Apprentissage autonome de réseaux de neurones pour le pilotage en temps réel des systèmes de production basé sur l'optimisation via simulation

Wiem Mouelhi-Chibani

N. d'ordre : DU : 1958

N. EDSPIC : 450

UNIVERSITE BLAISE PASCAL – CLERMONT II

ECOLE DOCTORALE

SCIENCES POUR L'INGENIEUR DE CLERMONT-FERRAND

THESE

Présentée par

Wiem MOUELHI-CHIBANI

Pour obtenir le grade de

DOCTEUR D'UNIVERSITE

Spécialité : INFORMATIQUE

Apprentissage autonome de réseaux de neurones pour le pilotage en temps réel des systèmes de production basé sur l'optimisation via simulation

---

Soutenue publiquement le 12 octobre 2009 devant le jury :

M. Alain QUILLOT	Président
M. Pierre CASTAGNA	Rapporteur
M. Georges HABCHI	Rapporteur
Mme. Anne-Lise HUYET	Examinateur
M. Henri PIERREVAL	Directeur de thèse

Remerciements

Une thèse n'est pas une fin en soi, mais c'est un moment particulier dans la vie d'un chercheur : il y aura eu un avant qui ne sera plus, et il y aura un après à construire. Aussi, au moment de franchir cette limite, je ne peux pas ne pas penser à tous ceux qui, de près ou de loin, auront contribué à ce grand effort car, si l'épreuve est individuelle, ses implications sont sociales, académiques, familiales, et humaines tout simplement.

Je tiens tout d'abord à remercier M. Henri PIERREVAL, qui m'a accepté dans son équipe, a consacré beaucoup de son temps à mon travail, et a fait de nombreuses remarques, toujours intéressantes, sur cette thèse, je le remercie pour sa patience, son soutien et ses conseils qui ont contribué à rendre ce travail très enrichissant sur le plan des connaissances et sur le plan personnel. Je lui exprime ici toute ma gratitude et toute ma sympathie. M. Henri PIERREVAL a toujours su diriger mes recherches dans la bonne direction; je dois beaucoup à lui pour ses nombreuses contributions à la réussite de cette thèse.

Je remercie M. Alain QUILLOT, directeur du LIMOS de m'avoir accueillie dans son laboratoire et d'avoir accepté de présider le jury de thèse.

Je suis extrêmement reconnaissante à Monsieur Pierre CASTAGNA, Professeur à l'université de Nantes et à Monsieur Georges HABCHI, professeur à l'Université de Savoie d'avoir accepté la charge d'évaluer ce travail en qualité de rapporteurs, et d'avoir relevé les insuffisances et erreurs du manuscrit.

Mes remerciements à tout le personnel de l'Institut Français de Mécanique Avancée (IFMA) et le personnel du Laboratoire d'Informatique de Modélisation et Optimisation des systèmes (LIMOS) pour leur aide et soutien.

Je tiens aussi à remercier en particulier Madame Anne-Lise Hyuet qui m'a beaucoup aider et m'a beaucoup soutenu et qui a accepté d'être l'examinateur de ma thèse.

J'ai gardé une petite place à part pour tout que je considère mes amis et qui ont été toujours là pour me soutenir : Feiza, Saber, Sabeur, Mahmoud et Béatrice Bourdieu des formidables personnes qui m'ont épaulées pendant les quatre années précédentes.

Bien évidemment j’adresse un grand merci à toute ma famille qui a toujours été présente lorsque j’en ai eu besoin, en particulier à ma mère, à mon père, à mon frère et ma grand-mère. Merci tout simplement d’avoir été là et de me donner jour après jour autant d’amour.

Enfin, mais non des moindres, tous mes remerciements à mon mari bien aimé, sans lequel cette thèse n'aurait jamais vu le jour. Je tiens à le remercier pour tout ce qu’il m’apporte au quotidien d’attention, de soutien et d’amour.

Je n’oublierai sans doute pas mon cousin Driss qui m’a donné le tout premier coup de pousse pour commencer mon grand parcours de chercheur ; à l’âme de son ainé adoré Bélir.

SOMMAIRE

INTRODUCTION GENERALE ……12

Chapitre I CONTEXTE – PROBLEMATIQUE …….15

1. Introduction ……..15

2. Le pilotage des systèmes de production ……..……..16

2.1. Pilotage centralisé……..17

2.2. Pilotage hiérarchisé……18

2.3. Pilotage coordonné…….18

2.4. Pilotage distribué……...19

2.5. Pilotage distribué supervisé ……..…19

2.6. Pilotage holonique…….....19

2.7. Pilotage par le produit……...20

3. Prise de décision au niveau opérationnel (physique)……21

3.1. Prise de décision en temps réel…….21

3.2. Les types de décisions prises en temps réel …….22

3.2.1. Les décisions d'affectation……..22

3.2.2. Les décisions d'attribution…….23

3.2.3. Les décisions de remise en cause……25

3.2.4. Les décisions d'anticipation……...26

3.2.5. Les décisions de déclenchement des activités de maintenance ou d'entretien……27

3.2.6. Décisions de régulation et d'ajustement……28

3.2.7. Combinaison de différents types de décisions……..28

3.3. Les critères de prise de décision……...29

4. Conclusion……..…31

Chapitre II : L'APPRENTISSAGE AUTOMATIQUE PAR SIMULATION DANS LES SYSTEMES DE PRODUCTION……33

1. Introduction……..33

2. L'apprentissage automatique……..34

2.1. Méthode d'induction …….35

2.2. Les machines à vecteurs supports (SVM)…….38

2.3. Les réseaux de neurones……40

2.3.1. Les réseaux de neurones non bouclés……41

2.3.2. Les réseaux de neurones bouclés……...42

2.3.3. L'apprentissage des réseaux de neurones…….43

2.4. Méthodes statistiques : Naive Bayes…….44

2.5. Méthodes Hybrides d'Apprentissage Automatique……45

2.5.1. **Systèmes Symbolique-Flous……...…46**

2.5.2. Systèmes Symbolique-Génétiques ……46

2.5.3. Systèmes Neuro-Flous ……46

3. L'apprentissage basé sur la simulation pour le pilotage des systèmes de production ……..…….…….……47

3.1. Le rôle de la simulation dans l'apprentissage pour le pilotage des systèmes de production ……..……48

3.2. Les travaux relatifs à l'apprentissage automatique à partir de la simulation des systèmes de production ……49

4. Conclusion ……59

Chapitre III : APPRENTISSAGE AUTONOME DES RESEAUX DE NEURONES BASE SUR OPTIMISATION VIA SIMULATION……...60

1. Introduction ……60

2. Notre formulation du problème : description et objectifs……61

3. L'approche proposée……66

3.1. Une approche basée sur les réseaux de neurones ……...66

3.1.1. Les réseaux de neurones : approximateurs universels parcimonieux …...66

3.1.2. La prise de décision par le réseau de neurones ……...69

3.2. Auto-apprentissage des réseaux de neurones par optimisation via simulation……...…71

3.2.1. L'optimisation via simulation ……...72

3.2.1.1.Le recuit simulé……...73

3.2.1.2.Les algorithmes évolutionnistes ……....74

3.2.1.3.La recherche tabou ……....76

3.2.1.4.Algorithme d'échantillonnage ……...76

3.2.1.5.Méthode de surface de gradient……..77

3.2.2. Le principe d'auto apprentissage des réseaux de neurones ……...77

4. Implémentation ……79

4.1. Module de simulation ……...……81

4.1.1. Notions de base ……...81

4.1.2. Lien entre les modèles ARENA et les fichiers DLL ……85

4.2. Module du réseau de neurones…….87

4.2.1. Architecture du réseau de neurones …….87

4.2.2. Fonction d'activation …….89

4.2.3. Adaptation des entrées/sorties du réseau de neurones aux données de la simulation ……...91

4.3. Module de l'optimisation ……..92

5. Conclusion ……95

Chapitre IV : APPLICATIONS ……96

1. Introduction ……96

2. Exemple 1 : Application de l'approche d'apprentissage autonome des réseaux de neurones pour le pilotage d'un atelier Flowshop …….……96

2.1. Description de l'exemple ……..……96

2.2. Expérimentations ……99

2.2.1. Le module du réseau de neurones ……..100

2.2.1.1.Les entrées du réseau de neurones : un réseau de neurones pour les deux files d'attente……...101

2.2.1.2.Les entrées du réseau de neurones : un réseau de neurones pour chaque file d'attente……....102

2.2.1.3.Sorties du réseau de neurones ……105

2.2.1.4.Architecture du réseau de neurones ……...…105

2.2.2. Le module d'optimisation ……105

2.3. Résultats des expérimentations ……..107

2.3.1. Les meilleures combinaisons de règles de priorité trouvées par Barrett et Barman (1986) ……..108

2.3.2. Les résultats obtenus par le réseau de neurones ……...……108

3. Exemple 2 : Application de l'approche d'apprentissage autonome des réseaux de neurones pour le pilotage d'un atelier Jobshop ……...……115

