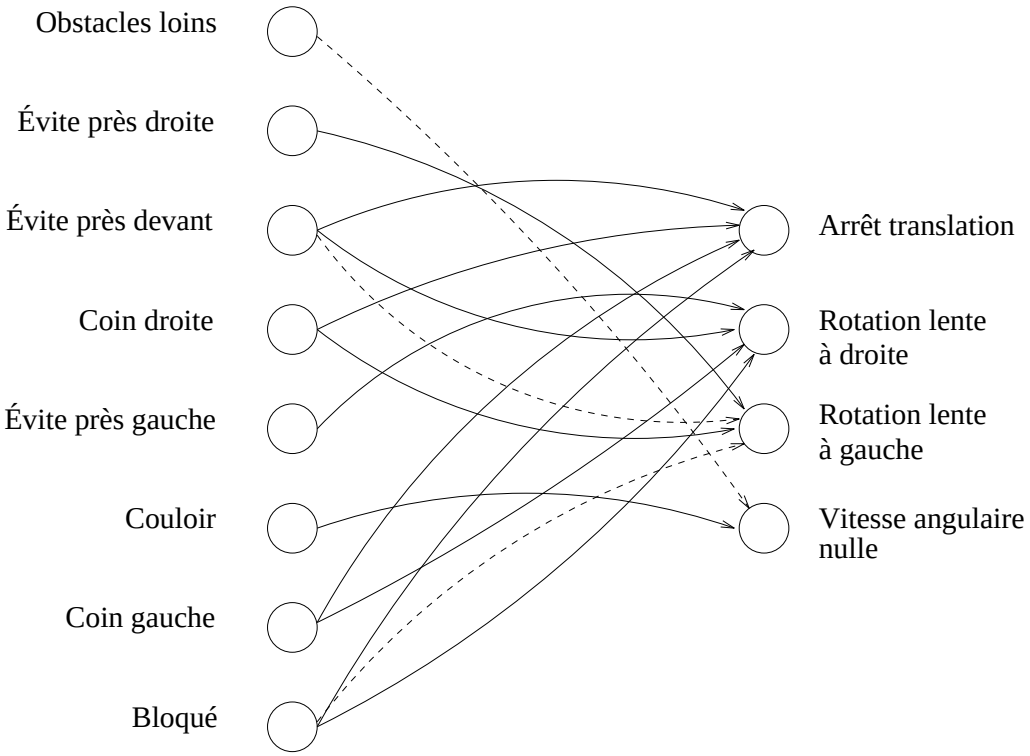


Fig. 5.25 - *Connexions entre la couche d'entrée et la couche de sortie de notre réseau versatile. Les nouvelles connexions sont indiquées en pointillés.*

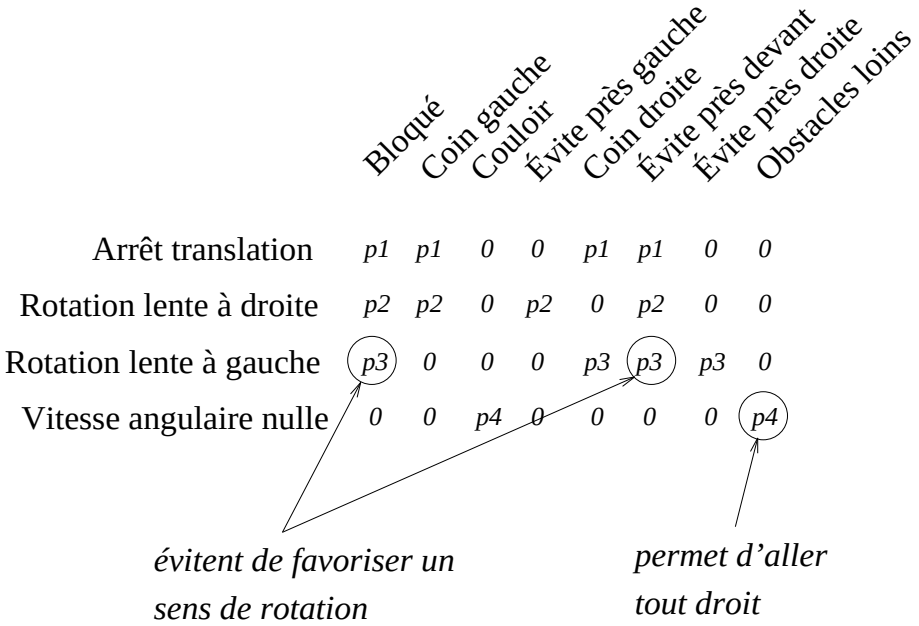


Fig. 5.26 - *Matrice des poids w_{ik} des connexions ci-dessus. Dans les expérimentations numériques nous avons pris :*

$$p1 = 0.25,\ p2 = 0.25,\ p3 = 0.25, p4 = 0.5$$

5.5.1 Point de vue des SHNS

Qu'apporte notre architecture du point de vue des SHNS ?

Premièrement, elle offre des **perspectives originales** :

1. **Prise en compte du temps** : actuellement, notre système de traitement des données capteurs prend des « photographies instantanées » de l'environnement du robot, et ne tient pas compte de l'évolution au cours du temps de cet environnement. Or il est possible de réaliser, avec des réseaux de neurones du type SRN, des détecteurs spatio-temporels (voir par exemple Bartell et Cottrell [1991] qui présentent une application simple : la détection de certaines caractéristiques du mouvement d'une boule de billard, telles que la boule va vite ou lentement, va à droite ou à gauche). De tels détecteurs[13] pourraient facilement s'insérer dans notre architecture :

   - soit pour remplacer les détecteurs actuels : d'une part, le détecteur des obstacles à l'avant prend en entrée la vitesse linéaire, ce qui n'est pas homogène avec le fonctionnement des autres détecteurs ; un détecteur spatio-temporel permettrait de ne prendre en entrée que les données provenant des capteurs. D'autre part, l'utilisation des détecteurs spatio-temporels sur les côtés du robot permettrait de détecter plus efficacement les obstacles en tenant compte implicitement de la vitesse angulaire ;
   - soit pour détecter de nouvelles situations perceptives, qui restent à identifier. Dans ce cas, il serait nécessaire d'ajouter de nouvelles règles au niveau neuro-symbolique.

   Cette perspective est intéressante, car il ne semble pas exister de SHNS utilisant de tels réseaux ; la prise en compte du temps dans les niveaux neuronal et neurosymbolique permet aussi d'envisager une évolution du niveau symbolique vers le raisonnement temporel (voir par exemple [Haton, Bouzid, et al. 1991]).

2. **Utilisation de réseaux à prototypes** : ces réseaux sont bien adaptés à la reconnaissance des formes [Azcarraga 1993], et pourraient permettre d'identifier des formes d'obstacles dès le niveau neuronal, permettant ainsi des prises de décisions beaucoup plus fines dans les niveaux supérieurs. L'ajout de nouvelles règles de décision serait bien sûr nécessaire.

3. **Développement de mécanismes inter-niveaux et évolution vers un couplage fort** : là également, plusieurs possibilités originales existent :

   - réglage, en cours d'utilisation, des divers paramètres du réseau versatile :

13. Nous avons réalisé une étude préliminaire(voir [Gauthier 1994]) avec des SRN, mais l'apprentissage s'est révélé difficile (temps de calcul énorme même pour des exemples simples, petit nombre d'exemples appris). Des travaux récents permettent d'expliquer ces difficultés : Meeden, McGraw, et Blank [1994] montrent notamment, de manière expérimentale, qu'il est utile de forcer les SRN à se comporter de manière prédictive. Il faut par conséquent contraindre un SRN à prédire en sortie son entrée à l'instant suivant, en plus de ce que l'on souhaite obtenir en sortie. Nous avons repris une partie des travaux de Gauthier [1994] et obtenu des résultats encourageants, comme la détection du sens de rotation du robot en utilisant les valeurs des capteurs d'un seul côté du robot.

c_i, d_{ij} et w_{ik} ; l'un des aspects originaux de ce réseau est en effet que ces coefficients ne sont pas fixés une fois pour toutes avant utilisation. Ainsi, le niveau symbolique pourrait comporter des règles fixant ces paramètres, en fonction du contexte (but recherché, situation observée, type du milieu exploré par le robot). Il s'agirait alors d'interactions de type **méta-traitement** ;

- ajout de nouvelles unités d'entrée au réseau versatile, activées quand certains faits symboliques sont déduits. Le niveau symbolique peut ainsi provoquer des changements de décisions dans le réseau versatile. L'intérêt de ce couplage est que les deux modules peuvent alors fonctionner en parallèle et de manière asynchrone ;
- mécanisme de sélection de connaissances, dans l'esprit de la liaison directe entre les deux modules de SYNHESYS : via une telle liaison, les niveaux inférieurs peuvent « pré-activer » des connaissances dans le niveau symbolique, soit des règles comme dans SYNHESYS, soit des « tâches » comme celles de SMECI (des paquets de règles). Ainsi on peut regrouper des règles par types de situation (urgence, navigation normale, etc.) et réagir très rapidement. De plus la bonne règle ou tâche pouvant être difficile à déterminer, l'apprentissage peut alors être d'un grand intérêt.

4. **Utilisation d'autres d'approches de l'intégration neurosymbolique** : parmi les diverses approches (chapitre 2), nous avons pour le moment utilisé seulement celle qui consiste à transformer une base de règles sous forme de réseau (un cas particulier de l'approche semi-hybride). De nombreux autres travaux, notamment purement connexionnistes, sont disponibles pour créer éventuellement d'autres niveaux intermédiaires.

De plus, une méthode d'apprentissage des poids d'un réseau versatile est en cours d'étude au LIFIA (travail de DEA sous la direction d'A. Labbi), et notre application permettra de tester cette méthode sur un problème relativement réaliste.

Deuxièmement, NESSY3L/ROB offre bien les avantages que nous attendions (section 5.1.2), et qui nous paraissent utiles pour la conception de SHNS facilement utilisables :

- la **simplicité des interfaces** : dans l'application présentée ci-dessus, les interfaces inter-niveaux servent uniquement, pour le moment, à copier des activations de neurones de sortie ;
- l'**absence de contrainte inter-niveaux** : actuellement, l'architecture est parfaitement modulaire, et aucun niveau ne fait d'hypothèse sur le fonctionnement des autres niveaux. Ainsi, chaque module peut évoluer indépendamment des autres, et notamment le module symbolique dans lequel il est possible d'utiliser des règles d'ordre 1, des objets structurés[14] Cette indépendance pourra toutefois être un peu réduite si l'on décide d'utiliser le module symbolique pour régler les paramètres du réseau versatile, mais seule une partie du module symbolique sera utilisée pour cette tâche ;

14. Ce qui n'était pas possible dans SYNHESYS à cause de la contrainte exercée par l'extraction de règles.

- une **bonne répartition des tâches** : en raison de cette absence de contraintes, chacun des composants garde ses bonnes caractéristiques, et est exploité pour les tâches pour lesquelles il est le plus compétent (par exemple, niveau neuronal / reconnaissance de formes, niveau symbolique / représentation de connaissances logiques et structurées) ;
- le **respect de nos trois contraintes** :
  - les réseaux de neurones utilisés dans le niveau purement neuronal sont des plus simples et des plus classiques, et respectent bien l'adéquation algorithme architecture. De plus de nombreux travaux visent à les réaliser sur circuit intégré. Le réseau versatile respecte aussi cette adéquation ;
  - en raison de la bonne répartition des tâches, on évite l'héritage des points faibles ;
  - l'extraction de règles ne paraît pas indispensable dans une telle architecture.

NESSY3L possède toutefois quelques inconvénients :

- le coût de la conception et de la réalisation d'un niveau intermédiaire, qui est important, et qui donc demande que l'on identifie des applications de cette architecture où ce coût soit faible par rapport aux avantages obtenus ;
- son caractère empirique à l'heure actuelle. Étant donné la variété de possibilités pour un niveau intermédiaire, il nous a semblé plus judicieux de commencer par développer une première application permettant de faire des choix (celui du réseau versatile par exemple), de voir si les avantages attendus a priori étaient bien observés. Il est maintenant indispensable d'envisager un travail plus théorique sur l'architecture.

5.5.2 Point de vue de l'application

Pour nous, cette application était principalement destinée à éclaircir et valider nos idées sur la conception d'un SHNS multi-niveaux, et à offrir des perspectives pour de futures recherches. Nous avons atteint ces objectifs et réalisé une architecture pouvant servir de support à l'expérimentation de plusieurs idées originales.

Par conséquent, notre but n'était pas d'apporter immédiatement une solution novatrice aux problèmes rencontrés depuis plusieurs années par les roboticiens.

Néanmoins NESSY3L/ROB apporte déjà des résultats suffisamment intéressants, compte tenu de sa simplicité, pour justifier la poursuite de son développement. Dans l'immédiat, il reste à ajouter des règles permettant de chercher à atteindre un but (un point précis de l'environnement), de manière à atteindre le niveau de compétences du système flou de Reignier [1994b]. À ce moment là il sera possible de comparer les performances de notre architecture par rapport à ce système flou qui est notre référence (car P. Reignier a lui-même effectué des comparaisons avec d'autres approches de la navigation réactive).

Il est clair que pour être objectives, ces comparaisons devront aussi se faire via des

expériences sur le robot réel, et non pas sur le robot simulé. L'un des intérêts de MOLUSC est que notre système devrait pouvoir prendre en compte les données capteurs réelles quasiment sans modifications.

En ce qui concerne les inconvénients de NESSY3L/ROB :

- NESSY3L/ROB hérite bien sûr du caractère empirique de NESSY3L;
- les activations des unités de sortie du réseau versatile n'ont pas de signification suffisamment précise d'un point de vue opératoire, et il est nécessaire de déterminer des heuristiques pour les transformer en commandes (vl, va). Ce problème d'absence de signification peut toutefois être résolu en utilisant des travaux formels visant à donner une sémantique à certains modèles connexionnistes. Un point de départ intéressant est FEL (*Fuzzy Evidential Logic*) de Sun [1991], qui est une extension de la théorie de Shoham. Toutefois dans le cas du réseau versatile, il est probable que les inter-connexions d'inhibitions poseront des problèmes théoriques. En effet elles introduisent des cycles dans le graphe orienté des connexions, alors que le travail de Sun repose sur des réseaux acycliques, beaucoup plus simples à manipuler ;
- le réseau versatile n'est sans doute pas indispensable dans l'état actuel de NESSY3L/ROB, et l'on obtiendrait peut être d'aussi bons résultats en associant directement à chaque situation perceptive un couple (vl, va). Mais son utilité apparaîtra dès que l'on cherchera à intégrer de nouvelles règles permettant d'atteindre un but, et aussi dès que l'on recherchera des couplages plus fins avec le niveau symbolique.