3.1. Description de l'exemple ……...……..115

3.2. Expérimentations ……118

3.2.1. Le module de simulation……...118

3.2.2. Le module du réseau de neurones ……..……..…118

3.2.3. Le module d'optimisation ……..……...120

3.3. Les résultats des expérimentations ……...……..121

4. Conclusion …….……..125

CONCLUSION GENERALE ET PERSEPECTIVES ……...……..126

Bibliographie ……...…….129

Annexe A ……...……141

TABLE DES ILLUSTRATIONS

Figure 1. Décomposition systémique : (a) système entreprise, (b) système de production. (Le Moigne, 1990) 16
Figure 2. Le produit actif comme le lien entre les diverses approches de pilotage (Morel *et al.*, 2007) 20
Figure 3. Décision d'affectation mono-ressource 23
Figure 4. Décision d'affectation multi-ressources 23
Figure 5. Décision d'attribution simple 24
Figure 6. Décision d'attribution multiple 24
Figure 7. Réactions aux perturbations 25
Figure 8. Remise en cause suite à des dérives 26
Figure 9. Déclenchement d'actions de maintenance ou entretien 27
Figure 10. Combinaison affectation/ attribution 28
Figure 11. Exemple d'Arbre de Décision Simple [F. Santos Osório, 1998] 37
Figure 12. La transformation non linéaire des données peut permettre une séparation linéaire des exemples dans un nouvel espace. Adapté de [A. Cornuéjols et al. 2002]. 39
Figure 13. L'hyperplan séparateur optimal est celui qui maximise la marge dans l'espace de redescription. 39
Figure 14. Un neurone formel 40
Figure 15. Un modèle de réseau de neurones 41
Figure 16. Forme canonique d'un réseau de neurones bouclé 43
Figure 17. Principe d'un modèle de simulation 49
Figure 18. Développement d'un métamodèle à partir d'un réseau de neurones 51
Figure 19. Intégration du réseau de neurones dans le modèle de simulation pour l'aide à la décision. 53
Figure 20. Choix des règles de priorité statiquement 54
Figure 21. Choix des règles de priorité périodiquement 55
Figure 22. Choix des règles de priorité dynamiquement 55
Figure 23. La prise de décision par le réseau de neurones dans un système de production 69
Figure 24. Optimisation de la logique de prise de décision via simulation 77
Figure 25. La méthode globale de l'apprentissage autonome des réseaux de neurones par optimisation via simulation. 78
Figure 26. Interaction entre les trois modules (la première phase de l'approche proposée) 79
Figure 27. Interaction entre les deux modules (la deuxième phase de l'approche proposée) 79
Figure 28. Architecture du réseau de neurone utilisé dans notre approche. 87
Figure 29. Le Principe du calcul du réseau de neurones 88
Figure 30. Courbe de la tangentielle : y = tanh(x) 90
Figure 31. Flux de données entre les fichiers externes et les modules : Optimisation, Réseau de neurones et Simulation. 94
Figure 32. L'atelier flowshop. (Barett et Barman, 1986) 96
Figure 33. Un paysage d'énergie basé sur le recuit simulé. 111
Figure 34. Représentation de l'atelier à cheminements multiples étudié et des différents flux . 116
Figure 35. Le réseau de neurones utilisé pour le choix des règles de priorité 119
Figure 36. Evolution du retard moyen 121

LISTE DES TABLEAUX

Tableau 1. Ensemble d'Apprentissage : Conditions Météorologiques 37
Tableau 2. Variables d'état d'un système de production (Shiue et Guh, 2006) ..... **Erreur ! Signet non défini.**
Tableau 3. Tableau récapitulatif des différents types de fonctions d'activation les plus utilisées 89
Tableau 4. Variables d'entrée du réseau de neurones communiquées par la simulation (proposition d'un réseau de neurones pour les deux files d'attente) 100
Tableau 5. Variables d'entrées du premier réseau de neurones de la première file d'attente communiquées par la simulation. 102
Tableau 6. Variables d'entrées du deuxième réseau de neurones de la deuxième file d'attente communiquées par la simulation. 103
Tableau 7. Les règles de priorités 104
Tableau 8. Les paramètres du recuit simulé à initialiser par l'utilisateur 106
Tableau 9. Retards moyens obtenus à partir des meilleures combinaisons de règles de priorité proposées par Barrett et Barmann 107
Tableau 10. Les résultats des différentes expérimentations de l'apprentissage du réseau de neurones avec nombre de neurones dans la couche cachée est 3. 108
Tableau 11. Les résultats des différentes expérimentations de l'apprentissage du réseau de neurones avec nombre de neurones dans la couche cachée est 5. 109
Tableau 12. Les résultats des différentes expérimentations de l'apprentissage du réseau de neurones avec nombre de neurones dans la couche cachée est 7. 110
Tableau 13. Le résultat de l'expérimentation de l'apprentissage du réseau de neurones avec nombre de neurones dans la couche cachée est 12. 110
Tableau 14. Les résultats des différentes expérimentations de l'apprentissage du réseau de neurones avec nombre de neurones dans la couche cachée est 3. 114
Tableau 15. Séquences des pièces en fonction de leur type 115
Tableau 16. Paramètres des lois des durées opératoires 115
Tableau 17. Les résultats des six règles de priorité sur les quatre machines de l'atelier jobshop 120
Tableau 18. Les résultats des différentes expérimentations de l'apprentissage du réseau de neurones 123
Tableau 19. Les résultats des différentes expérimentations de l'apprentissage du réseau de neurones 124

INTRODUCTION GENERALE

Pour rester compétitives sur des marchés de plus en plus incertains, les entreprises ont besoin d'être réactives. Elles doivent souvent faire face à des événements imprévus, comme par exemple la prise en compte d'une commande urgente, une annulation ou une modification de commande, les interruptions aléatoires du système de production, etc. Ceci nécessite un pilotage efficace de la production en temps réel, afin de pouvoir réagir face aux événements critiques. En effet, le pilotage des systèmes de production est devenu un sujet qui a de plus en plus d'importance dans la mesure où ils nécessitent une prise de décisions plus complexes d'une part, et une poursuite de plus en plus fine d'objectifs associés à des niveaux de performance élevés d'autre part.

De ce fait, les systèmes de production ont de plus en plus besoin d'être pilotés en temps réel afin d'être plus réactifs: ils réagissent continûment aux stimulis de leur environnement. Ils doivent, en plus, réagir dans un délai donné, en remplissant les contraintes imposées par l'environnement externe et les contraintes internes (ressources limitées, délais courts, commandes urgentes,…) quelle que soit la charge de l'atelier.

Cependant, la difficulté réside dans la prise de décisions adéquates en temps réel dans le but d'optimiser globalement tout un système, c'est-à-dire, optimiser une performance donnée en tenant compte de toutes les décisions prises au temps réel de telles sortes que ces dernières soient cohérentes avec les décisions passées et les décisions futures. Par exemple, le problème de savoir quand ajouter, retirer des ressources en temps réel se pose souvent dans la pratique. De même l'affectation des ressources doit se faire en temps réel (chariot auto-moteur, navette,…).

Cependant, les décisions nécessaires qui doivent être prises à chaque fois celles-ci sont requises pendant le processus du pilotage du système en temps réel, dépendent de plusieurs critères. D'abord, il est nécessaire de prendre en compte les caractéristiques de l'atelier, tels que le nombre de machines, le nombre d'opérateurs sur chaque machine, les temps de livraison des pièces, le taux de chargement de l'atelier, etc. D'autre part, il est essentiel de caractériser l'état courant de l'atelier qui change et évolue constamment dans le temps, il est donc primordial de déterminer les variables qui reflètent au mieux l'état de l'atelier et sa dynamique.

Plusieurs travaux sont présentés dans la littérature pour aider au pilotage en temps réel; il existe les méthodes qui sont basées sur les heuristiques, les méthodes basées sur les modèles dynamiques, les méthodes basées sur l'intelligence artificielle…

Parmi toutes ses méthodes utilisées dans la littérature pour le pilotage en temps réel et l'aide à la décision, nous nous intéressons en particulier aux méthodes basées sur l'intelligence artificielle couplées aves la simulation. Notre objectif est de montrer l'intérêt d'utiliser la simulation pour « créer » de la connaissance indispensable à l'apprentissage automatique des méthodes basées sur l'intelligence artificielle.

L'intelligence est généralement issue de connaissances et elle est éventuellement capable de prendre en compte de l'imprécision, telles que les approches floues ou les approches possibilistes. La difficulté connue de ces approches réside dans l'élicitation des connaissances dont doivent être dotée ces systèmes. En effet, la recherche de connaissances pertinentes pour piloter efficacement constitue un écueil majeur pour la mise en œuvre du pilotage intelligent d'ateliers. Mais *Comment obtenir les connaissances nécessaires au pilotage ?*

Ces connaissances doivent notamment être capables de déterminer comment affecter au mieux les moyens de production (machines, transport, opérateurs, outils, etc.) de sorte à satisfaire aux objectifs de production (e. g. minimiser les en-cours). Dans la mesure où le pilotage s'effectue sur la base d'un état du système changeant dynamiquement au cours du temps, il est très difficile de disposer d'exemples des « meilleures pratiques » pour le pilotage.

Dans cette thèse nous allons présenter une étude complète qui résout le problème du pilotage en temps réel en se basant sur des méthodes d'intelligence artificielle.

Le manuscrit de la thèse est organisé comme suit : d'abord et dans le premier chapitre nous présentons les systèmes de production et expliquons l'importance de leur pilotage en temps réel.

Ensuite dans le deuxième chapitre nous présentons un état de l'art des méthodes d'apprentissage automatique, et l'importance des exemples d'apprentissage pour l'apprentissage de ces méthodes, à la fin du chapitre nous montrons l'intérêt que représente la simulation pour l'élicitation de la connaissance nécessaire à l'apprentissage de ces méthodes pour le pilotage des systèmes de production.

Dans le troisième chapitre nous formulons le problème que nous visons à résoudre et présentons la méthode d'apprentissage automatique des réseaux de neurones basée sur l'optimisation via simulation pour le pilotage des systèmes de production.

Enfin dans le quatrième chapitre nous montrons la capacité d'apprentissage de notre approche en l'appliquant sur deux exemples d'atelier le premier un atelier flowshop et le second un atelier jobshop.

Chapitre I

CONTEXTE – PROBLEMATIQUE

1. Introduction

Pour rester compétitives, les entreprises manufacturières sont contraintes d'améliorer leur pilotage, tant au niveau stratégique, pour s'adapter aux progrès de la technologie ou suivre les évolutions du marché, qu'au niveau opérationnel, pour réagir face aux aléas.

Les systèmes de production peuvent être des systèmes très complexes et difficiles à gérer au vue de toutes leurs composantes fonctionnelles (fabrication, maintenance, gestion, pilotage, …). Plusieurs approches ont été envisagées dans le but de mieux comprendre leur fonctionnement et de mieux les appréhender.

Dans ce chapitre, nous exposons les principales approches de pilotage des systèmes de production. Une étude des propriétés de chaque approche nous a permis de mettre l'accent sur son utilité et son intérêt, pour aboutir finalement à la justification de la technique adoptée dans ce travail, et le contexte dans lequel nous nous positionnons par rapport à ces approches de pilotage. Nous mettons en évidence l'importance de la prise de décision en temps réel pour le pilotage pour pouvoir obtenir un fonctionnement efficace et efficient du système de production et satisfaire les objectifs assignés. Par la suite, après avoir présenté le cadre de travail de cette étude, nous abordons les différents types de décisions pouvant êtres prises en temps réel.

Le lecteur remarquera que ce chapitre fait référence à de nombreuses notions de base et des définitions de concepts bien connus dans la littérature. Nous attirons son attention sur le fait que notre objectif n'est pas de détailler ces concepts déjà largement développés dans la littérature, mais uniquement de donner un rapide « descriptif » de la bibliographie associée aux approches par rapport auxquelles nous nous positionnons dans cette thèse.

2. Le pilotage des systèmes de production

Un système de production est généralement vu comme l'association d'un ensemble de ressources en interaction entre elles pour réaliser une activité de production. En effet, la production s'effectue par une succession d'opérations dites de transformation, de transfert, d'assemblage et de désassemblage en exploitant les ressources disponibles (machines, moyens de transfert, …) afin de transformer les matières premières (composants entrants dans le système) en des produits finis sortants de ce système.

Le modèle conceptuel du système de gestion de production exprime la décomposition des systèmes de production en trois composantes : le sous-système physique de production, le sous-système de décision et le sous-système d'information (Le Moigne, 1990).

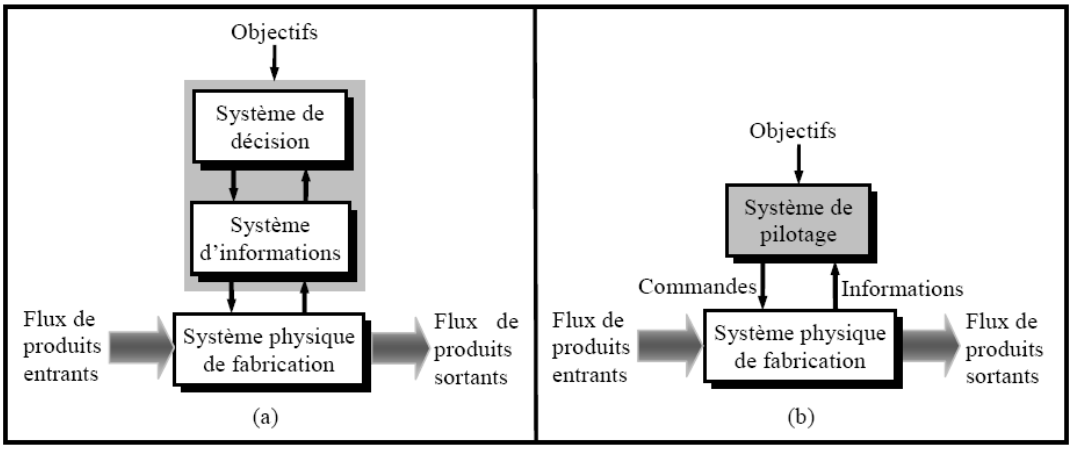


Figure 1. Décomposition systémique : (a) système entreprise, (b) système de production. (Le Moigne, 1990)

La figure 1 illustre les liaisons entre ces trois systèmes et les trois types de flux associés. Pour un fonctionnement coordonné de l'entreprise, les flux physiques, d'information et de décision doivent être synchronisés en fonction du temps.

Le système physique, il est également appelé système opérationnel ou système technologique ou système de fabrication, et a comme fonction la transformation d'inputs en outputs. Ce système comprend des ressources (humaines et matérielles) et un ensemble d'opérations de transformations requises pour réaliser, à partir des inputs et des ressources, le produit fini. Le système physique peut être organisé de différentes façons : en lignes flexibles, en sections

homogènes, en chaînes de transfert, en îlots de production, ... Afin que ce système puisse fonctionner et être piloté de façon efficace, il est nécessaire qu'il soit accompagné d'un système de décision et d'un système d'information.

Le système d'information, il assure essentiellement les fonctions de la collecte, du traitement, et de la mémorisation des informations venues de l'environnement du système de production, mais également du système lui-même. Il assure aussi l'échange de ces informations entre le système de décision et le système physique.

Le système de décision, il est également appelé système de conduite ou système de pilotage. Il a pour but, à partir des commandes prévisionnelles, de définir les paramètres de production et de piloter, par ses décisions, l'évolution du système physique. Il décide en fonction du comportement du système physique, de l'état de l'environnement et des objectifs qui lui ont été assignés. Les décisions peuvent être humaines, assistées par ordinateur (systèmes d'aide à la décision, systèmes experts) ou automatisées. La structuration (organisation) du système de décision est étroitement liée aux caractéristiques physiques de l'atelier, tels que la fréquence des commandes et le type de production à réaliser.

Certes, il est difficile d'établir une classification exhaustive de toutes les approches de pilotage des systèmes de production publiés dans la littérature. Cependant, nous essayons de proposer dans ce qui suit les approches de pilotage les plus connues.

2.1. Pilotage centralisé

Il s'agit de l'approche la plus classique et la plus ancienne. Elle se caractérise par un pilotage localisé au sein d'une ressource unique qui supervise la production et gère en temps réel, les événements qui surviennent tout au long de la production, sur le système composé de plusieurs ressources.

Le pilotage est très spécifique à l'environnement de production. Cependant, la capacité de régulation est réduite par la rigidité structurelle. Ainsi, la structure centralisée est par essence peu robuste et ne peut efficacement intégrer la résistance aux perturbations (Trentesaux, 1996).