5.6 Conclusion

Dans ce chapitre, nous avons présenté une architecture originale de SHNS, ainsi qu'un domaine d'application approprié, la robotique mobile. Cette architecture consiste à ajouter au moins un niveau intermédiaire (dit « neurosymbolique ») entre un niveau symbolique et un niveau neuronal, ce qui a plusieurs avantages : simplicité des interfaces, relâchement des contraintes inter-niveaux, bonne répartition des tâches entre composants, respect de trois contraintes que nous nous étions imposées dans le chapitre précédent (respect de l'adéquation algorithme architecture, éviter l'héritage des points faibles, proposer des architectures où l'extraction de règles n'est pas indispensable).

Nous avons également présenté l'adaptation de l'architecture à l'application retenue (l'évitement d'obstacles), ainsi que sa réalisation, de manière assez détaillée, de telle sorte que ce travail puisse aisément être poursuivi par d'autres chercheurs.

À l'heure actuelle, la principale contribution de cette architecture est la variété de perspectives originales qu'elle offre pour les SHNS :

1. prise en compte du temps,
2. utilisation de réseaux à prototypes,

3. développement de mécanismes inter-niveaux et évolution vers un couplage fort,
4. utilisation des idées d'autres d'approches purement connexionnistes.

Ses inconvénients sont son caractère empirique actuel et le coût de la conception et de la réalisation d'un niveau intermédiaire.

Du point de vue de l'application NESSY3L/ROB en robotique, il est nécessaire de poursuivre le développement de notre prototype avant de pouvoir réaliser une évaluation précise, par comparaison avec d'autres travaux réalisés au LIFIA. Les inconvénients actuels de NESSY3L/ROB sont son caractère empirique hérité de NESSY3L, l'absence de signification opératoire des activations des unités du réseau versatile (nous indiquons toutefois une piste pour résoudre ce problème : FEL [Sun 1991]), le caractère non indispensable du réseau versatile dans l'état actuel de NESSY3L/ROB (dont l'intérêt apparaîtra toutefois dès que l'on passera à un deuxième prototype plus complet).

Accessoirement, cette application a permis d'illustrer l'intérêt du « neuro-calcul » [Hecht-Nielsen 1989], en tant qu'alternative au « calcul programmé » classique. En effet, durant la mise au point de notre architecture, le robot s'est fort mal comporté lors des premières expériences : il tournait beaucoup trop tard, et se collait aux murs. L'origine de ce comportement s'est révélé être un « bug » fort classique : un signe « - » avait été oublié dans le calcul de $vl(t)$. Cela illustre le caractère **fragile** de la partie symbolique de notre architecture, tandis que la partie neuronale est particulièrement **robuste**, puisque qu'elle donne des résultats exploitables alors que les réseaux utilisés ne montrent pas nécessairement des performances excellentes dans les tests (parfois seulement 80% des exemples ont été appris), et que les jeux d'exemples d'apprentissage comportaient toujours un (léger) bruit et souvent des erreurs.

Toutefois, cela ne nous conduit pas à proposer le neuro-calcul comme remplacement du calcul classique, des représentations de connaissances symboliques, des modélisations. Nous le proposons seulement comme **complément** pour certaines applications où son emploi se justifie pleinement.

Conclusion générale

Nous présentons ici la conclusion de notre travail, ainsi que ses perspectives.

Conclusion

Dans ce mémoire nous avons présenté notre travail de recherche sur les Systèmes Hybrides NeuroSymboliques (SHNS). Ce travail a porté sur des niveaux différents mais complémentaires :

- étude et application de SHNS existants ;
- réflexion de fond sur des problèmes relatifs aux réseaux neuronaux (comparaison avec des méthodes plus classiques, recherche d'une définition plus contraignante) et aux SHNS (classification des systèmes existants, répartition des tâches entre composants, rentabilité, directions de recherche) ;
- proposition d'un modèle original de SHNS, permettant d'explorer de nouvelles directions de recherche (prise en compte du temps, simplification des interfaces inter-composants, évolution vers le couplage fort) ;
- coopérations avec des utilisateurs potentiels de SHNS, à savoir des experts de différents domaines (micro-électronique, géographie, robotique), souvent non-informaticiens.

Concrètement, trois problèmes ont été étudiés dans ce mémoire.

Problème 1 : Présenter une vision structurée du vaste domaine de l'intégration neurosymbolique

Nous avons montré, essentiellement dans le chapitre 2, que le mot « hybride » avait des sens différents pour les auteurs concernés par l'intégration des bonnes caractéristiques des mondes symboliques et neuronaux. Il faut en effet distinguer entre **hybridité stricte**, où l'on exige que chacun des deux composants soit complet et fonctionne, et entre **hybridité large**, où l'on trouve notamment de nombreux travaux purement connexionnistes ainsi que des approches de type traduction. De façon

à structurer et éclaircir ce domaine, nous avons proposé une typologie, comprenant :

- l'approche hybride proprement dite, avec pour sous-classes couplages faible, étroit, fort ;
- l'approche semi-hybride, regroupant les approches de type traduction, comme l'extraction et la compilation de règles ;
- l'approche purement connexionniste, avec pour sous-classes les approches localiste, distribuée et combinée.

Nous avons illustré chacune de ces classes par des exemples représentatifs, en insistant, lorsque cela était possible, sur les systèmes appliqués à des problèmes réels. Il s'agit la plupart du temps de SHNS à couplage faible ou étroit, qui sous-traitent des tâches très précises (de type classification) à des réseaux de neurones artificiels.

Notre étude montre toutefois que dans le cas des SHNS les plus complexes, plusieurs questions restent posées : sont-ils « rentables » ? Résolvent-ils des problèmes fondamentaux ? Existe-t-il une répartition idéale des tâches entre les composants ? Ces questions n'ayant pas de réponses tranchées, nous proposons simplement quelques éléments de réflexion dans le chapitre 2.

Enfin, précisons qu'une telle étude faisait défaut, et que notre travail est une contribution directe au projet Esprit MIX (*Modular Integration of Symbolic and Connectionnist Processing in Knowledge-Based Systems*), qui se terminera en 1997.

Problème 2 : Évaluer et appliquer l'existant

Les questions ci-dessus nous ont conduit à examiner ce deuxième problème, de manière à développer les éléments de réflexions du chapitre 2. Dans les chapitres 3 et 4, nous étudions le SHNS SYNHESYS, l'un des plus complexes à l'heure actuelle, puis nous présentons deux applications à des problèmes réels.

SYNHESYS est un SHNS riche de mécanismes. En effet, il comporte :

- des modules symbolique (système expert d'ordre 0^+) et connexionniste (réseau à base de prototypes) ;
- trois schémas d'interactions entre ces deux modules ;
- des mécanismes de transfert de connaissances, qui peuvent être utilisés pour extraire et/ou raffiner des bases de règles ;
- un couplage direct entre les deux modules, permettant une amélioration des performances du chaînage arrière ;
- une version « floue ».

Par rapport à SYNHESYS lui-même, notre travail a consisté en :

- la programmation et l'expérimentation du mécanisme d'interdiction des intersections indésirables entre les régions d'activation des hyper-rectangles ;
- une évaluation quantitative du couplage direct entre les deux modules ;

– diverses expérimentations de SYNHESYS.

Par rapport aux applications de SYNHESYS, deux coopérations, l'une avec le CNET, l'autre avec le LAMA, nous ont permis de voir concrètement les problèmes que pose l'expérimentation d'un système d'IA. La coopération avec le CNET a montré que les problèmes industriels complexes relevaient plus des SHNS de type sous-traitance que des SHNS de type coopération comme SYNHESYS, qui sont plus adaptés à des problèmes restreints comme la classification de vecteurs. La coopération avec le LAMA a par contre porté sur l'expérimentation directe des mécanismes de SYNHESYS, et elle a montré que :

– la prise en compte de données hétérogènes (quantitatives et qualitatives) et bruitées est délicate,

– le problème posé pouvait se ramener à une classification en des classes non-disjointes, cas que SYNHESYS ne peut pas traiter,

– deux problèmes subsistent : la non-convergence de l'algorithme d'apprentissage, et des règles extraites difficilement exploitables dans le cas de données qualitatives. La procédure d'extraction de règles doit par conséquent être adaptée.

Les questions du chapitre 2 ont alors trouvé des réponses partielles :

– actuellement SYNHESYS est « rentable » surtout comme outil de recherche permettant de poser de nouveaux problèmes et de proposer de nouvelles directions de recherches ;

– SYNHESYS vise, entre autres, à résoudre des problèmes fondamentaux comme la modélisation hybride de connaissances, ou d'un point de vue plus technique, comme l'extraction et le raffinement de règles ;

– SYNHESYS propose une répartition particulière de tâches entre ses deux modules ; cette répartition est bien adaptée aux applications visées (classification de vecteurs) mais n'est pas « idéale » car moins adaptée dans d'autres cas.

En partie grâce à ces applications, nous avons aussi soulevé trois problèmes nouveaux :

1. le caractère neuronal des réseaux à base d'hyper-rectangles, et surtout des perfectionnements qui leur ont été apportés, est quelque peu discutable, ce qui a plusieurs conséquences :

   – cela entretient une certaine confusion chez les utilisateurs potentiels de SHNS, comme les géographes, car ils perdent de vue les spécificités des réseaux neuronaux par rapport aux méthodes classiques ;

   – SYNHESYS se trouve dans un état intermédiaire entre un SHNS et un système d'IA symbolique, et certaines simplifications sont nécessaires pour utiliser SYNHESYS comme base de nouvelles recherches sur l'intégration neurosymbolique.

   Nous pensons qu'il serait profitable pour les SHNS d'imposer certaines contraintes sur la définition d'un réseau de neurones artificiels. Nous avons proposé comme

point de départ le **respect de l'adéquation algorithme architecture**, une des caractéristiques essentielles des réseaux de neurones artificiels ;

2. le choix d'une architecture de SHNS peut avoir des conséquences importantes comme l'héritage de points faibles (chaque module impose certaines de ses moins bonnes caractéristiques au SHNS) ;
3. pour les SHNS, l'extraction de règles est l'une des directions de recherches importantes à l'heure actuelle, mais nous avons montré que son coût est très important par rapport à ses avantages. Aussi nous proposons de mettre l'accent sur de nouvelles architectures où l'extraction de règles serait moins nécessaire (par exemple en utilisant des réseaux de neurones artificiels suffisamment bien connus mathématiquement pour ne pas nécessiter une explication sous forme de règles).

Problème 3 : Proposer un modèle offrant de nouvelles perspectives

Dans le chapitre 5, nous nous appuyons sur toutes les idées ci-dessus pour proposer une nouvelle architecture de SHNS et une application appropriée.

Le principe de base de l'architecture NESSY3L (*NEuroSymbolic SYstem with 3 Levels*) est d'introduire un (ou plusieurs dans de futures NESSYnL) niveau intermédiaire entre les niveaux neuronaux et symboliques, de manière à réduire le coût des interfaces et à rendre moins nécessaires les extractions massives de règles. La difficulté se trouve alors dans la définition de niveaux intermédiaires appropriés ; nous proposons comme point de départ l'étude de niveaux « neurosymboliques », fondés sur les approches purement connexionnistes de l'intégration neurosymbolique. En particulier, les travaux consistant à coder des bases de règles sous forme de réseaux de neurones artificiels peuvent trouver un emploi ici, car le système obtenu est justement intermédiaire entre niveaux symboliques et neuronaux.

Les inconvénients de NESSY3L sont son caractère empirique actuel et le coût de la conception et de la réalisation d'un niveau intermédiaire.

En ce qui concerne l'application, la robotique mobile est un domaine d'application « naturel » pour les SHNS :

- les réseaux de neurones artificiels, notamment ceux pouvant être réalisés sous forme de circuits intégrés[15], sont bien adaptés (classification de données sensorielles, etc.) ;
- des architectures complexes combinent déjà des niveaux symboliques et numériques, et les SHNS peuvent s'intégrer dans ces architectures ;
- les roboticiens sont préparés à l'utilisation des SHNS, dans le sens où ils connaissent bien les systèmes symboliques et les réseaux de neurones artificiels, leurs points forts et leurs points faibles ; ceci n'était pas le cas dans les

15. D'où l'importance du respect de l'adéquation algorithme architecture.

autres applications que nous avons réalisées, et a été source de difficultés et de confusions.

Nous avons adapté notre architecture NESSY3L au problème particulier de l'évitement d'obstacles, en navigation réactive, en utilisant comme niveau intermédiaire un réseau dit versatile qui est adapté au problème et est bien caractérisé mathématiquement. Puis nous avons programmé cette architecture et nous l'avons intégrée dans un simulateur de robot, obtenant ainsi le système NESSY3L/ROB. Nous avons alors réalisé plusieurs expériences, proposé des améliorations, et montré que les avantages attendus de cette architecture étaient bien observés. Du point de vue de l'application, il reste par contre un important travail d'évaluation et de comparaison. Les inconvénients actuels de NESSY3L/ROB sont son caractère empirique hérité de NESSY3L, l'absence de signification des activations des unités du réseau versatile (nous indiquons toutefois une piste pour résoudre ce problème : FEL [Sun 1991]), le caractère non indispensable du réseau versatile dans l'état actuel de NESSY3L/ROB (dont l'intérêt apparaîtra toutefois dès que l'on passera à un deuxième prototype plus complet).

Ayant réalisé notre propre SHNS, nous identifions mieux maintenant les raisons du manque actuel d'évaluation des SHNS complexes ou originaux. Il faut en effet respecter un équilibre, difficile à obtenir, entre la phase d'évaluation et les autres étapes de la conception d'un SHNS qui sont coûteuses en temps :

- acquisition de connaissances et de compétences relatives aux composants,
- recherche et réalisation de mécanismes de couplage,
- recherche d'une application appropriée,
- comparaisons avec les autres approches pour le problème choisi, soit purement neuronales, soit purement symboliques, soit autres.

Par rapport au projet MIX, cette architecture NESSY3L est intéressante du point de vue de ses motivations, de ses principes et de ses perspectives ; elle peut éventuellement servir de support pour l'une des trois applications, celle fournie par le partenaire industriel allemand Kratzer, qui nous paraît présenter quelques similitudes avec la nôtre. D'autre part, le projet MIX co-organise, en 1995, un workshop IJCAI dont l'un des objectifs est d'étudier les possibilités et l'intérêt d'un rapprochement entre les approches hybrides proprement dites et les approches purement connexionnistes de l'intégration neurosymbolique. Aussi notre architecture est une contribution directe à cet effort de recherche actuel.

Perspectives

À l'issue de ce travail, trois types de perspectives peuvent être envisagées :

- perspectives pour SYNHESYS, que nous avons cherché à appliquer et à évaluer, dans une certaine mesure,
- perspectives pour NESSY3L et NESSY3L/ROB,

– perspectives pour les SHNS en général.

Perspectives pour SYNHESYS

Notre étude de SYNHESYS a conduit A. Giacometti à réaliser une nouvelle version qui est en cours d'application sur le problème de la déprise agricole (une base de règles et une base d'exemples ont déjà été fournies par la Chambre d'Agriculture de l'Isère), ce qui illustre le processus itératif de transfert de technologie dont nous avons parlé ci-dessus. Cette nouvelle version revient notamment au cadre neuronal initial via certaines simplifications de l'algorithme d'apprentissage des réseaux à base de prototypes.

D'autre part, SYNHESYS contient un grand nombre de mécanismes, qui peuvent être exploités dans d'autres SHNS :

– certains mécanismes sont actuellement réutilisés dans la conception d'un autre système hybride combinant RBC et réseaux à base de prototypes [Malek et Labbi 1995],

– NESSY3L pourra également reprendre certains mécanismes (réseaux à base de prototypes, couplage fort).

Perspectives pour NESSY3L et NESSY3L/ROB

En ce qui concerne NESSY3L, les principales perspectives sont la recherche de nouveaux niveaux intermédiaires et une étude plus théorique.

Par contre les perspectives pour NESSY3L/ROB sont plus concrètes, et sont détaillées dans la section 5.5. Nous les rappelons ici :

1. prise en compte du temps,
2. utilisation de réseaux à prototypes pour l'identification des obstacles,
3. développement de mécanismes inter-niveaux et évolution vers un couplage fort,
4. utilisation d'autres d'approches de l'intégration neurosymbolique.

Les perspectives à court terme sont les suivantes :

1. ajout de règles permettant la recherche d'un but, pour réaliser des comparaisons avec un autre système réalisé au LIFIA,
2. évaluation plus poussée de l'architecture,
3. expérimentation d'une méthode d'apprentissage pour les réseaux versatile, en cours d'étude.

Perspectives pour les SHNS

Il reste à évoquer les perspectives pour les SHNS.

Nous pensons qu'en leur état actuel, les SHNS permettent de résoudre certaines des difficultés rencontrées par les systèmes symboliques et les réseaux de neurones artificiels pris séparément, mais qu'ils ont un coût important dès que l'on aborde d'autres couplages que les couplages faibles. Aussi, une première perspective pour les SHNS est d'identifier des niches, au sens de niche écologique, particulières, où ce coût se justifie totalement. Notre analogie de la voiture hybride (chapitre 1) permet d'illustrer cette opinion. En effet la voiture hybride ne remplace toujours pas les voitures classiques, bien qu'elle soit à l'étude depuis plusieurs années. Aussi il est douteux que la voiture hybride soit une solution idéale pour les problèmes des automobiles. Par contre, des niches ont été trouvées pour des véhicules hybrides : il existe depuis longtemps des locomotives hybrides, à moteur diesel et moteur électrique. Le moteur diesel sert à entraîner le moteur électrique via une génératrice de courant, permettant ainsi à la locomotive de circuler sur des lignes non électrifiées. L'intérêt de ce couplage est d'éviter le système de transmissions complexes que demanderait le moteur diesel seul (cf. les systèmes de bielles des machines à vapeur).

Pour dépasser ce stade actuel, il faudra trouver de nouvelles idées. Des réseaux de neurones artificiels plus plausibles biologiquement, les études menées en Sciences Cognitives sur la notion de symbole, sur l'ancrage des symboles, peuvent ouvrir des perspectives plus larges pour les SHNS.

Annexe A

Modifications apportées à SYNHESYS

Nous décrivons ici quelques modifications que nous avons apportées à SYNHESYS, après avoir remarqué certains problèmes, comme nous l'avons indiqué page 102. Ces modifications concernent uniquement l'apprentissage relatif aux hyper-rectangles, et ont pour but d'interdire les intersections entre régions d'activations d'hyper-rectangles associés à des décisions différentes. L'existence de telles intersections n'est normalement pas possible d'après l'algorithme donné par A. Giacometti, mais nous avons été conduit à corriger la programmation de cet algorithme. L'algorithme d'extraction de règles suppose que de telles intersections n'existent pas, si bien que leur présence conduit à l'extraction de règles contradictoires. Nous nous en sommes aperçu en testant SYNHESYS sur la base d'exemples des **iris**, très utilisée en apprentissage automatique [Weiss et Kulikowski 1991], et dans laquelle quatre variables quantitatives comme la longueur des pétales décrivent des iris qui appartiennent à trois classes : Iris Setosa, Iris Versicolour et Iris Virginica (tab. A.1).

A.1 Phase d'accomodation

Nous avons tout d'abord remarqué que le mécanisme d'accomodation, qui normalement ne devrait rien faire (voir la présentation de l'algorithme d'apprentissage section 3.3.2), dilatait parfois certains hyper-rectangles, comme sur la figure A.2. Ces dilatations conduisent également à des intersections indésirables. Nous avons donc simplement supprimé le corps de la fonction `accomodation_hyper` dans le fichier `hyper.c`.

A.2 Phase d'absorption

L'une des particularités de l'algorithme d'apprentissage est la phase d'absorption, qui précède la phase d'assimilation. En effet, lorsqu'un point doit être assimilé, on

```
REGLE R1

  Si s_l in [4.9,7.3] et s_w in [1.0,4.5] et
     p_l in [2.5,4.5] et p_w in [0.0,3.4]
  Alors X5

FINREGLE R1

[ ... ]

REGLE R5

  Si X5
  Alors virginica

FINREGLE R5

[ ... ]

REGLE R9

  Si X5
  Alors versicolour

FINREGLE R9

[ ... ]
```

Tab. A.1 - *Exemple d'extraction de règles contradictoires avec SYNHESYS et la base des exemples des iris. Nous n'avons laissé que les règles incriminées. Dans ces règles,* s_l *représente* sepal length, s_w *représente* sepal width, p_l *représente* petal length, p_w *représente* petal width, *et tous sont exprimés en centimètres. Ce comportement peut être reproduit avec les paramètres standards de SYNHESYS et les fichiers* `orsier/hybride/donnees/rgl/iris.rgl` *et* `orsier/hybride/donnees/bdd/iris.bdd`.

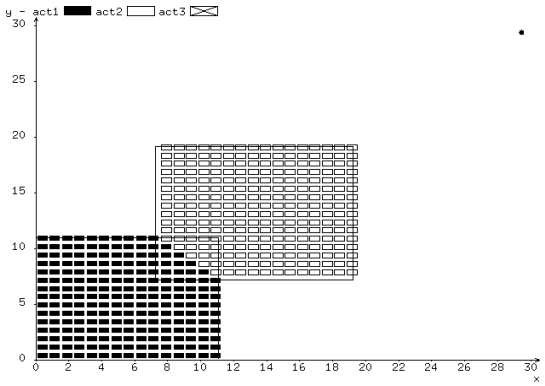


Fig. A.1 - *Un exemple montrant que le mécanisme d'assimilation actuel conduit à créer des intersections non désirées entre régions d'influences. Dans cet exemple, le point* $(5,5)$ *de classe* $act1$ *puis le point* $(13,13)$ *de classe* $act2$ *ont été présentés au réseau, qui a créé les deux rectangles ci-dessus, de dimensions arbitraires constantes. Les tous petits rectangles indiquent les régions d'activations – noter la compétition entre les 2 neurones dans l'intersection entre les régions.*

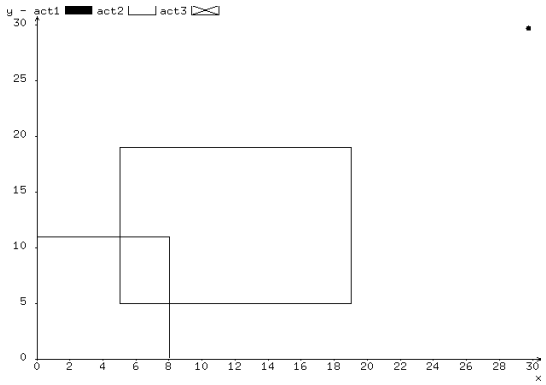


Fig. A.2 - *Un exemple montrant le comportement anormal de l'accommodation dans SYNHESYS. Le point* $(8,8)$ *de classe* $act2$ *a été présenté au réseau de la figure de gauche, ce qui a conduit à une différentiation du rectangle en bas à gauche, puis le point a été présenté à nouveau, et le réseau a donné la bonne réponse, ce qui le place en situation d'accommodation, où il ne devrait en principe rien faire ([Giacometti 1992], p. 173). Mais le rectangle en haut à droite a été un peu dilaté. Il s'agit vraisemblablement d'un reste du programme de Broissiat [1991], simplifié par A. Giacometti.*

tente d'absorber ce point en agrandissant un hyper-rectangle voisin, associé à la même décision. En cas de réussite, cela évite de créer un hyper-rectangle supplémentaire ; en cas d'échec, l'assimilation proprement dite est effectuée. Cette phase d'absorption conduit également à des intersections indésirables. Il était possible de la corriger de manière à interdire ces intersections, mais cette procédure est relativement complexe (il faut notamment choisir entre plusieurs hyper-rectangles candidats et aussi choisir quelle face de l'hyper-rectangle retenu doit être déplacée) et son apport n'a pas été précisément mesuré. Aussi, au moins dans un premier temps, nous avons préféré interdire les absorptions. Cela a conduit à modifier la fonction `test_assimilation_hyper` dans le fichier `hyper.c`.

A.3 Phase d'assimilation

La phase d'assimilation conduit également à des intersections indésirables (voir figure A.1). A. Giacometti avait indiqué une méthode pour corriger cette phase, dans le cas des hyper-sphères : il s'agissait de calculer le rayon de la nouvelle sphère de telle sorte que celle-ci n'empiète pas sur ses voisines. Mais une telle méthode n'a pas été programmée dans le cas des hyper-rectangles, si bien que la procédure d'assimilation crée un hyper-rectangle de taille constante (figure A.1).

Nous avons modifié la phase d'assimilation de manière à créer un hyper-rectangle qui n'empiète pas sur ses voisins. En fait nous avons réalisé deux variantes : dans la première, l'hyper-rectangle ne peut empiéter sur aucun de ses voisins ; dans la deuxième, il peut empiéter sur ceux de ses voisins qui sont associés à la même décision (de telles intersections sont prises en charge par la procédure d'extraction de régles, et devraient permettre un meilleur recouvrement de l'espace d'entrée. Par meilleur, on entend avec moins d'hyper-rectangles, et avec des hyper-rectangles de volume plus important). Notre algorithme, présenté ci-dessous, consiste à sélectionner les hyper-rectangles proches du point à assimiler : un hyper-rectangle est considéré comme proche s'il possède une intersection avec un voisinage du point, ce voisinage étant défini par un hyper-rectangle centré sur le point et de longueur constante dans toutes les dimensions. On calcule ensuite pour chaque dimension la longueur maximum que l'on peut donner à l'hyper-rectangle (voir figure A.3). Ces deux variantes sont programmées dans la fonction `bo_cherche_distances_methode_2` dans le fichier `hyper.c` ; leur programmmation a également conduit à la modification de la fonction `assimilation_hyper` dans le même fichier.

Notations relatives à l'algorithme :

- Si h est un hyper-rectangle,
- $h[i]$ désigne le segment projection de h sur l'axe de la dimension i,
- $h[i].a$ désigne l'extrémité gauche de ce segment,
- $h[i[.b$ désigne son extrémité droite.

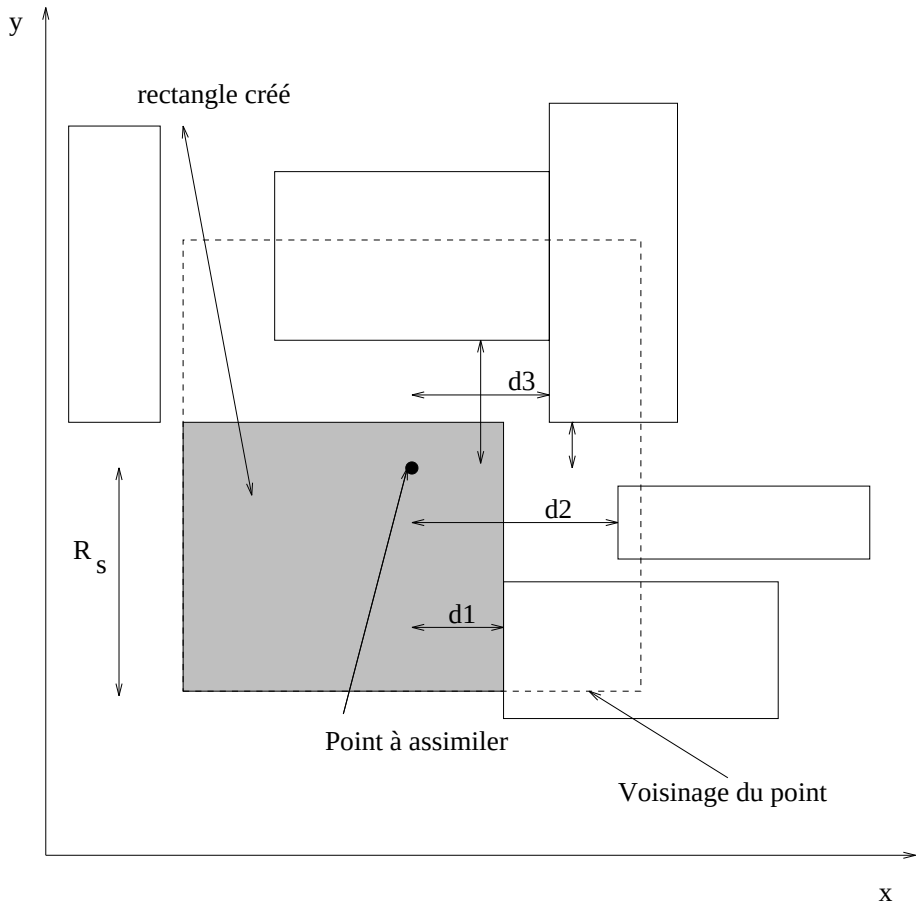


Fig. A.3 - *Exemple de fonctionnement de l'algorithme d'assimilation en dimension 2. Seuls sont pris en compte les rectangles ayant une intersection avec le voisinage du point. La longueur maximum que peut prendre le rectangle créé, sur la dimension x et à droite du point, est $MIN(d1, d2, d3)$. De même à gauche, et de même pour toutes les dimensions.*

Algorithme 1 *Assimilation d'un exemple par un hyper-rectangle*

Soit *les données :*
- $exemple$ *l'exemple à apprendre, en fait un couple (situation, action),*
- $situation$ *la situation associée à l'exemple,*
- $action$ *l'action associée,*
- R_s *la moitié de la longueur maximum permise pour une face d'hyper-rectangle créé par assimilation,*
- l_hyp *la liste de tous les hyper-rectangles existants avant cette assimilation,*
- n *la dimension de l'espace d'entrée,*
- $variante \in \{1, 2\}$ *la variante de l'algorithme choisie.*

Créer $hyper$ *l'hyper-rectangle centré sur* $situation$ *et dont toutes les faces ont pour longueur* $2 * R_s$.
Soit $l\_candidats$ *la liste de tous les éléments de* l_hyp *ayant une intersection avec* $hyper$.
Si $variante = 2$
Alors *retirer de* $l\_candidats$ *tous les hyper-rectangles ayant pour action associée* $action$
FinSi
Pour $i \in [1, ..., n]$ **faire**
  $min\_droite \leftarrow min\_gauche \leftarrow R_s$
  Pour $h \in l\_candidats$ **faire**
    Si $h[i].a > situation[i]$
    Alors $min\_droite \leftarrow MIN(min\_droite, h[i].a - situation[i])$
    Sinon
      Si $h[i].b < situation[i]$
      Alors $min\_gauche \leftarrow MIN(min\_gauche, situation[i] - h[i].b)$
      FinSi
    FinSi
  Fin Pour
  $hyper[i].a \leftarrow situation[i] - min\_gauche$
  $hyper[i].b \leftarrow situation[i] - min\_droite$
Fin Pour

Résultat : $hyper$

Nous présentons maintenant les résultats obtenus en appliquant ces différentes modifications (tab. A.2). Nous prenons comme base de comparaison la version de SYNHESYS développée par A. Giacometti (appelée « version initiale » dans le tableau). Les trois autres méthodes testées sont la version initiale sans absorption ni accomodation, et les deux variantes de notre algorithme d'assimilation.