2.2. Pilotage hiérarchisé

Dans ce type d'approche, la notion de niveau d'abstraction permet de modéliser une usine complète. Chaque niveau coordonne les unités de pilotage du niveau inférieur, et ce jusqu'au niveau le plus bas. Donc, chaque niveau a des relations de dépendance vis à vis du niveau supérieur et de dominance vis à vis du niveau inférieur. Chaque décision est élaborée au niveau où un problème est détecté. Les niveaux inférieurs traitent cette décision comme une contrainte et transmettent en retour une information de suivi au niveau supérieur. La gestion temps réel concerne les niveaux atelier, cellule, poste de travail et automate. Différentes présentations détaillées des nombreuses études réalisées, sont fournies dans (Archimède 1991, Trentesaux 1996, Dindeleux 1992, Youssef 1998).

Ainsi, cette alternative basée sur une décomposition du problème global en sous-problèmes, et sur le concept d'agrégation, est souvent utilisée. Le principe de décomposition permet de ramener la résolution du problème global à la résolution successive de sous-problèmes de dimension et de complexité acceptables. Au niveau supérieur de la structure hiérarchisée, l'information est synthétisée. Le problème à résoudre concerne des variables agrégées représentant un ensemble de variables élémentaires. La solution agrégée fournie par ce niveau devient une contrainte pour le niveau inférieur, qui porte sur des variables plus détaillées.

2.3. Pilotage coordonné

L'approche coordonnée correspond à un ensemble de structures hiérarchisées. La différence est l'existence d'une coopération à un même niveau. Ces structures accroissent théoriquement la capacité de décision au sein de chacun de ces niveaux. Ainsi, cette coopération doit améliorer la réactivité en proposant une décomposition dynamique du problème au niveau où il apparaît, plutôt qu'une décomposition hiérarchique des ordres vers les niveaux inférieurs (un exemple de structure coordonnée : le modèle CODECO, Conduite DEcentralisée COordonnée d'atelier (Kallel 1985). Cet aspect dynamique et cette coopération enrichissent la qualité de la décision. Le résultat est une meilleure intégration en temps réel des perturbations.

2.4. Pilotage distribué

Il est fondé sur une distribution complète de la décision sur l'ensemble des ressources pilotant un centre de production. Le contrôle d'une telle structure est beaucoup plus complexe en raison de sa modularité, par rapport à une structure hiérarchique qui est plus rigide mais plus facilement maîtrisable (Trentesaux, 1996).

2.5. Pilotage distribué supervisé

Un pilotage distribué supervisé se caractérise par un ensemble d'entités coopérantes sous le contrôle d'une entité superviseur dont le rôle est d'imposer, de conseiller ou de modifier une décision afin de respecter un objectif plus global. Un superviseur, qui possède une vision plus globale du processus de production est rajouté à la méthode distribuée.

Cette approche est un compromis entre les structures distribuée et centralisée. Elle permet une gestion globale et efficace par la centralisation de contrôle d'une part, et une meilleure réaction aux perturbations par la distribution des capacités de décision d'autre part.

Le modèle de Kouiss par exemple (Kouiss et al. 1995), est dédié au pilotage d'atelier et se base sur une architecture distribuée supervisée dans un cadre sans ordonnancement prévisionnel. Le modèle utilise des agents (chacun des agents étant composé de trois sous-systèmes : le sous-système de connaissances, le sous-système d'expertise et le sous-système de communication) et le niveau de supervision est dédié aux règles d'allocations utilisées par chaque agent.

2.6. Pilotage holonique

L'approche holonique a été introduite par Koestler en 1989 d'après Roy (Roy 1998) pour des organisations sociales et des systèmes s'auto-organisant. C'est une approche originale qui considère un système comme un agglomérat d'unités distribuées et autonomes : les « holons ». Un système holonique est une société d'holons qui coopèrent pour atteindre un objectif et qui agissent en tant qu'ensembles d'entités coopérantes. On l'appelle une holarchie.

Cette approche marque une rupture avec les modèles hiérarchiques antérieurs, où les comportements sont du type 'maître – esclave' selon une topologie arborescente et invariante des centres de décision, le tout renforcé par le respect des ordres par le centre de décision esclave. En effet, le holon a une intelligence décisionnelle (Cardin et al. 2009, Blanc et al. 2008) qui lui permet d'agir sur son propre comportement, mais qui lui permet également d'intervenir sur le comportement du système auquel il appartient (Pujo et Ounnar 2007).

2.7. Pilotage par le produit

Le contrôle par le produit (Morel et al. 2007) est une déclinaison de cette approche distribuée, visant à articuler les diverses activités de l'entreprise autour de produits *actifs* (McFarlane et al. 2003), capables d'informer les acteurs qui gravitent autour de lui (Figure 2). Du point de vue du pilotage de la production, le produit pourra en particulier servir à faire coexister des approches réactives et locales de la prise de décision (par exemple des initiatives prises par des équipes d'opérateurs responsabilisés), avec des approches plus prédictives et globales (supportées par exemple par des progiciels de gestion intégrés ou des systèmes de planification avancés). Les apports espérés du pilotage par le produit sont la simplification du système d'information, centré sur un objet unique, ainsi que la possibilité d'hybrider les approches centralisées et distribuées de la prise de décision, pour arriver à des systèmes à la fois performants et flexibles (Thomas 2004).

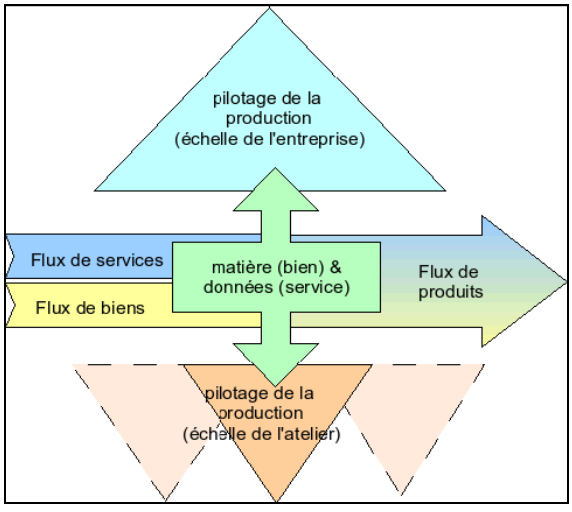


Figure 2. Le produit actif comme le lien entre les diverses approches de pilotage (Morel et al. 2007)

Depuis quelques temps émerge en France une communauté scientifique qui s'intéresse aux systèmes contrôlés par le produit (Bajic et al. 2007), plus particulièrement, à la maîtrise de l'interaction entre le procédé de fabrication et le produit, en intégrant dans la boucle cybernétique de nouvelles technologies telles que les communications sans fil (Wireless Sensors Network) et le RFID (Radio Frequency Identification). Ces technologies doivent permettre de doter le produit de capacités de mémorisation, de calcul et de communication : il devient alors « actif » au sein du système de production qui le traite. Il devient donc capable de capter les variations de son environnement, de prendre des décisions et donc d'interagir avec son environnement (ressources constituant le procédé, autres produits, opérateurs humains…).

De nombreux chercheurs voient dans ces nouvelles capacités l'opportunité d'améliorer amplement les performances du pilotage opérationnel des systèmes de production (McFarlane et al. 2002) (Gouyon 2004) (Pétin et al. 2007). Cette voie est également préconisée par le Comité d'Experts en Productique du CNRS, dans sa dernière analyse prospective (Bourrières et al. 2007).

3. Prise de décision au niveau opérationnel (physique)

3.1. Prise de décision en temps réel

Dans le cadre de cette étude nous nous intéressons à la prise de décision au niveau opérationnel. En effet, au niveau de l'atelier, la prise de décision signifie la détermination des actions à appliquer à court terme, voir même en temps réel, pour pouvoir obtenir un fonctionnement efficace et efficient du système de production et satisfaire les objectifs assignés. Pour ce faire, le décideur (l'opérateur du pilotage) prendra un ensemble de décisions pour conquérir au mieux à ces objectifs. En effet, il est classique de distinguer, parmi les décisions qui doivent intervenir lors de la vie d'un atelier, celles qui peuvent être prises « hors ligne », la veille, la semaine ou le mois précédent, de celles qui doivent être prises « en ligne », en temps réel. Dans le premier cas il est courant de faire appel à des logiciels, de type gestion de production par exemple, ou à des méthodes d'optimisation dont l'objectif est de déterminer la meilleure décision pour optimiser un ou plusieurs coûts, en prenant en compte des contraintes. Dans le second cas, la complexité et l'imprévisibilité de l'évolution dynamique du système de production nécessite souvent de prendre des décisions de pilotage « en ligne », en temps réel qui nécessite la connaissance de l'état du

système et qui permet d'évaluer les conséquences des décisions que l'on envisage de prendre immédiatement (Pujo et al. 2002a, Pujo et al. 2002b), alors, nous pouvons considérer que les mots clés en pilotage en temps réel sont : acquisition de données, analyse rapide et réaction quasi instantanée (Habchi 2001).

Ainsi, nous ne essayons pas de construire un système de pilotage global capable de prendre toute décision nécessaire au bon fonctionnement de l'atelier (stratégies de production, planification,…), mais, nous proposons une approche de prise de décisions au niveau opérationnel en temps réel qui prend en compte la dynamique de l'atelier.

3.2. Les types de décisions prises en temps réel

Il existe plusieurs décisions que nous pouvons prendre lorsqu'on se positionne dans le cas de prise de décisions en temps réel.

D'après Berchet (Berchet 2000) il existe plusieurs types de prise de décisions en temps réel. Elle s'est inspirée des travaux de Pierreval (Pierreval 1999), qui présente une typologie pour faciliter l'identification de ces décisions lors de l'analyse de l'atelier et faciliter leur représentation dans un modèle :

3.2.1. Les décisions d'affectation

Ces décisions résultent de la souplesse introduite au niveau du choix des ressources. Elles sont introduites par la flexibilité physique du système. Elles ont pour conséquence de réduire l'ensemble des ressources pouvant être concernées par l'exécution finale d'une tâche (Figure 3). Par exemple, lorsqu'une pièce a le choix entre plusieurs ressources différentes, la pièce doit à un certain moment être affectée à une de ces ressources. Lorsque plusieurs chariots filo-guidés sont disponibles pour effectuer une opération de transport d'une pièce qui vient d'être déterminée, il faut choisir celui à appeler. Ces décisions peuvent être prises à priori, à n'importe quel moment. Cependant, au regard de la simulation, il est important de noter que l'effet de la décision n'interviendra que lors d'événements prédéterminés, comme l'arrivée d'un nouvel ordre de fabrication dans l'atelier ou la fin de l'exécution d'une opération.

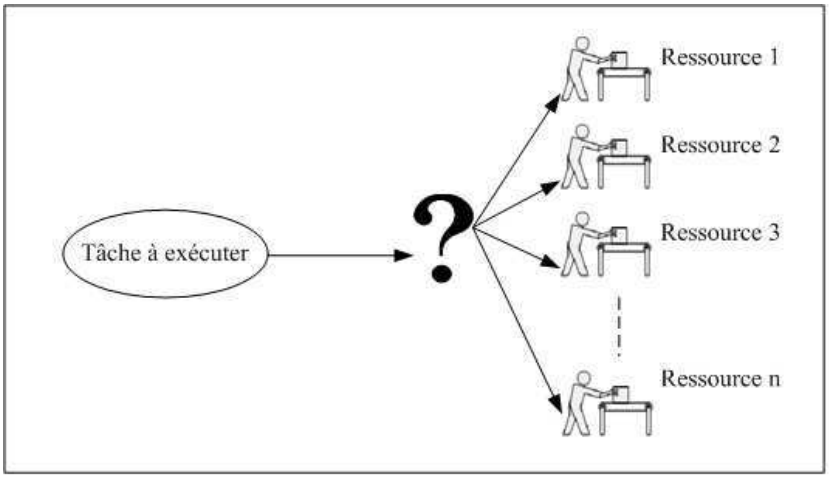


Figure 3. Décision d'affectation mono-ressource

Lorsqu'un seul type de ressource est requis l'affectation est mono-ressource (Figure 3), tandis qu'elle sera multi-ressource (Figure 4) si plusieurs types sont impliqués (machines, outils, palettes, opérateurs, etc.). Par exemple, en sous traitance mécanique, il peut être nécessaire de décider quelle machine va être utilisée mais également quel opérateur effectuera le travail.

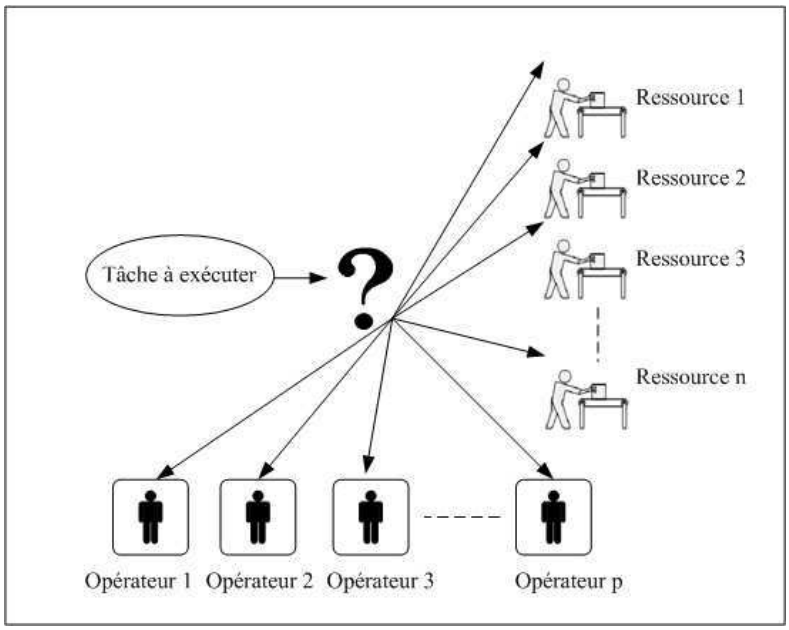


Figure 4. Décision d'affectation multi-ressources

3.2.2. Les décisions d'attribution

Ce type de décision survient lorsqu'à un instant donné, une ou plusieurs ressources ont le choix entre différents travaux à exécuter. Il est alors nécessaire de décider quel travail exécuter d'abord.

C’est le cas par exemple d’un opérateur qui, après avoir achevé une tâche doit choisir dans son stock amont la nouvelle pièce à usiner. Ces décisions résultent de la souplesse introduite par les en-cours au niveau du choix de l’ordre de succession des travaux liés à une ou plusieurs ressources. Ces décisions sont introduites par la flexibilité temporelle du système. Elles entraînent la réservation de ressources par une entité à partir d’un moment donné. De nombreuses règles de priorité ont été proposées et testées dans la littérature, pour les décisions d’attribution simples (Figure 5).

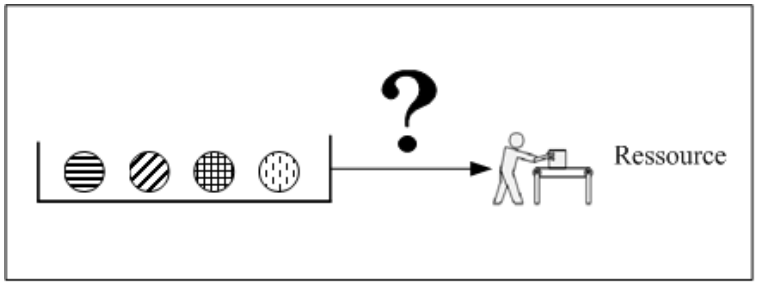


Figure 5. Décision d’attribution simple

Dans certaines situations, plusieurs travaux doivent être sélectionnés simultanément (Figure 6). C’est le cas par exemple d’une ressource de manutention qui doit transporter plusieurs palettes afin de « rentabiliser » son déplacement, ou d’un four de traitement thermique qui doit être rempli « au mieux » avant d’être démarré.

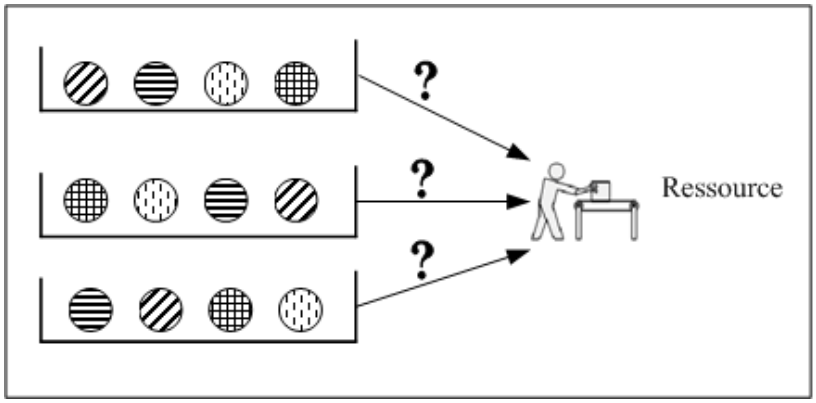


Figure 6. Décision d’attribution multiple

Ces décisions peuvent être prises à différents instants mais elles deviendront effectives que lors de la souvenance de l’événement de libération d’une ressource.

3.2.3. Les décisions de remise en cause

Un certain nombre de décisions prises sur le fonctionnement de l'atelier peuvent nécessiter une remise en cause, afin notamment d'éviter que les conséquences sur la production de perturbations survenues de façon imprévisibles soient trop importantes. L'arrivée de la perturbation (événement) va donc déclencher un choix entre plusieurs actions (activités) possibles (figure 7). La décision peut consister à ne rien modifier ou au contraire reconsidérer des activités initialement prévues (transformation de A_i en A'_k), par exemple : faire effectuer un travail par une autre machine ou faire passer un travail en priorité, etc.