L'analyse qui suit n'est bien sûr valable que pour le jeu d'exemples des IRIS, et ne préjuge pas du comportement des algorithmes sur d'autres bases de cas. Le tableau A.2 montre d'une part que la suppression de l'absorption conduit à un plus grand nombre de règles extraites (27 au lieu de 14). Le tableau montre d'autre part que

	version initiale (VI)	VI sans absorption ni accommodation	Variante 1	Variante 2
Après 10 présentations de la base des IRIS				
hyper-rectangles	4 (15 tués)	5 (17 tués)	36 (16 tués)	105 (1 tué)
exemples appris	70.7 %	71.3 %	99.3 %	96.0 %
Règles extraites	14	27	51	15
Après 50 présentations de la base des IRIS				
hyper-rectangles	4 (95 tués)	5 (97 tués)	36 (96 tués)	**505** (1 tué)
exemples appris	70.7 %	71.3 %	99.3 %	96.0 %
Règles extraites	14	27	51	16

Tab. A.2 - *Résultats obtenus sur la base d'exemples des IRIS. Les exemples sont présentés séquentiellement (càd ils ne sont pas mélangés aléatoirement). Ces résultats concernent le nombre d'hyper-rectangles, le pourcentage d'exemples bien appris après l'apprentissage, le nombre de règles extraites. Voir le texte pour l'analyse des résultats.*

nos deux variantes d'assimilation conduisent à un nombre d'hyper-rectangles plus important, mais avec un meilleur pourcentage d'exemples appris, tandis que la 2ème variante produit le même nombre de règles extraites que la version initiale.

La partie inférieure du tableau montre que les 40 présentations suivantes de la base d'exemples n'ont pas d'influence sur les résultats obtenus. Toutefois, le nombre curieusement important d'hyper-rectangles obtenus par la 2ème variante nous a conduit à étudier de plus près l'apprentissage effectué par le réseau ; nous avons constaté que l'algorithme d'apprentissage ne conduit pas à une stabilisation du réseau, dans le sens où il répète les mêmes créations d'hyper-rectangles suivies des mêmes différentiations. La trace ci-dessous, produite par la 2ème variante, illustre bien ce phénomène. Elle montre que les opérations effectuées sont cycliques. Il n'y a donc pas convergence de l'algorithme. Ceci n'est pas propre à notre deuxième variante. Toutes les autres méthodes souffrent du même problème, mais produisent des hyper-rectangles qui sont immédiatement « tués » par les différenciations suivantes. Au contraire la 2ème variante produit des hyper-rectangles suffisamment grands pour ne pas être tués.

Ce phénomène est propre à ce jeu d'exemples, et sur d'autres nous ne l'avons pas contaté (la 2ème variante produit alors bien le résultat attendu : moins d'hyper-rectangles sont créés). Le phénomène persiste si l'on présente aléatoirement les hyper-rectangles, mais le nombre d'hyper-rectangles est moins important.

Trace produite par la 2ème variante :

```
[...]
point [0.150 0.490 0.250 0.630] — HR 298 — volume de l'hyper cree : 0.003345
tentative de diff. sur HR 294 : DIFF sur face 2
+tentative de diff. sur HR 293 : DIFF sur face 2
+tentative de diff. sur HR 292 : DIFF sur face 2
+tentative de diff. sur HR 295 : DIFF sur face 2
+tentative de diff. sur HR 291 : DIFF sur face 2
```

```
point [0.160 0.510 0.270 0.600] --- HR 299 --- volume de l'hyper cree : 0.003517
tentative de diff. sur HR 296 : DIFF sur face 2
point [0.170 0.450 0.250 0.490] --- HR 300 --- volume de l'hyper cree : 0.009296
tentative de diff. sur HR 299 : DIFF sur face 2
point [0.200 0.510 0.320 0.650] --- HR 301 --- volume de l'hyper cree : 0.006413
tentative de diff. sur HR 299 : DIFF sur face 2
point [0.150 0.500 0.220 0.600] --- HR 302 --- volume de l'hyper cree : 0.008002
tentative de diff. sur HR 299 : DIFF sur face 2
+tentative de diff. sur HR 298 : DIFF sur face 2
point [0.180 0.490 0.270 0.630] --- HR 303 --- volume de l'hyper cree : 0.007041
tentative de diff. sur HR 299 : DIFF sur face 2
+tentative de diff. sur HR 298 : DIFF sur face 2
+tentative de diff. sur HR 297 : DIFF sur face 2
point [0.180 0.480 0.280 0.620] --- HR 304 --- volume de l'hyper cree : 0.007361
point [0.230 0.510 0.310 0.690] --- HR 305 --- volume de l'hyper cree : 0.012813
tentative de diff. sur HR 300 : DIFF sur face 2
point [0.130 0.450 0.280 0.570] --- HR 306 --- volume de l'hyper cree : 0.007700
tentative de diff. sur HR 304 : DIFF sur face 2
point [0.180 0.480 0.320 0.590] --- HR 307 --- volume de l'hyper cree : 0.003169
tentative de diff. sur HR 304 : DIFF sur face 2
+tentative de diff. sur HR 303 : DIFF sur face 2
```

- **point [0.150 0.490 0.250 0.630] — HR 308 — volume de l'hyper cree : 0.003345**

```
tentative de diff. sur HR 304 : DIFF sur face 2
+tentative de diff. sur HR 303 : DIFF sur face 2
+tentative de diff. sur HR 302 : DIFF sur face 2
+tentative de diff. sur HR 305 : DIFF sur face 2
+tentative de diff. sur HR 301 : DIFF sur face 2
point [0.160 0.510 0.270 0.600] --- HR 309 --- volume de l'hyper cree : 0.003517
tentative de diff. sur HR 306 : DIFF sur face 2
point [0.170 0.450 0.250 0.490] --- HR 310 --- volume de l'hyper cree : 0.009296
tentative de diff. sur HR 309 : DIFF sur face 2
point [0.200 0.510 0.320 0.650] --- HR 311 --- volume de l'hyper cree : 0.006413
tentative de diff. sur HR 309 : DIFF sur face 2
point [0.150 0.500 0.220 0.600] --- HR 312 --- volume de l'hyper cree : 0.008002
tentative de diff. sur HR 309 : DIFF sur face 2
+tentative de diff. sur HR 308 : DIFF sur face 2
point [0.180 0.490 0.270 0.630] --- HR 313 --- volume de l'hyper cree : 0.007041
tentative de diff. sur HR 309 : DIFF sur face 2
+tentative de diff. sur HR 308 : DIFF sur face 2
+tentative de diff. sur HR 307 : DIFF sur face 2


[...]
```

En conclusion, nos travaux sur SYNHESYS ont permis de mettre en évidence la non-convergence de l'algorithme d'apprentissage ; il est possible de trouver des critères d'arrêt (par exemple, avec la variante 1 de l'assimilation, on peut s'arrêter quand les hyper-rectangles créés sont de volume nul) mais il serait souhaitable de réaliser une étude mathématique du comportement de l'algorithme.

Annexe B

Algorithmes et modules divers

Notations :

- $\odot$: concaténation de chaînes de caractères,
- $e \oplus l$: e est inséré en tête de la liste l.

B.1 Génération automatique de règles

B.1.1 Algorithme utilisé

Algorithme 2 *Génération automatique de règles*

Étape 1 : *génération des règles de premier niveau*
Soit *les variables :*
–$dimensions$ *une liste de dimensions* $(dim^1 ... dim^n)$, *par exemple* $("a"\ "b"\ "c")$,
–$nbipd$ *le nombre d'intervalles souhaités par dimension, par exemple 3,*
–l *la longueur d'un intervalle, par exemple 1.*

Pour $dim \in dimensions$ **faire**
Pour $i \in [1, ..., nbpid]$ **faire**
générer la règle « $si\ dim\ \in [i \times l,\ (i+1) \times l[\ alors\ dim_i$ »
Fin Pour
Fin Pour

Étape 2 : *génération des règles de deuxième niveau*
Soit *les variables :*
–$decisions_{courantes}$ *la liste* $(dim^1_1 ... dim^1_{nbpid})$,
–$decisions_{nouvelles}$ *la liste vide* $()$.

Pour $j \in [2, ..., n]$ **faire**
Pour $dec \in decisions_{courantes}$ *et* $k \in [1, ..., nbpid]$ **faire**
générer la règle « $si\ dec\ et\ dim_k^j\ alors\ dec \odot dim_k^j$ »
$decisions_{nouvelles} \leftarrow dec \odot dim_k^j \ \oplus decisions_{nouvelles}$
Fin Pour
$decisions_{courantes} \leftarrow decisions_{nouvelles}$
Fin Pour

Étape 3 : *génération des règles de troisième niveau*
Soit *les variables :*
–$actions$ *une liste d'actions, au sens de SYNHESYS, par exemple* $(act_1\ act_2\ act_3)$,
–$action_{aleat}$ *une action choisie aléatoirement dans* $actions$.

Pour $dec \in decisions_{courantes}$ **faire**
choisir aléatoirement $action_{aleat}$
générer la règle « $si\ dec\ alors\ action_{aleat}$ »
Fin Pour

B.1.2 Exemple

Avec :

- $dimensions$ valant ("a" "b" "c" "d"),
- $nbipd$ valant 2,
- l valant 1,
- $actions$ valant $(act_1\ act_2\ act_3)$,

on obtient 52 règles, dont une partie est présentée par la figure B.1.

B.1.3 Calcul du nombre de règles générées

premier niveau : $n \times nbpid$,
deuxième niveau : $nbpid^2 + ... + nbpid^n = nbpid^2 \times \frac{1-nbpid^{n-1}}{1-nbpid}$,
troisième niveau : $nbpid^n$.

En additionnant, on obtient :

$$\boxed{nbpid[n + nbpid \times \frac{1-nbpid^{n-1}}{1-nbpid} + nbpid^{n-1}]}$$

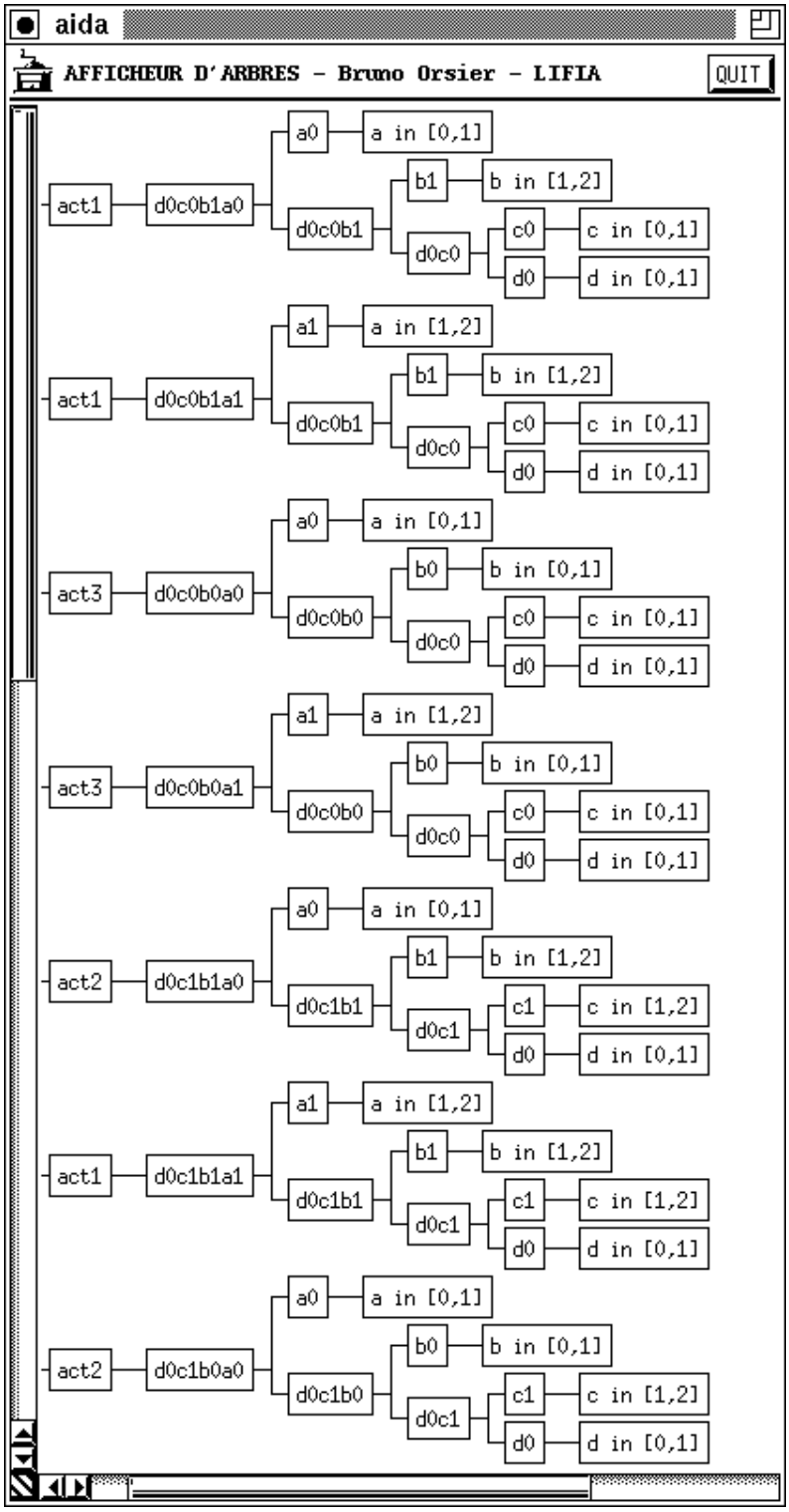


Fig. B.1 - *Une partie de l'arbre des règles générées. Les prémisses sont à droite, les conclusions à gauche.*

B.2 Module de gestion des réseaux versatiles

La programmation du système d'équations différentielles

$$\begin{aligned}\frac{dx_i}{dt} &= f_i(X,E) \\ &= -c_i.x_i + (1-x_i).(\sum_{k=1}^{m} w_{ik}.f_k(e_k)) - (1+x_i).(\sum_{j=1}^{n} d_{ij}.g_j(x_j)), \\ & \quad i = 1,...,n\end{aligned}$$

est assez directe. Aussi nous décrirons simplement l'interface fonctionnelle du module correspondant, écrit en langage C.