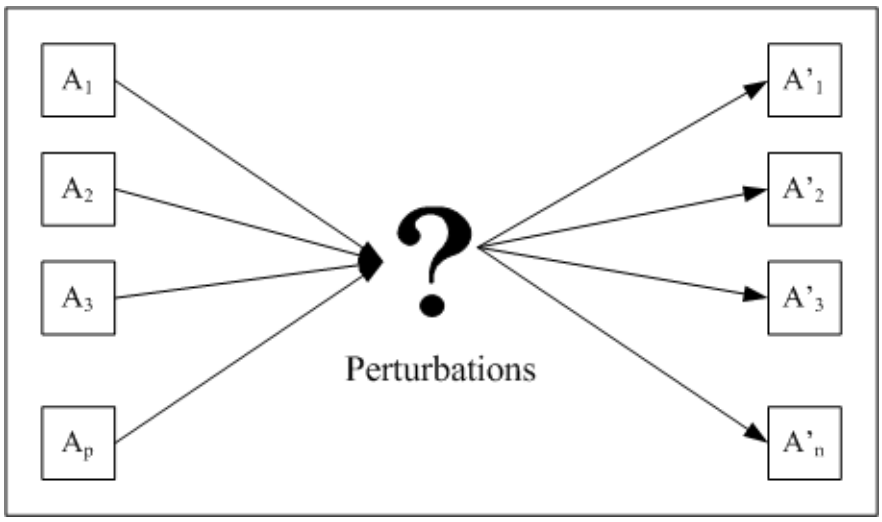


Figure 7. Réactions aux perturbations

L'arrivée d'une commande urgente relève également du même type de décision. Certaines remises en cause peuvent également provenir de dérives successives de l'atelier (Figure 8) vers un état non souhaitable (encombrement de machines, goulots, retards, etc.), qu'il convient d'essayer de corriger en reconsidérant le déroulement de certaines activités. Par exemple la formation d'un goulot peut entrainer des réaffectations de ressources afin de décharger la ressource critique. Aussi, la panne d'une ressource critique peut engendrer la révision globale de l'ordonnancement prévu, voire le recours à des heures supplémentaires ou de la sous-traitance et donc modifier des décisions prises initialement avant la fabrication.

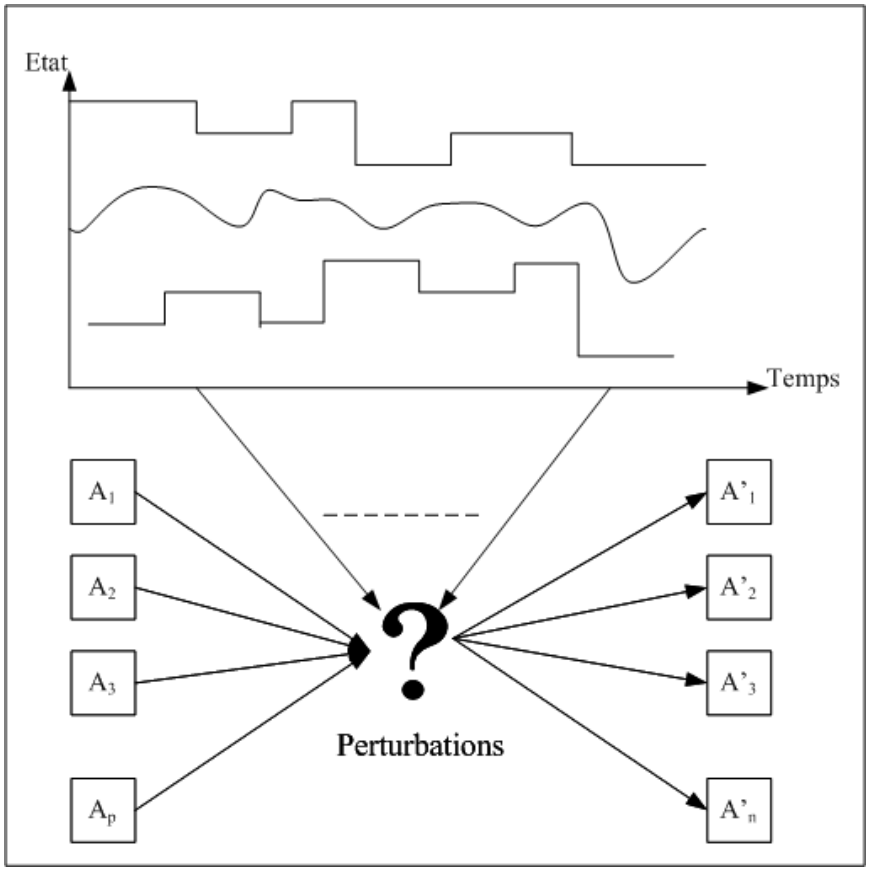


Figure 8. Remise en cause suite à des dérives

3.2.4. Les décisions d'anticipation

Elles proviennent de l'initiative d'effectuer certaines activités, à un moment où les couts qui résultent (temps, matière, énergie, etc.), sont plus faibles que si elles avaient été effectuées plus tard. Par exemple, des montages ou démontages d'outils en temps masqué sont des décisions d'anticipation. Souvent, elles font références à un événement spécifique tel que le démarrage d'une opération de fabrication (pour laquelle les outils devront être montés et régler par exemple). Compte tenu des aléas nombreux qui régissent la vie d'un atelier, ces décisions ne peuvent souvent pas être prises à l'avance de façon statique. Il est donc nécessaire de décider en temps réel le bon instant pour mettre un outil en chauffe, par exemple dans une forge, afin qu'il arrive à la bonne température lorsque l'on voudra l'utiliser, sachant que si on arrive trop tard une attente sera générée et s'il arrive trop tôt il refroidira. Certains ordres de réapprovisionnement (en composants par exemple) entrent dans cette catégorie.

Ces décisions ne sont en général pas déclenchées par un événement. Il s'agit donc de déterminer, lors du déroulement de la simulation et des changements d'état, l'instant propice.

3.2.5. Les décisions de déclenchement des activités de maintenance ou d'entretien

Dans un certain nombre de cas, ces décisions ne peuvent être planifiées convenablement à l'avance et sont prises en temps réel afin de conserver l'outil de production dans son état de performance nominale (en productivité et qualité). Ces décisions sont importantes dans la mesure où elles sont de nature à immobiliser un certain nombre de ressources. Il s'agit donc de choisir l'instant où elles doivent être effectuées. Trop tôt, elles peuvent perturber inutilement la production, tandis que trop tard, elles risquent d'entrainer une dégradation des installations, ou de la qualité de la production. Le déclenchement d'actions de maintenance doit aussi prendre en compte le caractère urgent de certaines commandes. Un compromis doit en général être établi, sur le moment où elles doivent être mises en œuvre. Par exemple, les fours électroniques d'une fonderie doivent être nettoyés (décrassés) périodiquement, sous peine d'entrainer de forte diminution de qualité. Ces opérations sont effectuées chaque fois que c'est possible lorsque des arrêts de production surviennent (ex. pannes). Néanmoins, il est souvent nécessaire d'interrompre l'activité d'un four pendant la production.

Ces décisions peuvent entrer dans la catégorie des décisions d'anticipation si elles se reflètent à un événement spécifique. Dans le cas le plus général, l'instant de déclenchement est déterminé par rapport à l'état du système (disponibilité de machines de substitution, ordre en retards, stocks disponibles, etc.) (Figure 9).