Ce module exporte les définitions suivantes :

```
#define NMAX   20                    /* nb max d'unites de sortie */
#define MMAX   30                    /* nb max d'unites d'entree */
```

ainsi que la structure d'un réseau, qui doit être connue du programme appelant pour fixer les entrées du réseau, et en exploiter les sorties (on peut éventuellement encapsuler cette structure si l'on fournit les fonctions correspondantes) :

```
struct reseau_Jabbi {                /* un reseau est compos'e de : */
  float e[MMAX];                     /* un vecteur d'entr'ee */
  float x[NMAX];                     /* un vecteur de sortie */
  float c[NMAX];                     /* un vecteur de coefficients de fatigue */
  float d[NMAX][NMAX];  /* une matrice d'inhibitions mutuelles */
  float w[NMAX][MMAX];  /* une matrice de poids positifs ou nuls */
  int n;                             /* dimension vecteur de sortie */
  int m;                             /* dimension vecteur d'entr'ee */
};
```

Seules deux fonctions sont exportées. D'une part, la fonction

```
struct reseau_Jabbi * creer_reseau(int n, int m,
                                   float fatigue[NMAX],
                                   float inhib[NMAX][NMAX],
                                   float poids[NMAX][MMAX]);
```

qui rend un réseau à partir de ses paramètres. Cette fonction vérifie que la condition de convergence est bien vérifiée (si elle ne l'est pas, on sort brutalement du programme – une gestion plus fine des erreurs serait nécessaire).

D'autre part, la fonction (qui est une procédure en fait)

```
void etat_suivant (struct reseau_Jabbi * r, float delta_t )
```

qui permet de faire progresser l'état du réseau d'un pas de temps.

B.3 Module de gestion des tubes Unix

Ce module, inspiré de [Braquelaire 1991], exporte quatre fonctions.

int *creer_tube* (**char** **nom_du_reseau*)

permet de lancer un programme qui chargera en mémoire le réseau en question, puis attendra des informations sur son entrée standard. La fonction rend un nombre que le programme appelant utilisera pour communiquer avec ce réseau.

void *fermer_tube*(**int** *num*)

ferme le tube de communication avec le réseau repéré par le paramètre.

int *tubeputs*(**int** *num*, **char** **s*)

envoie une chaîne de caractères dans le tube du réseau repéré par le paramètre. Retourne le nombre de caractères écrits.

int *tubegets*(**int** *num*, **char** **s*, **int** *l*)

lit une chaîne de caractères dans le tube du réseau repéré par le paramètre. Retourne le nombre de caractères lus, qui peut être 0.

Un exemple d'utilisation :

```
int a,b ;
char resa[80] ;
char resb[80],ligne[80] ;

a = creer_tube("/home/meteore/orsier/SNNS/Carbot4/detect-obst-devant.net") ;
b = creer_tube("/home/meteore/orsier/SNNS/Carbot4/dog-cc.net") ;


tubegets(a, resa, 80) ;
while (strcmp("PRET",resa) != 0) {                                    10
   printf("resultats sur a : %s\n",resa) ;
   tubegets(a, resa, 80) ;
}

tubegets(b, resb, 80) ;
while (strcmp("PRET",resb) != 0) {
   printf("resultats sur b : %s\n",resb) ;
   tubegets(b, resb, 80) ;
}
                                                                       20

gets(ligne) ;
while (strcmp("FIN",ligne) !=0 ) {
   tubeputs(a,ligne) ;
```

```
    tubegets(a,resa,80) ;
     printf("resultats sur a : %s\n",resa) ;
    gets(ligne);
    tubeputs(b,ligne) ;
    tubegets(b,resb,80) ;
     printf("resultats sur b: %s\n",resb) ;
    gets(ligne);

}


tubeputs(a,"FIN") ;
tubeputs(b,"FIN") ;


fermer_tube(a) ;
fermer_tube(b) ;
```

Annexe C

Coopération avec le LAMA

C.1 Les données

Le fichier initialement fourni par l'IGA a tout d'abord été prétraité par nos soins, puisqu'il fallait identifier et supprimer les « clashs » pour pouvoir utiliser SYNHESYS. Il est ici présenté sous cette forme prétraitée[1]. Le fichier comporte 7 colonnes dont `No` qui est le numéro de la parcelle (ou de l'individu, en terme d'analyse de données) étudiée, et `MOD` qui est une variable supplémentaire non utilisée. Comment lire ce fichier prétraité? Certaines lignes contiennent `Ligne testee` : cela indique que la ligne qui suit immédiatement est une ligne du fichier original dont les variables d'entrée ont été extraites (elles sont indiquées entre parenthèses) et ont servi a repérer toutes les lignes suivantes du fichier original ayant les mêmes entrées. Ces lignes identiques du point de vue des entrées ont été rassemblées en dessous de la ligne « testée ». Ceci permet au lecteur de facilement repérer les « clashs », en regardant la troisième colonne qui est la variable décision. Chaque fois qu'elle contient des valeurs différentes pour une suite de parcelles encadrées par deux indications `Ligne testee`, il y a un clash. On voit donc qu'ils sont très nombreux, ce qui explique les difficultés rencontrés par SYNHESYS lors de l'apprentissage de ces données.

Dans un deuxième temps, après avoir fait ce tri, nous avons cherché à supprimer les clashs. Pour cela nous avons pris dans chaque paquet la décision qui intervenait le plus fréquemment, et nous avons indiqué la parcelle choisie par une étoile (*) en fin de ligne. Dans certains cas, un tel choix n'a pas été possible car plusieurs décisions apparaissaient avec la même fréquence. Nous avons alors purement et simplement ignoré les parcelles du paquet, ce qui est indiqué ci-dessous par un point d'interrogation (?).

Cette procédure est parfaitement arbitraire et nous en sommes conscients. Toutefois, à ce stade, notre objectif n'est plus d'obtenir des résultats thématiques à l'aide de SYNHESYS, qui n'est finalement pas approprié pour ce type de données, mais d'obtenir un fichier de données permettant d'extraire des règles pour finir d'identi-

1. Toutefois, le fichier complet, qui s'appelle `resultats-veron-apres-sup` comporte environ 8 pages, et nous nous sommes contenté de donner seulement les deux premières.

fier les problèmes que pose l'application de SYNHESYS à des données qualitatives nominales. Il est clair que les règles extraites auront peu d'intérêt du point de vue thématique car nous avons profondément modifié le jeu de données.

En récupérant chaque ligne contenant une étoile, nous avons construit un fichier de 51 exemples, sans clash, qui a permis d'extraire les règles présentées ci-dessous.

No	MOD	EVU	SAU	AGS	REG	FVL	
Ligne testee :			(1	1	1	2)	
1	3	2	1	1	1	2	*
2	2	2	1	1	1	2	
Ligne testee :			(1	1	1	1)	
3	4	4	1	1	1	1	
90	2	3	1	1	1	1	
301	4	2	1	1	1	1	*
302	4	2	1	1	1	1	
324	3	2	1	1	1	1	
328	5	4	1	1	1	1	
Ligne testee :			(1	3	3	3)	
4	6	5	1	3	3	3	
10	2	2	1	3	3	3	*
24	2	2	1	3	3	3	
154	2	2	1	3	3	3	
155	3	2	1	3	3	3	
178	3	2	1	3	3	3	
245	3	3	1	3	3	3	
Ligne testee :			(1	3	1	3)	?
5	2	2	1	3	1	3	
93	2	3	1	3	1	3	
340	2	2	1	3	1	3	
341	3	3	1	3	1	3	
Ligne testee :			(3	1	1	3)	
6	2	2	3	1	1	3	*
7	2	2	3	1	1	3	
8	3	2	3	1	1	3	
101	2	2	3	1	1	3	
108	3	3	3	1	1	3	
111	4	2	3	1	1	3	
115	4	3	3	1	1	3	
117	4	2	3	1	1	3	
118	4	2	3	1	1	3	
119	4	2	3	1	1	3	
120	4	2	3	1	1	3	
121	4	2	3	1	1	3	
122	4	2	3	1	1	3	
123	4	2	3	1	1	3	
124	2	2	3	1	1	3	
125	4	2	3	1	1	3	
126	4	4	3	1	1	3	
127	4	2	3	1	1	3	
128	4	2	3	1	1	3	
129	4	2	3	1	1	3	
130	4	2	3	1	1	3	
131	4	2	3	1	1	3	
132	4	2	3	1	1	3	
133	4	2	3	1	1	3	

```
134     4       2       3       1       1       3
135     4       2       3       1       1       3
136     4       2       3       1       1       3
144     4       2       3       1       1       3
145     4       2       3       1       1       3
146     4       2       3       1       1       3
147     4       2       3       1       1       3
148     4       2       3       1       1       3
149     4       2       3       1       1       3
150     4       2       3       1       1       3
208     5       3       3       1       1       3
213     5       3       3       1       1       3
216     4       3       3       1       1       3
219     3       1       3       1       1       3
227     4       3       3       1       1       3
241     4       2       3       1       1       3
362     3       2       3       1       1       3
365     2       2       3       1       1       3
Ligne testee :          (3      1       1       2)
9       3       2       3       1       1       2       *
354     4       2       3       1       1       2
363     5       3       3       1       1       2
364     2       2       3       1       1       2
Ligne testee :          (1      3       3       1)
11      2       3       1       3       3       1       *
25      2       2       1       3       3       1
54      2       3       1       3       3       1
153     3       2       1       3       3       1
193     4       3       1       3       3       1
196     2       2       1       3       3       1
246     3       3       1       3       3       1
247     3       1       1       3       3       1
Ligne testee :          (3      2       1       3)
12      5       3       3       2       1       3
13      3       2       3       2       1       3       *
14      2       2       3       2       1       3
15      2       2       3       2       1       3
30      2       3       3       2       1       3
31      2       3       3       2       1       3
46      3       2       3       2       1       3
205     4       3       3       2       1       3
223     4       2       3       2       1       3
243     4       2       3       2       1       3
253     6       3       3       2       1       3
254     2       2       3       2       1       3
257     3       2       3       2       1       3
258     4       2       3       2       1       3
259     4       3       3       2       1       3
260     2       2       3       2       1       3
261     4       2       3       2       1       3
262     5       3       3       2       1       3
263     2       3       3       2       1       3
264     4       2       3       2       1       3
266     4       3       3       2       1       3
268     4       3       3       2       1       3
269     4       3       3       2       1       3
```

```
270     4       3       3       2       1       3
273     2       2       3       2       1       3
277     2       2       3       2       1       3
281     2       2       3       2       1       3
282     2       2       3       2       1       3
283     2       2       3       2       1       3
285     2       2       3       2       1       3
291     2       1       3       2       1       3
292     4       3       3       2       1       3
293     3       2       3       2       1       3   [ ... ]
```

C.2 Règles brutes extraites par SYNHESYS

À partir du fichier de 51 exemples, très facilement appris (taux de bonnes réponses de 99 %, 24 prototypes), SYNHESYS a extrait 35 règles. Nous reproduisons les dix premières telles quelles ci-dessous.

```
REGLE R0

Si s1 in [0.0,0.2] et s2 in [0.0,0.2] et s3 in [0.8,1.0] et s4 in [0.0,0.2] et
a1 in [0.0,0.2] et a2 in [0.0,0.2] et a3 in [0.8,1.0] et a4 in [0.0,0.2] et a5
in [0.0,0.2] et r1 in [0.0,0.2] et r2 in [0.0,0.2] et r3 in [0.0,0.2] et r4 in
[0.8,1.0] et f1 in [0.0,0.2] et f2 in [0.0,0.2] et f3 in [0.8,1.0] et f4 in
[0.0,0.2] Alors depmo

REGLE R1

Si s1 in [0.0,0.2] et s2 in [0.0,0.2] et s3 in [0.0,0.2] et s4 in [0.8,1.0] et
a1 in [0.0,0.2] et a2 in [0.0,0.2] et a3 in [0.0,0.2] et a4 in [0.8,1.0] et a5
in [0.0,0.2] et r1 in [0.0,0.2] et r2 in [0.0,0.2] et r3 in [0.8,1.0] et r4 in
[0.0,0.2] et f1 in [0.0,0.2] et f2 in [0.0,0.2] et f3 in [0.8,1.0] et f4 in
[0.0,0.2] Alors depmo

REGLE R2

Si s1 in [0.0,0.8] et s2 in [0.0,0.2] et s3 in [0.0,0.2] et a1 in [0.0,0.2] et
a4 in [0.0,0.2] et r1 in [0.0,0.2] Alors depfa

REGLE R3

Si s1 in [0.0,0.2] et s2 in [0.8,1.0] et s3 in [0.0,0.2] et s4 in [0.0,0.2] et
a1 in [0.0,0.2] et a2 in [0.8,1.0] et a3 in [0.0,0.2] et a4 in [0.0,0.2] et a5
in [0.0,0.2] et r1 in [0.8,1.0] et r2 in [0.0,0.2] et r3 in [0.0,0.2] et r4 in
[0.0,0.2] et f1 in [0.0,0.2] et f2 in [0.8,1.0] et f3 in [0.0,0.2] et f4 in
[0.0,0.2] Alors depfa

REGLE R4

Si s1 in [0.0,0.2] et s2 in [0.0,0.2] et s3 in [0.8,1.0] et s4 in [0.0,0.2] et
a1 in [0.0,0.2] et a2 in [0.0,0.2] et a4 in [0.8,1.0] et a5 in [0.0,0.2] et r1
in [0.8,1.0] et r2 in [0.0,0.2] et r3 in [0.0,0.2] et r4 in [0.0,0.2] et f1 in
[0.0,0.2] et f2 in [0.8,1.0] et f3 in [0.0,0.2] et f4 in [0.0,0.2] Alors depfa

REGLE R5
```

```
Si s2 in [0.0,0.2] et s3 in [0.0,0.2] et s4 in [0.0,0.2] et a1 in [0.0,0.2] et
a2 in [0.8,1.0] et a4 in [0.0,0.2] et a5 in [0.0,0.2] et r1 in [0.0,0.2] et r2
in [0.8,1.0] et r3 in [0.0,0.2] et r4 in [0.0,0.2] et f1 in [0.0,0.2] et f4 in
[0.0,0.2] Alors depfa

REGLE R6

Si s2 in [0.0,0.2] et s3 in [0.0,0.2] et s4 in [0.0,0.2] et a1 in [0.0,0.2] et
a3 in [0.8,1.0] et a4 in [0.0,0.2] et a5 in [0.0,0.2] et r1 in [0.0,0.2] et r2
in [0.0,0.2] et r3 in [0.8,1.0] et r4 in [0.0,0.2] et f1 in [0.8,1.0] et f2 in
[0.0,0.2] et f3 in [0.0,0.2] et f4 in [0.0,0.2] Alors depfa

REGLE R7

  Si s1 in [0.2,1.0] et s2 in [0.2,1.0] et s4 in [0.0,0.2] et f4 in [0.0,0.2]
  Alors empr

REGLE R8

Si s3 in [0.0,0.2] et s4 in [0.0,0.2] et a2 in [0.0,0.2] et a5 in [0.0,0.2] et
r1 in [0.8,1.0] et r2 in [0.0,0.2] et r3 in [0.0,0.2] et r4 in [0.0,0.2] et f4
in [0.0,0.2] Alors empr

REGLE R9

Si s1 in [0.0,0.2] et s2 in [0.8,1.0] et s3 in [0.0,0.2] et s4 in [0.0,0.2] et
a3 in [0.0,0.2] et a4 in [0.0,0.2] et a5 in [0.0,0.2] et r3 in [0.0,0.2] et r4
in [0.0,0.2] et f1 in [0.0,0.2] et f2 in [0.0,0.2] Alors empr

REGLE R10

Si s1 in [0.0,0.2] et s2 in [0.0,0.2] et s3 in [0.8,1.0] et s4 in [0.0,0.2] et
a2 in [0.0,0.2] et a5 in [0.0,0.2] et r1 in [0.8,1.0] et r2 in [0.0,0.2] et r3
in [0.0,0.2] et r4 in [0.0,0.2] et f1 in [0.0,0.2] et f2 in [0.0,0.2] et f3 in
[0.8,1.0] et f4 in [0.0,0.2] Alors empr

[ ... ]
```

Annexe D

Coopération avec le CNET

Nous décrivons dans cette annexe l'état dans lequel se trouve actuellement la maquette symbolique réalisée pour le CNET, en indiquant comment les données sont stockées et quelles données le sont effectivement.

D.1 Un cadre pour stocker les données

Nous avons développé à l'aide du langage à objets de Smeci un cadre permettant de stocker les données provenant du SGBD BASIL et les informations saisies par les experts. Il s'agit d'un noyau permettant de commencer à travailler, et qui sera enrichi en fonction des besoins. Ce cadre est défini à l'aide de catégories Smeci (classes) qui sont instanciées lorsque l'on veut créer des objets précis.

BASIL contient nombre des données qui nous intéressent, mais sous une forme insuffisamment structurée et cohérente pour permettre leur traduction automatique vers un format utilisable par Smeci, surtout dans le cadre de la réalisation d'une maquette. BASIL répond à d'autres objectifs et modélise en fait les traitements plus par leurs zones géographiques dans la salle de traitement que par leur rôle.

D.1.1 Filières

Une filière (fig. D.1) est simplement vue comme la liste des lots qui en font partie. La saisie de filières est automatique.

Une seule filière est stockée : DEMO 0.7.

D.1.2 Lots

Un lot (fig. D.1) est composé de la filière à laquelle il appartient, ses plaques, ses dégroupages, sa date de lancement. La saisie de lots est automatique.

Huit lots ont été saisis : CA, CJ, CN, AU, AV, AX, AY, AZ.

D.1.3 Dégroupages

C'est l'un des objets les plus importants pour le moment. Il se compose du lot auquel il appartient, le traitement appliqué (une séquence de phases), et les plaques qui ont suivi ce traitement (fig. D.1). La saisie de dégroupages est manuelle.

Huit dégroupages, correspondant à quatre lots, ont été saisis.

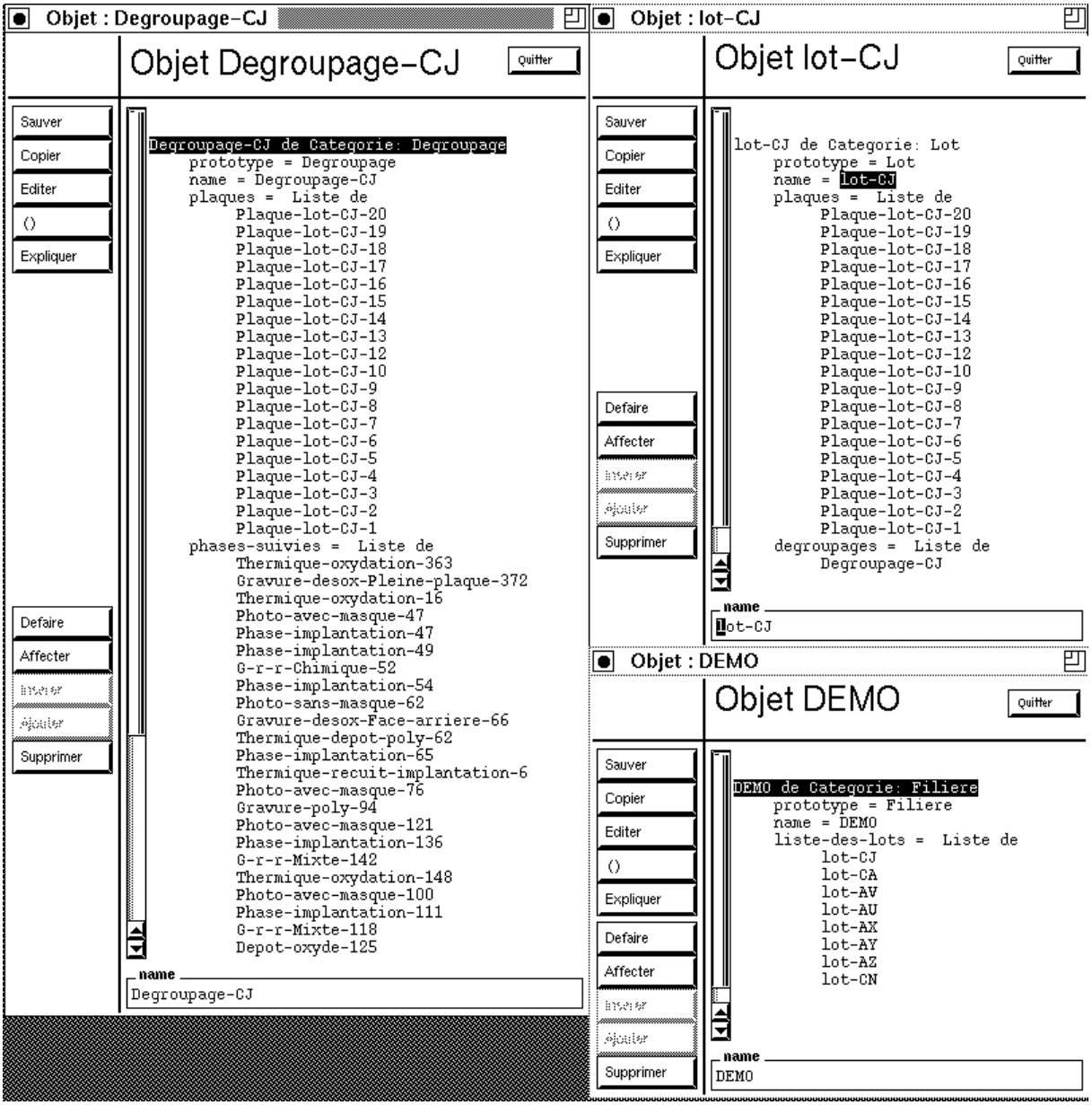


Fig. D.1 - *A gauche : un exemple de dégroupage, le «Degroupage-CJ». A droite en haut : un exemple de lot, le «lot-CJ», qui ne comporte qu'un seul dégroupage, le «Degroupage-CJ». A droite en bas : la filière modélisée dans notre étude, comportant en particulier le «lot-CJ».*

D.1.4 Plaques

Une plaque (fig. D.2) est définie par son dégroupage, son numéro dans ce dégroupage, ses valeurs de paramètres électriques. La saisie de plaques est automatique.

Les 138 plaques des 8 lots ont été saisies.

D.1.5 Paramètres électriques

Ils ne sont pas seulement caractérisés par une valeur, puisque on travaille sur les 21 mesures faites pour une plaque. On utilise donc la médiane des mesures et leur écart-type (fig. D.2). D'autre part, il arrive que les mesures ne puissent être effectuées pour certains sites. On doit donc connaître, au niveau du paramètre électrique, le nombre de mesures validées et le nombre de rejets par cause de rejet.

Tous les paramètres électriques des 138 plaques ont été saisis.

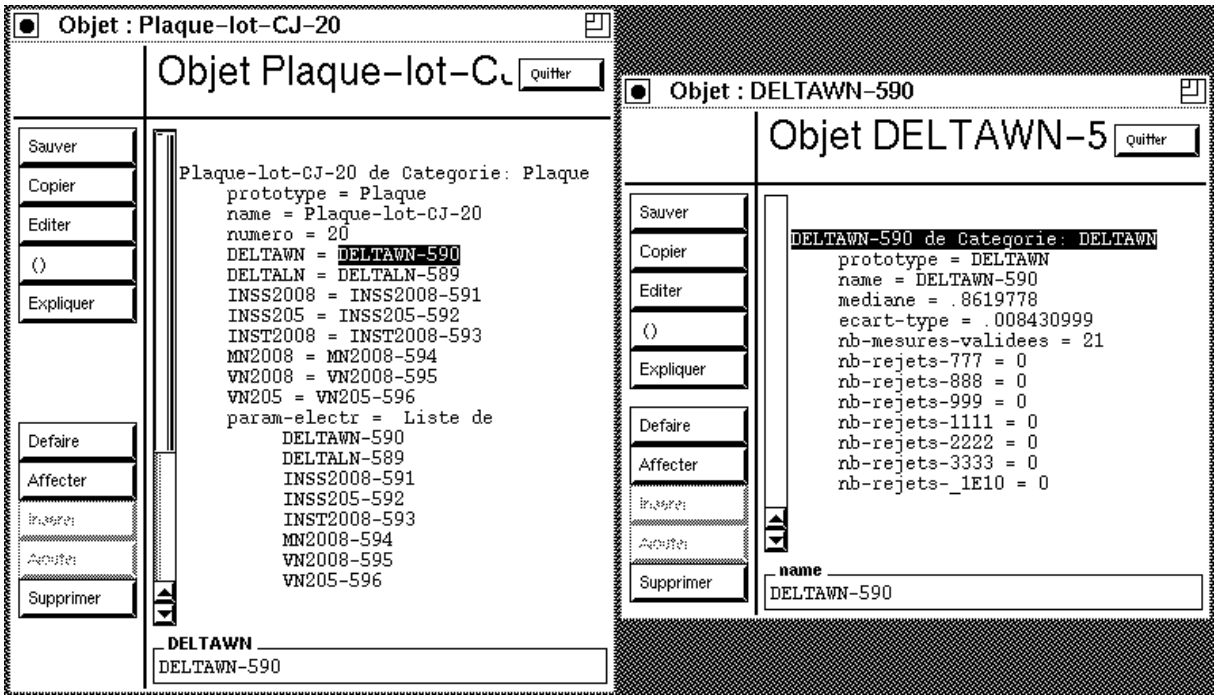


Fig. D.2 - *A gauche, un exemple de plaque, la plaque numéro 20 du lot CJ. A droite, son paramètre électrique DELTAWN, avec sa médiane et son écart-type ; les causes de rejets sont indiqués par les codes (777, 888, etc.) habituellement utilisés au CNET.*

D.1.6 Phases et étapes

Les traitements sont composés de **phases**, qui sont elles-mêmes des séquences d'**étapes**. Chaque étape est caractérisée par des paramètres qui décrivent précisément le traitement effectué, et notamment par une **recette**. Pour le moment, il n'a pas semblé

utile de décrire très en détail les recettes, qui seront donc simplement modélisée par leur **code**.

Une phase est caractérisée par son code et son nom BASIL, sa séquence d'étapes, et maintenant par une propriété spéciale qui précise son rôle dans la filière (par exemple, préciser qu'une implantation sert à implanter du « bore profond »). La saisie des phases est manuelle. La hiérarchie de catégories de phases que nous avons définie se compose de 48 sous-catégories.

Une « étape avec recette » est caractérisée par un code, un nom de recette, et une machine au sens de BASIL, tandis qu'une « étape sans recette » n'a pas de caractéristiques particulières. D'autres caractéristiques apparaissent en fait plus bas dans la hiérarchie : par exemple, une étape d'implantation se caractérise par les propriétés : énergie, impureté, dose, v-tilt, v-twist, nb-rotations, refroidissement. La saisie des étapes est manuelle. Notre hiérarchie d'étapes se compose actuellement de 36 sous-catégories.