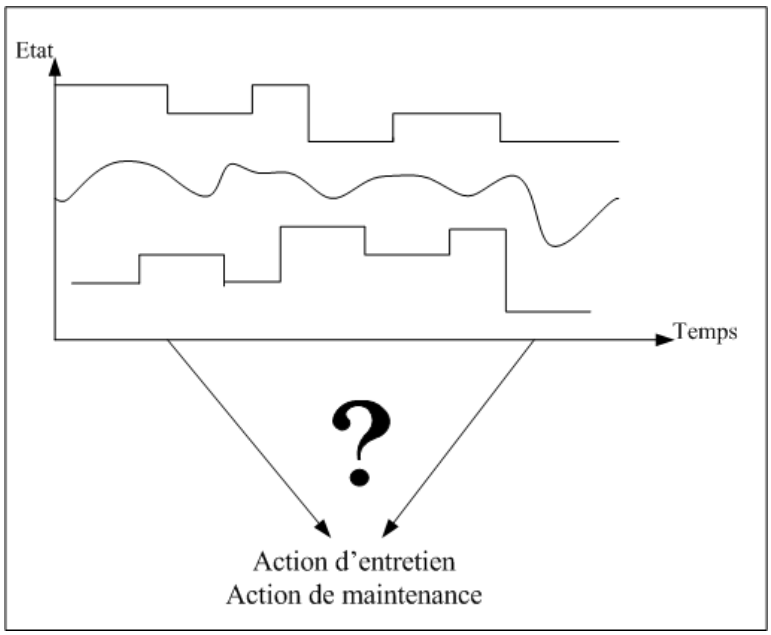


Figure 9. Déclenchement d'actions de maintenance ou entretien

3.2.6. Décisions de régulation et d'ajustement

Ces décisions visent à exercer un certain contrôle sur des flux en vue de les ralentir ou les accélérer. Elles découlent de la possibilité technologique de certaines machines d'ajuster leur débit, ou de la capacité de certains systèmes de se réguler, ce qui autorise plus de flexibilité dans le pilotage du flux. Par exemple, le réglage du débit d'un cubilot de fonderie doit être ajusté régulièrement. Un débit important permet de satisfaire sans interruption la demande en fonte liquide, mais en contrepartie, peut conduire à vider un four et entrainer une rupture d'approvisionnement, mais induit des attentes de livraison plus longues.

L'ajout ou le retrait de cartes kanban pour s'adapter à de nouvelles circonstances ou l'augmentation et la diminution de tailles de lots de transferts sont aussi des décisions de régulation.

3.2.7. Combinaison de différents types de décisions

Le pilotage d'une unité de production complexe engendre des situations où des différents types de décision évoqués précédemment peuvent se combiner. Ainsi, les décisions d'attribution et d'affectation peuvent s'enchaîner. La figure 10 montre une situation où plusieurs ressources étant disponibles et plusieurs opérations à effectuer, des décisions d'affectation et d'attribution doivent être prises.

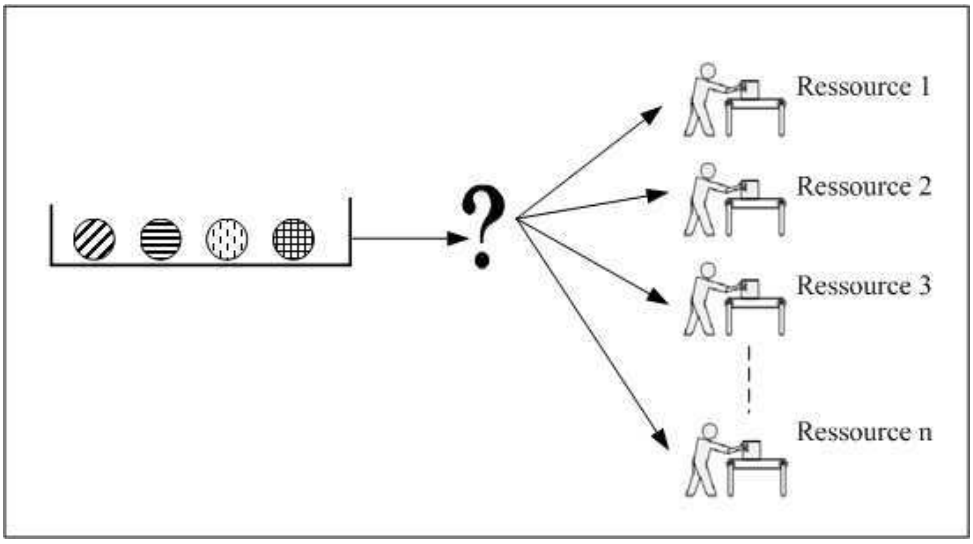


Figure 10. Combinaison affectation/ attribution

3.3. Les critères de prise de décision

Les décisions prises en temps réel dans un atelier ont un impact souvent très important sur les flux des produits. Une des caractéristiques les plus importantes à prendre en compte lors du pilotage en temps réel est la dynamique du système de décision. Pour prendre les décisions nécessaires, on doit alors disposer d'un maximum d'informations permettant de refléter la situation lors de la prise de décision dans l'atelier.

L'objectif est de permettre au gestionnaires de piloter un système manufacturier discret complexe, étant donné un ensemble d'objectifs et de contraintes, de la même manière qu'il est possible de piloter un système dynamique continu, comme une centrale nucléaire ou une automobile, c'est-à-dire en surveillant l'évolution d'un nombre "restreint" de paramètres clés. La difficulté consiste à découvrir ces paramètres clés et à évaluer leur incidence sur le système de production de manière à pouvoir choisir les actions déterminantes, suivant les circonstances, afin que le système évolue dans la direction désirée.

Cependant, les décisions en temps réel dans les systèmes de production dépendent de plusieurs critères. Il est nécessaire de prendre en compte les caractéristiques de l'atelier, tels que le nombre de machines, le nombre d'opérateurs sur chaque machine, les temps de livraison des pièces, le taux de chargement de l'atelier, etc. Aussi, il est essentiel de caractériser l'état courant de l'atelier qui change et évolue constamment dans le temps, il est donc primordial de déterminer les variables qui reflètent au mieux l'état de l'atelier et sa dynamique.

Pierreval en 1999 (Pierreval 1999) a proposé un formalisme de prise de décisions en temps réel ; il a proposé un vecteur de 7-uplets (ID, A, C, H, R, AL, DU) pour caractériser la prise de décision en temps réel :

ID : Les données initiales utilisées dans le processus de prise de décisions : les variables d'état du système et les objectifs de production, etc.

A : L'ensemble des alternatives possibles dans lequel un choix doit être fait à la fin de la prise de décision.

C : Un point ou une condition de prise de décision. Généralement, trois situations peuvent se présenter : un point de décision auquel nous déclenchons la prise de décision (une machine se

libère), un point décision qui dépend d'une période de temps régulière (activité de maintenance), et enfin, un point décision qui dépend d'un ensemble de conditions à remplir, pour cela le système doit être scanné pour pouvoir détecter quand les conditions exigées sont satisfaites.

H : L'horizon de la décision : pour chaque décision nous considérons une période de temps de sa validité (une machine est sélectionnée pour effectuer un job pour une durée de douze heures et une autre décision doit être prise pour sélectionner la machine à la fin de cette période de temps). La décision dure jusqu'au déclenchement d'un événement donné.

R : Les ressources acquises pour chaque type de décision.

AL : La logique associée à la prise de décision, cette logique peut être aussi simple que celle utilisée dans des règles de priorité dans les files d'attente ou bien très complexe comme celle qui est basée sur l'expertise humaine.

DU : La durée de la prise de décision.

L'objectif de cette typologie est de faciliter la prise des décisions en temps réel et de contribuer à l'identification des caractéristiques du système de production nécessaires au bon fonctionnement du processus de pilotage.

Par ailleurs, il existe plusieurs approches dans la littérature destinées à aider les prises de décisions en temps réel de sorte à ce qu'elles concourent de la meilleure façon aux objectifs de l'entreprise, on peut en distinguer plusieurs. Parmi les plus connues on rencontre : les approches basées sur des heuristiques, les approches basées sur des modèles dynamiques, utilisés en ligne et les approches basées sur l'intelligence artificielle. Les approches basées sur l'intelligence artificielle sont destinées à fournir une aide « intelligente » afin d'aider la prise de décision ou de prendre les décisions de façon automatique. Cette intelligence peut être destinée à résoudre un problème localisé (systèmes experts, systèmes à base de connaissance) (e. g., Pierreval 1992, Monostori 2003) ou des problèmes distribués (systèmes multi agents) (e. g., Roy et al. 2001).

Le problème abordé ici consiste donc à déterminer comment nous pouvons aider à prendre les décisions pour le pilotage des systèmes de production à partir de l'apprentissage ?

L'intelligence est généralement issue de connaissances « expertes » et est éventuellement capable de prendre en compte de l'imprécision (approches floues, possibilistes). La difficulté connue des approches basées sur l'apprentissage réside dans l'élicitation des connaissances dont doivent être dotée ces systèmes. En effet, la recherche de connaissances pertinentes sur lesquelles est basé l'apprentissage (Huyet 2004, Zhao et al. 2001, Metan et al. 2005, Shiue et al. 2006) pour piloter efficacement constitue un écueil majeur pour la mise en œuvre du pilotage intelligent d'ateliers.

4. Conclusion

Dans ce chapitre nous avons présenté une vue globale du pilotage des systèmes de production et quelques approches de pilotage présentées dans la littérature (pilotage centralisé, pilotage hiérarchisé, pilotage coordonné, pilotage distribué, pilotage distribué supervisé, pilotage holonique, pilotage par le produit). Nous avons déterminé le cadre dans lequel nous nous situons : prise de décision au niveau opérationnel et en temps réel. Dans ce chapitre nous avons montré aussi que le pilotage en temps réel des systèmes de production nécessite une étude approfondie des différents critères de prise de décision tels que le nombre de machines, le nombre d'opérateurs sur chaque machine, les temps de livraison des pièces, le taux de chargement de l'atelier, etc.