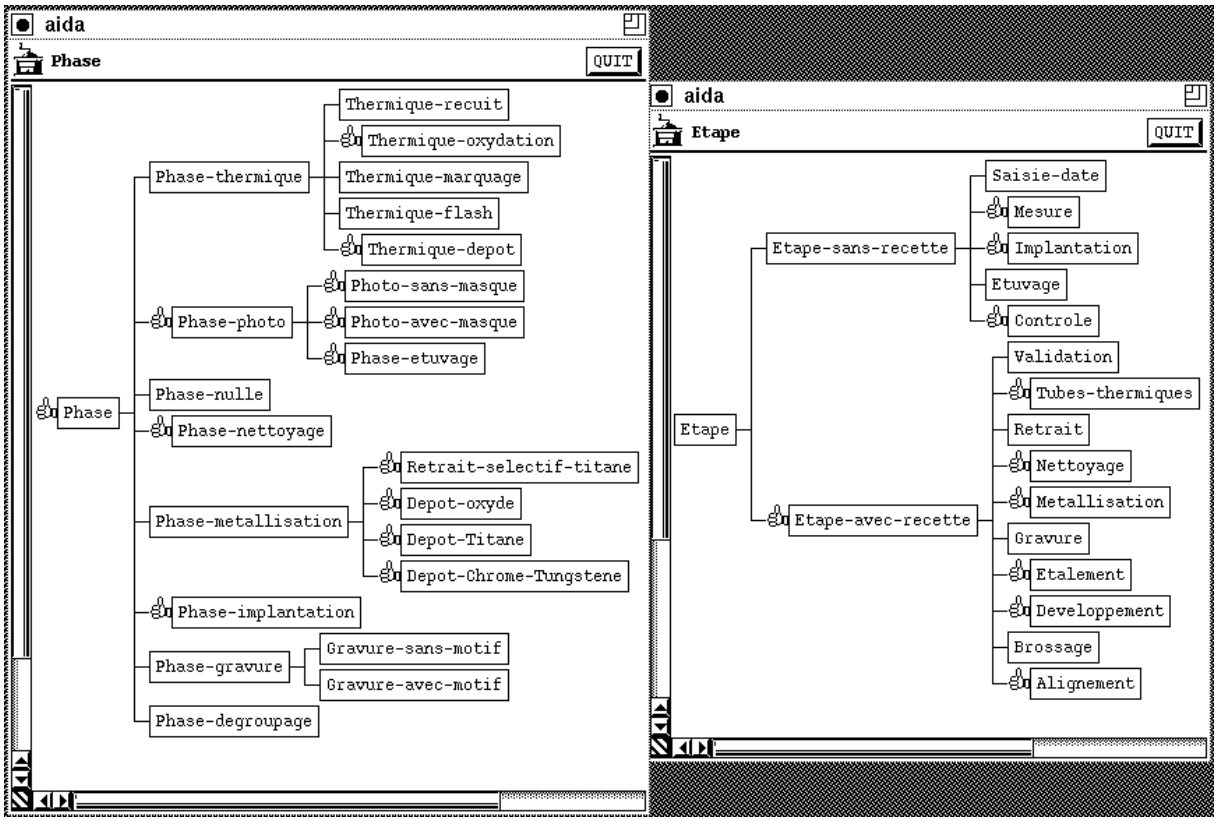


Fig. D.3 - *Les phases (à gauche) et étapes (à droite) les plus générales (les arbres affichés sont seulement de profondeur 2) modélisées dans notre système. Le petit symbole affiché à gauche de certaines classes indiquent celles qui ont été effectivement détaillées, les autres ne sont là que pour la clarté de la modélisation. Noter que ces arbres ne sont pas affichés avec l'interface standard de Smeci, mais avec une interface spécialement développée.*

D.2 Bilan

Le bilan de ce modèle de données est très satisfaisant. Il est très proche de BASIL, tout en étant mieux structuré et plus rigoureux. La saisie effective des données a donné lieu à peu de modifications du modèle initial, et ces modifications ont été simples grâce au langage à objets de Smeci. Ce modèle est d'autre part assez naturel pour les experts. Enfin, le système bénéficie des interfaces standards (éditeur d'objets, afficheurs de catégories, etc.) de Smeci, très pratiques dans cette phase de prototypage, et qui facilitent grandement les saisies. Les interfaces standards offrent de plus un puissant langage de requêtes.

Notons cependant que ce modèle présente des faiblesses car, tout comme BASIL, il ne met pas encore suffisamment l'accent sur le rôle des phases de traitement dans la filière. Mais l'élaboration rapide d'un modèle simple était indispensable pour démarrer.

La saisie du premier lot a demandé un travail important : huit heures de travail pour le seul dégroupage du lot CJ. Il s'agissait en fait d'un investissement vite rentabilisé, car on peut maintenant dupliquer de nombreux éléments de ce lot. Actuellement, la saisie d'un lot de quatre dégroupages demande une demi-heure de travail. Plusieurs outils ont d'autre part été développés pour faciliter cette saisie : fonctions de duplication récursive de phases, de copie de dégroupages, de création (resp. suppression) automatique d'étapes lors d'une création (resp. suppression) de phase. Une interface graphique a été développée (voir fig. 4.3).

Nous avons d'autre part commencé à l'enrichir, en ajoutant par exemple une propriété pour décrire le rôle dans la filière des phases.

Actuellement, notre maquette contient environ 1700 instances, toutes catégories confondues, qui occupent environ 200 K-octets. Ce modèle n'est donc pas très coûteux en mémoire, et on pourrait stocker facilement un nombre de lots de l'ordre de la centaine.

Références

Abdi, H. [1994]. *Les réseaux de neurones*. Presses Universitaires de Grenoble. 269p.

AISB [1992]. *Artificial Intelligence and Simulation of Behaviour, Special Issue on Hybrid Models - part II*, Number 79.

Alexandre, F. [1990, avril]. Une modélisation fonctionnelle du cortex : la colonne corticale – Aspects visuels et moteurs. Thèse de doctorat, Université de Nancy I, France.

Alpaydin, E. [1991, mai]. GAL : Networks that grow when they learn and shrink when they forget. Rapport technique TR-91-032, International Computer Science Institute, Berkeley.

Amy, B. [1991, mai]. Les systèmes hybrides en intelligence artificielle. Dans P. Bessiere, A. Guérin, et al. (éditeurs), *École de printemps NSI 91*, Villard de Lans, Isère, France.

Amy, B. et al. (éditeurs) [1992]. *Émergence dans les modèles de la cognition*, Paris. Actes des journées ENST - LIFIA, TELECOM Paris 92S003.

Amy, B. et al. (éditeurs) [1993, juin]. *Formation des symboles dans les modèles de la cognition*, Grenoble. Actes des journées ENST-LIFIA.

Azcarraga, A. P. [1993, mars]. Modèles neuronaux pour la classification incrémentale de formes visuelles. Thèse de doctorat, Institut National Polytechnique de Grenoble, France.

Azcarraga, A. P. et A. Giacometti [1991]. A prototype-based incremental neural network for classification tasks. Dans *Proc. of the 4 th International Conference on Neural Networks and their Applications*, Nîmes, France. EC2.

Baffes, P. T. et R. J. Mooney [1993]. Symbolic Revision of Theories with M-of-N Rules. Dans *Proc. of the 13 th IJCAI*, Chambéry, France, pp. 1135–1139.

Barnden, J. [1992a, sept-oct]. Book Review «Connectionist Symbol Processing - G. Hinton (ed.)». *Neural Networks 5*(5), 853–855.

Barnden, J. [1992b]. Connectionism, Generalization, and Propositional Attitudes : A Catalogue of Challenging issues. Dans Dinsmore [1992a], Chapitre 7, pp. 149–178.

Bartell, B. T. et G. W. Cottrell [1991]. A Model of Symbol Grounding in a Temporal Environment. Dans *IEEE-IJCNN1991*, Volume I, pp. 805–810.

Beauboucher, N. [1994, juin]. ANAÏS : raisonnement à partir de cas en résolution de problèmes. Thèse de doctorat, Université Paris VI, France.

Beaudot, W. H. [1994, décembre]. Le traitement neuronal de l'information dans la rétine des vertébrés : un creuset d'idées pour la vision artificielle. Thèse de doctorat, Institut National Polytechnique de Grenoble, France.

Becraft, W. R., P. L. Lee, et R. B. Newell [1991, août]. Integration of Neural Networks and Expert Systems for Process Fault Diagnosis. Dans *Proc. of the 12 th IJCAI*, Sidney, Australia, pp. 832–837.

Belaïd, A. et Y. Belaïd [1992]. *Reconnaissance des formes, Méthodes et Applications*. InterEditions. 429p.

Belala, Y. [1992, décembre]. Systèmes de production et unification connexionnistes. Thèse de doctorat, Université Paris-Sud, Centre d'Orsay, France.

Besnard, Y., V. Rialle, A. Vila, et al. [1991]. NEUROP: an expert system in electromyography based on a multilevel knowledge representation. Dans *Proc. of the 13th An. Int. Conf. IEEE Engineer. in Med. and Biol. Soc.*, Volume 13, pp. 1302–1303.

Bisson, G. [1993, avril]. Induction de Bases de Connaissances en Logique des Prédicats. Thèse de doctorat, Université Paris-Sud, Centre d'Orsay, France.

Bookman, L. A. et R. Sun (éditeurs) [1993]. *Connection Science, Special Issue: Architectures for Integrating Neural and Symbolic Processes*, Volume 5. Carfax Publishing Company.

Bouroche, J.-M. et G. Saporta [1987]. *L'analyse des données* (3ème ed.). Que sais-je? Presses Universitaires de France.

Bourret, P., J. Reggia, et M. Samuelides [1991]. *Réseaux neuronaux, une approche connexionniste de l'intelligence artificielle*. Teknea.

Brachman, R. J. et D. L. McGuinness [1988, juin]. Knowledge Representation, Connectionism, and Conceptual Retrieval. Dans *Proc. of SIGIR, 11th Int. Conf. on R. & D. in Information Retrieval*, Grenoble, pp. 161–174. PUG.

Braquelaire, J.-P. [1991]. *Méthodologie de la programmation en langage C, principes et applications*. Masson. 459p.

Broissiat, P. [1991]. Apprentissage par un réseau localiste et incrémental destiné à faire partie d'un système expert hybride. Mémoire de DEA, ENSIMAG-UJF, Grenoble.

Callan, J. P. [1991]. Adaptative Case-Based Reasoning. Dans *Proc. of the DARPA Case-Based Reasoning Workshop*, pp. 179–190. Morgan Kaufman.

Chaillot, M. [1993, septembre]. Une architecture de contrôle réactif pour la résolution coopérative de problèmes. Thèse de doctorat, Institut National Polytechnique de Grenoble, France.

Ciesielski, V., S. Hayes, et B. Kelly [1992, juillet]. Comparison of an expert system and a hybrid neural network/expert system for a respiratory monitoring problem. Dans Sun et Bookman [1992].

Cornu, T. [1992, octobre]. Machine Cellulaire Virtuelle, définition, implantation et exploitation. Thèse de doctorat, Université de Nancy I, France.

Cottrell, M. [1993, octobre]. Bases mathématiques des réseaux de neurones artificiels. COURS N° 1, Module 1, Neuro-Nîmes.

Cottrell, M., J. C. Fort, et G. Pagès [1994, avril]. Two or three things that we know about the Kohonen algorithm. Dans Verleysen [1994], pp. 235–244.

Cousin, E. et al. [1988, novembre]. Les machines à réduction et l'intelligence artificielle. *La Recherche 19*(204), 1348–1361.

Cremilleux, B. [1991, février]. Induction automatique : aspects théoriques, le système ARBRE, applications en médecine. Thèse de doctorat, Université Joseph Fourier, Grenoble, France.

Crucianu, M. [1994]. Représentations structurées dans les réseaux connexionnistes. Thèse de doctorat, Université Paris XI Orsay, France.

d'Alché Buc, F. [1993, décembre]. Modèles neuronaux et algorithmes constructifs pour l'apprentissage de règles de décision. Thèse de doctorat, Université Paris-Sud, Centre d'Orsay, France.

David, J.-M., J.-P. Krivine, et R. Simmons [1993]. Second Generation Expert Systems: A Step Forward in Knowledge Engineering. Dans J.-M. David, J.-P. Krivine, et R. Simmons (éditeurs), *Second Generation Expert Systems*. Springer Verlag.

Dedieu, E. et P. Bessière [1994, mai]. La caractérisation sensorielle des comportements. Dans Marchal et al. [1994], pp. 111–114.

Dinsmore, J. (éditeur) [1992a]. *The Symbolic and Connectionist Paradigms : Closing the Gap*. Lawrence Erlbaum Associates. 300p.

Dinsmore, J. [1992b]. Thunder in the Gap. Dans Dinsmore [1992a], Chapitre 1, pp. 1–24.

Dishaw, J. et J. Pan [1989, août]. AESOP : A Simulation-Based Knowledge System for CMOS Process Diagnosis. *IEEE Trans. on Semiconductor Manufacturing 2*(3).

Duval, J.-P. [1991]. Extraction de règles dans un système hybride et mise en correspondance de connaissances. Mémoire de DEA, ENSIMAG-UJF, Grenoble.

Elman, J. L. [1990]. Finding structure in time. *Cognitive science* (14), 179–211.

Fodor, J. A. et Z. W. Pylyshin [1988]. Connectionism and cognitive architecture: a critical analysis. *Cognition* (28), 2–71.

Fu, L. [1994]. *Neural Networks in Computer Intelligence*. McGraw-Hill. 460p.

Gauthier, E. [1994, septembre]. Étude des réseaux récurrents et application à la navigation réactive. Rapport de stage de Magistère Informatique, Université Joseph Fourier, Grenoble.

Giacometti, A. [1992, novembre]. Modèles hybrides de l'expertise. Thèse de doctorat, École Nationale Supérieure des Télécommunications, Paris, France.

Giacometti, A., I. Iordanova, B. Amy, A. Vila, et al. [1992]. A Hybrid Approach to Computer Aided Diagnosis in Electromyography. Dans *Proc. of the 14^{th} An. Int. Conf. IEEE Engineer. in Med. and Biol. Soc.*, Volume 14.

Giambiasi, N., R. Lbath, et C. Touzet [1989, novembre]. Une approche connexionniste pour calculer l'implication floue dans les systèmes à base de connaissances. Dans *Proc. of the 2^{nd} International Conference on Neural Networks and their Applications*, Nîmes, France. EC2.

Goller, C. [1994, août]. A Connectionist Control Component for the Theorem Prover SETHEO. Dans Hilario [1994], pp. 88–93.

González, J. C., J. R. Velasco, et al. [1994, décembre]. A Multiagent Architecture for Symbolic-Connectionist Integration. Document interne MIX/WP1/UPM/3.0, Universidad Politécnica de Madrid.

Gould, J. et R. Levinson [1991]. Method Integration for Experience-Based Learning. Rapport technique UCSC-CRL-91-27, Baskin Center for Computer Engineering and Information Sciences, University of California, Santa Cruz.

Grumbach, A. [1993]. Genèse du symbole artificiel. *Technique et science informatique 12*(3), 347–369.

Grumbach, A. [1994]. *Cognition artificielle*. Addison-Wesley. 232p.

Gutknecht, M. et R. Pfeifer [1990]. Experiments with a hybrid architecture : integrating expert systems with connectionist networks. Dans *Proc. of the 10 th International Conference on Artificial Intelligence, Expert Systems and Natural Language*, Avignon, France.

Handelman, D. A., S. H. Lane, et J. J. Gelfand [1992]. Robotic skill acquisition based on biological principles. Dans Kandel et Langholz [1992], Chapitre 14, pp. 302–327.

Harnad, S. [1990]. The Symbol Grounding Problem. *Physica D 42*(1-3), 335–346.

Hassoun, M. [1994, décembre]. Contrôle d'exécution des mouvements d'un robot mobile : application à l'assistance à la conduite automobile. Thèse de doctorat, Institut National Polytechnique de Grenoble, France.

Haton, J.-P., N. Bouzid, et al. [1991]. *Le raisonnement en intelligence artificielle – Modèles, techniques et architectures pour les systèmes à base de connaissances*. InterEditions.

Hecht-Nielsen, R. [1989]. *Neurocomputing*. Addison-Wesley.

Hendler, J. [1989]. Problem Solving and Reasoning : A Connectionist Perspective. Dans R. Pfeifer et al. (éditeurs), *Connectionism in Perspective*, pp. 229–243. Elsevier Science Publishers B. V. (North Holland).

Hilario, M. [1993]. MIX : Modular Integration of Connectionist and Symbolic Processing in Knowledge-Based Systems. Proposal for Basic Research Project, EEC, n° 09119.

Hilario, M. (éditeur) [1994, août]. *ECAI94 Workshop «Combining Symbolic and Connectionist Processing»*, Amsterdam.

Hrycej, T. [1992]. *Modular Learning in Neural Networks*. John Wiley & Sons. 235p.

ILOG [1992]. *SMECI Version 1.65, Manuel de Référence*. Gentilly, France, ILOG.

Jacobsen, H.-A., I. Iordanova, et A. Giacometti [1994, juillet]. Extraction de règles floues dans un système expert hybride. Dans *IPMU, Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, Paris.

Jodouin, J.-F. [1993, mars]. Réseaux de neurones et traitement du langage naturel. Thèse de doctorat, Université Paris XI Orsay, France.

Josselin, D. [1995]. Systèmes d'induction et d'informations géographiques : application à la déprise agricole. Thèse de doctorat, Université Joseph Fourier, Grenoble, France. version préliminaire.

Josselin, D. et B. Orsier [1993]. SIG et réseaux neuronaux, application à la déprise agricole en moyenne montagne. Dans *Séminaire CASSINI*, Grenoble. Institut de Géographie Alpine. à paraître.

Jutten, C. [1991, mai]. Réseaux neuronaux, apprentissage sans superviseur et incrémental. Dans P. Bessiere, A. Guérin, et al. (éditeurs), *École de printemps NSI 91*, Villard de Lans, Isère, France.

Jutten, C. et J. Hérault [1994]. La mémoire des réseaux neuronaux. *La recherche, numéro spécial << La mémoire >>*.

Kandel, A. et G. Langholz (éditeurs) [1992]. *Hybrid Architectures for Intelligent Systems*. CRC Press. 420p.

Kasabov, N. K. [1990]. Hybrid connectionist rule-based systems. Dans P. Jorrand et V. Sgurev (éditeurs), *Artificial Intelligence IV: Methodology, Systems, Applications*, pp. 227–235. Elsevier Science Publishers.

Kasabov, N. K. et S. H. Petkov [1992, août]. Neural Networks and Logic Programming - a Hybrid Model and its Applicability to Building Expert Systems. Dans B. Neumann (éditeur), *Proc. of the 10 th European Conference on Artificial Intelligence*, Vienna, Austria.

Kaski, S. et T. Kohonen [1994]. Winner-Take-All Networks for Physiological Models of Competitive Learning. *Neural Networks* 7(6/7), 973–984.

Kurfess, F. et M. Reich [1989]. Logic and Reasoning with Neural Nets. Dans R. Pfeifer et al. (éditeurs), *Connectionism in Perspective*, pp. 365–376. Elsevier Science Publishers B. V. (North Holland).

Kwasny, S. C. et K. A. Faisal [1992]. Symbolic Parsing Via Subsymbolic Rules. Dans Dinsmore [1992a], Chapitre 9, pp. 209–235.

Labbi, A. [1993, décembre]. Sur l'approximation et les systèmes dynamiques dans les réseaux neuronaux. Thèse de doctorat, Institut National Polytechnique de Grenoble, France.

Lacher, R. C. [1991, avril]. Expert Networks : Paradigmatic Conflict, Technological Rapprochement. Dans *Proc. of the 15 th annual Symposium in Philosophy*, Greensboro. University of North Carolina.

Lallement, Y. et S. Durand [1994, mai]. Ancrage de symboles dans des systèmes connexionnistes dédiés à des tâches de perceptions sensorielles. Dans Marchal et al. [1994], pp. 221–224.

Laurière, J.-L. [1987]. *Intelligence artificielle : résolution de problèmes par l'homme et la machine*. Paris, Eyrolles.

Letz, R., J. Schumann, S. Bayerl, et W. Bibel [1992]. SETHEO: A High-Performance Theorem Prover. *Journal of Automated Reasoning* 8(2), 183–212.

Lippman, R. P. [1987, avril]. An Introduction to Computing with Neural Nets. *IEEE ASSP Magazine*, 4–22.

Malek, M. et A. Labbi [1995, janvier]. A Coprocessing Model for Integrating CBR and Connectionnist Paradigms. Rapport technique, IMAG.

Marchal, P. et al. (éditeurs) [1994, mai]. *7mes journées Neurosciences et sciences de l'ingénieur (NSI94)*, Chamonix. NSI-ACTH.

Masa, P., K. Hoen, et H. Wallinga [1993]. 20 million patterns per second VLSI neural network pattern classifier. Dans *Proc. of the 3 nd International Conference on Artificial Neural Networks*, Amsterdam, The Netherlands, pp. 1058–1061.

Medsker, L. R. [1994]. *Hybrid Neural Networks and Expert Systems*. Kluwer Academic Publishers. 240p.

Medsker, L. R. et D. L. Bailey [1992]. Models and Guidelines for Integrating Expert Systems and Neural Networks. Dans Kandel et Langholz [1992], Chapitre 8, pp. 154–171.

Meeden, L., G. McGraw, et D. Blank [1994]. Emergent Control and Planning in an Autonomous Vehicle. Dans *Proc. of the Fifteenth Annual Conference of the Cognitive Science Society*. à paraître.

Memmi, D. [1993]. Representations in connectionist networks. Dans *Proc. of the 6 th International Conference on Neural Networks and their Applications*, Nîmes, France, pp. 361–370.

Mendelsohn, P. [1992]. Communication personnelle. Faculty of Psychology and Education Sciences, Geneva University.

Murtagh, F. et M. Hernández-Pajares [1994]. The Kohonen Self-Organizing Map Method : An Assessment. *Journal of Classification*. à paraître.

Nerrand, O., P. Roussel-Ragot, L. Personnaz, G. Dreyfus, et S. Marcos [1993]. Neural Networks and Nonlinear adaptative Filtering: Unifying Concepts and New Algorithms. *Neural Computation* (5), 165–199.

Newell, A. et H. A. Simon [1976]. Computer science as empirical inquiry : symbols and search. *Communications of the ACM 19*(3), 113–126.

Orsier, B. [1990, juin]. Intégration de tâches, méthodes et procédures dans une représentation de connaissances par objets. Rapport de DEA, Institut National Polytechnique de Grenoble.

Orsier, B. [1993a, juin]. L'ancrage des symboles, un second souffle pour les systèmes hybrides? Dans Amy et al. [1993], pp. 119–126.

Orsier, B. [1993b, février]. Modélisation de l'expertise en test paramétrique : faisabilité d'un système hybride << symboli-connexionniste >>. Rapport de fin d'études, Convention CNET – ADR/LIFIA, N° 923B001. 57 p.

Orsier, B., B. Amy, V. Rialle, et A. Giacometti [1994, août]. A Study of the Hybrid System SYNHESYS. Dans Hilario [1994], pp. 1–9.

Orsier, B., I. Iordanova, V. Rialle, A. Giacometti, et A. Vila [1994]. Hybrid systems for expertise modeling : from concepts to a medical application in electromyography. *Computers and Artificial Intelligence 13*(5), 423–440.

Pan, J. et J. Tenenbaum [1986]. PIES : An Engineer's Do-It-Yourself Knowledge System for Interpretation of Parametric Test Data. *AI Magazine Fall*, 62–68.

Perez, R. A., L. O. Hall, S. Romaniuk, et J. T. Lilkendey [1992]. Inductive Learning For Expert Systems In Manufacturing. Dans J. F. Nunamaker (éditeur), *Proc. of the Hawaii International Conference on System Sciences*, Volume III, pp. 14–25. IEEE Computer Society Press.

Pham, K. M. [1992]. IntelliSphere (c) Version 1.1, Générateur de Systèmes à Base de Connaissances Distribués Macro-connexionnistes. Document pour information, InferOne S. A.

Pham, K. M. et P. Degoulet [1989]. MOSAIC : Medical Knowledge Processing Based on a Macro-Connectionist Approach to Neural Networks. Dans *Proc. of the 6 th Congress on Medical Informatics MEDINFO 89*, Volume I, pp. 82–86.

Pican, N., P. Bresson, F. Alexandre, et al. [1993]. A perceptron with optimized backpropagation learning algorithm to preset a temper mill machine : NEUROSKIN. Dans *Proc. of the 6 th International Conference on Neural Networks and their Applications*, Nîmes, France, pp. 17–24.

Pinkas, G. [1992, juillet]. Representing Unrestricted First-Order Logic Formulas in Connectionist Networks. Dans Sun et Bookman [1992].

Plunket, K. et V. Marchman [1991, janvier]. U-shaped learning and frequency effects in a multilayered perceptron : implications for child language acquisition. *Cognition 38*(1), 43–102.

Pollack, J. B. [1990]. Recursive Distributed Representations. *Artificial Intelligence* (46), 77–105.

Pomerleau, D. A., J. Gowdy, et C. E. Thorpe [1991]. Combining Artificial Neural Networks and Symbolic Processing for Autonomous Robot Guidance. *Engng Applic. Artif. Intell. 4*(4), 279–285.

Prechelt, L. [1994, août]. A Study of Experimental Evaluations of Neural Network Learning Algorithms: Current Research Practice. Rapport technique 19/94, Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany.

Quinlan, J. R. [1986]. Induction of Decision Trees. *Machine Learning 1*, 81–106.

Ram, A. et J. C. Santamaria [1993]. A Multistrategy Case-Based and Reinforcement Learning Approach to Self-Improving Reactive Control Systems for Autonomous Robotic Navigation. Dans *Proc. of the Second International Workshop on Multistrategy Learning*, Harpers Ferry, WV.

Rechenmann, F. [1991, juin]. Intelligence artificielle et modélisation de systèmes dynamiques. Mémoire d'habilitation à diriger des recherches, INPG, Grenoble.

Rechenmann, F., P. Uvietta, et P. Fontanille [1988]. Shirka: un système à base de connaissances orienté objets. Manuel de référence, INRIA/ARTEMIS, Grenoble, France.

Reignier, P. [1994a]. Fuzzy Logic Techniques for Mobile Robot Obstacle. *International Journal on Robotics and Autonomous Systems 12*, 143–153. ISSN 0921-8890.

Reignier, P. [1994b, décembre]. Pilotage réactif d'un robot mobile – étude du lien entre la perception et l'action. Thèse de doctorat, Institut National Polytechnique de Grenoble, France.

Rialle, V. [1993]. Communication personnelle.

Salzberg, S. [1991]. A nearest hyperrectangle learning method. *Machine Learning 3*(3), 251–276.

Schwartz, J. [1992]. Who's Afraid of Multiple Realizability?: Functionalism, Reductionism, and Connectionism. Dans Dinsmore [1992a], Chapitre 5, pp. 89–112.

Shastri, L. [1987]. A Connectionist Encoding of Semantic Networks. Dans M. Huhns (éditeur), *Distributed Artificial Intelligence*, Chapitre 7, pp. 177–202. Morgan Kaufman, Calif.

Simonet, A. [1984]. Types abstraits et bases de données : formalisation du concept de partage et analyse statique de contraintes d'intégrité. Thèse de doctorat, Université Joseph Fourier, Grenoble, France.

Simonet, A., M. Simonet, C. G. Bassolet, et J. Demongeot [1994, juin]. Une architecture connexionniste pour un système de représentation de connaissances orienté-objet. Dans *Colloque sur le neuromimétisme*, Nouveaux outils, Lyon. Hermès.

Stolcke, A. et D. Wu [1992, juillet]. Tree Matching with Recursive Distributed Representations. Dans *Proc. of the AAAI-92 Workshop «Integrating Neural and Symbolic Processes - The Cognitive Dimension»*, San Jose, California.

Sun, R. [1991]. *Integrating Rules and Connectionism for Robust Reasoning.* Ph. D. thesis, Brandeis University.

Sun, R. [1992]. A connectionist module for commonsense reasoning incorporating rules and similarities. *Knowledge Acquisition 4*, 293–321.

Sun, R. et L. Bookman [1993, Summer]. How do Symbols and Networks Fit Together (A report from the Workshop on Integrating Neural and Symbolic Processes). *AI Magazine 14*(2), 20–23.

Sun, R. et L. A. Bookman (éditeurs) [1992, juillet]. *Proc. of the AAAI-92 Workshop «Integrating Neural and Symbolic Processes - The Cognitive Dimension»*, San Jose, California.

Tirri, H. [1991]. Implementing Expert System Rule Conditions by Neural Networks. *New Generation Computing* (10), 55–71.

Touretzky, D. S. [1989, août]. BoltzCONS : Dynamic Symbol Structures in a Connectionist Network. Rapport technique CMU-CS-89-182, Carnegie Mellon University.

Touretzky, D. S. et G. E. Hinton [1986]. A Distributed Connectionist Production System. Rapport technique CMU-CS-86-172, Carnegie Mellon University.

Towell, G. G. [1992, janvier]. Symbolic knowledge and neural networks : insertion refinement and extraction. Rapport technique 1072, Univ. of Wisconsin-Madison, Computer Science Dept.

Varela, F. J. [1989]. *Connaître les sciences cognitives, tendances et perspectives.* Seuil.

Verleysen, M. (éditeur) [1994, avril]. *European Symposium on Artificial Neural Networks*, Brussels.

Weigend, A. S. [1993]. Book Review. *Artificial Intelligence* (62), 93–111.

Weiss, S. M. et C. A. Kulikowski [1991]. *Computer Systems That Learn.* Morgan Kaufman.

Willamovski, J. [1994, avril]. Modélisation de tâches pour la résolution de problèmes en coopération système-utilisateur. Thèse de doctorat, Université Joseph Fourier, Grenoble, France.

Wilson, A. et J. Hendler [1993]. Linking Symbolic and Subsymbolic Computing. Dans Bookman et Sun [1993], pp. 395–414.

Zeidenberg, M. [1990]. *Neural networks in artificial intelligence.* Ellis Horwood. 268p.

Zell, A. et al. [1993]. SNNS, Stuttgart Neural Network Simulator, User Manual, Version 3.0. Rapport technique 3/93, University of Stuttgart, Institute for Parallel and Distributed High Performance Systems.