Plusieurs approches dans la littérature sont proposées pour aider au pilotage en temps réel des systèmes, parmi les approches les plus connues : les approches basées sur l'intelligence artificielle. Ces approches nécessitent des connaissances expertes. Ces connaissances doivent notamment permettre de déterminer comment affecter au mieux les moyens de production (machines, transport, opérateurs, outils, etc.) de sorte à satisfaire aux objectifs de production (e. g. minimiser les en-cours). Dans la mesure où le pilotage s'effectue sur la base d'un état du système changeant dynamiquement au cours du temps, il est très difficile de disposer d'exemples des « meilleures pratiques » pour le pilotage.

Nous proposons donc une approche d'apprentissage autonome capable de générer sans expertise préalable les connaissances nécessaires pour doter un système informatique de la capacité à décider en temps réel.

Pour cela, nous présentons dans le chapitre suivant un état de l'art des travaux menés en apprentissage par simulation et leurs limites.

Chapitre II

L'APPRENTISSAGE AUTOMATIQUE PAR SIMULATION DANS LES SYSTEMES DE PRODUCTION

1. Introduction

Il existe un passage naturel des sciences de la décision à l'intelligence artificielle ; la bonne décision ne se découvre pas, *la bonne décision est un construit* qui s'élabore en relation avec des finalités, avec un contexte, avec des capacités d'appropriation par le milieu. De ce fait, le développement de l'aide à la décision qui prend appui sur des modes de raisonnement, des modèles, des algorithmes plus ou moins spécifiques doit laisser place à une assez grande flexibilité, à une assez grande ouverture. L'aide scientifique à la décision ne permet pas de postuler, comme on le fait couramment dans les sciences dures, qu'il existe quelque part une vérité. C'est pourquoi la démarche à suivre n'est pas une démarche de découverte mais une démarche de construction. De ce fait, dans ces conditions, ce qui a été fait en intelligence artificielle et notamment dans toute la voie de *l'apprentissage automatique* est très positif.

L'intelligence est généralement issue de connaissances « expertes » et est éventuellement capable de prendre en compte de l'imprécision tel que dans les approches floues ou les approches possibilistes. La difficulté connue de ces approches réside dans l'élicitation des connaissances dont doivent être dotée ces systèmes. En effet, la recherche de connaissances pertinentes (Huyet 2004, Zhao et al. 2001, Metan et al. 2005, Shiue et al. 2006) pour piloter efficacement constitue un écueil majeur pour la mise en œuvre du pilotage intelligent d'ateliers.

Ces connaissances doivent notamment être capables de déterminer comment affecter au mieux les moyens de production (machines, transport, opérateurs, outils, etc.) de sorte à satisfaire aux objectifs de production (e. g. minimiser les en-cours). Dans la mesure où le pilotage s'effectue

sur la base d'un état du système changeant dynamiquement au cours du temps, il est très difficile de disposer d'exemples des « meilleures pratiques » pour le pilotage.

D'abord, nous présentons quelques approches d'apprentissage automatique et nous montrons l'importance de l'élicitation des exemples d'apprentissage pour l'apprentissage de ses méthodes.

Ensuite, nous présentons l'intérêt de la simulation pour l'apprentissage pour le pilotage des systèmes de production.

Enfin nous présentons les différents travaux dans le domaine de l'apprentissage par simulation des systèmes de production.

2. L'apprentissage automatique

Les travaux de recherche en apprentissage, relayés ensuite par ceux en fouille de données, ont trouvé leur essor avec l'émergence des principes d'intelligence artificielle dans les années 1980 (Aler et al. 2002). L'objectif de ces travaux est globalement d'exhiber des connaissances à partir de données, notamment des bases d'exemples ou des corpus (Sebag et al. 2002, Cornuéjols et al. 2002). Ces connaissances peuvent prendre des formes variées (règles de production, règles floues, arbres de décision, etc.).

L'apprentissage automatique (ou machine-learning) fait référence au développement, à l'analyse et à l'implémentation de méthodes qui permettent à une machine (au sens large) d'évoluer grâce à un processus d'apprentissage, et ainsi de remplir des tâches qu'il est difficile ou impossible de remplir par des moyens algorithmiques plus classiques. Et d'après Cornuéjols et Miclet (Cornuéjols et al. 2003) : *« La notion de l'apprentissage automatique englobe toute méthode permettant de construire un modèle de la réalité à partir de données, soit en améliorant un modèle partiel ou moins général, soit en créant complètement le modèle. Il existe deux tendances principales en apprentissage, celle issue de l'intelligence artificielle et qualifiée de symbolique (arbre de décision,…), et celle issue des statistiques et qualifiée de numérique (réseau de neurones,…). »*

Il existe plusieurs méthodes d'apprentissage automatique tel que l'analyse discriminante linéaire, les machines à vecteurs support (SVM), les réseaux de neurones, les méthodes d'induction, etc.

2.1. **Méthode d'induction**

L'apprentissage par induction reste toujours une des principales méthodes étudiées dans le domaine de l'apprentissage automatique (Santos Osório, 1998). Dans cette approche, on cherche à acquérir des règles générales qui représentent les connaissances obtenues à partir d'exemples. Les règles ainsi obtenues peuvent être représentées d'une façon explicite (facilement interprétables) ou d'une façon implicite avec un codage qui n'est pas toujours facile à interpréter. L'algorithme d'apprentissage par induction reçoit un ensemble d'exemples d'apprentissage et doit produire des règles de classification, permettant de classer les nouveaux exemples. Le processus d'apprentissage cherche à créer une représentation plus générale des exemples, selon une *méthode de généralisation de connaissances*. Ce type de méthodes est aussi appelé apprentissage de concepts ou bien acquisition de concepts. Parmi les approches d'apprentissage empirique par induction les plus connues, on trouve les réseaux de neurones artificiels et les arbres de décision. L'algorithme d'apprentissage par induction peut fonctionner de façon *supervisée* ou *nonsupervisée*:

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

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

Le premier programme d'apprentissage par graphe d'induction a été réalisé par Sonquist et Morgan (Sonquist et al. 1963). L'objectif des méthodes à base de graphe d'induction est de construire une fonction de classement représentable par un graphe : le graphe est construit en partant de la racine et en allant vers les feuilles. On parle de méthode d'induction descendante et on trouve, dans la littérature anglaise, le terme de TDIDT (Top Down Induction of Decision Trees) afin de qualifier les méthodes à base de graphe d'induction. Le principe général des

graphes d'induction s'exprime ainsi: Chercher à discriminer les exemples selon leur classe en fonction d'attributs considérés comme les « meilleurs » parmi tous les autres au sens d'un critère donné. Sur l'échantillon d'apprentissage, est recherché l'attribut qui discrimine « le mieux » les exemples. Puis sont constitués autant de sous - échantillons que l'attribut possède de valeurs (chaque sous- échantillon contient les exemples ayant la valeur associée de l'attribut) et sur chaque sous- échantillon est recherché, de nouveau, le « meilleur » attribut. Ce processus récursif est réitéré jusqu'à ce que les exemples d'un même sous - échantillon possèdent tous la même classe.

Dans un graphe d'induction, chaque chemin correspond à une règle exprimée sous la forme «si condition alors conclusion» dans laquelle «condition» désigne une ou plusieurs propositions logiques de type « attribut, valeur » (Zighed et Rakotomalala 2000). L'ensemble des règles constitue ainsi le modèle de prédiction.

Parmi les formalismes des graphes d'induction nous citons les arbres de décision. Ils *s*ont composés d'une structure hiérarchique en forme d'arbre. Cette structure est construite grâce à des méthodes d'apprentissage par induction à partir d'exemples. L'arbre ainsi obtenu représente une fonction qui fait la classification d'exemples, en s'appuyant sur les connaissances induites à partir d'une base d'apprentissage. En raison de cela, ils sont aussi appelés arbres d'induction (*Induction Decision Trees*).

Traditionnellement, un arbre de décision se construit à partir d'un ensemble d'apprentissage par raffinements de sa structure. Un ensemble de questions sur les attributs est construit afin de partitionner l'ensemble d'apprentissage en sous-ensembles qui deviennent de plus en plus petits jusqu'à ne contenir à la fin que des observations relatives à une seule classe. Les résultats des tests forment les branches de l'arbre et chaque sous-ensemble en forme les feuilles. Le classement d'un nouvel exemple se fait en parcourant un chemin qui part de la racine pour aboutir à une feuille : l'exemple appartient à la classe qui correspond aux exemples de la feuille.

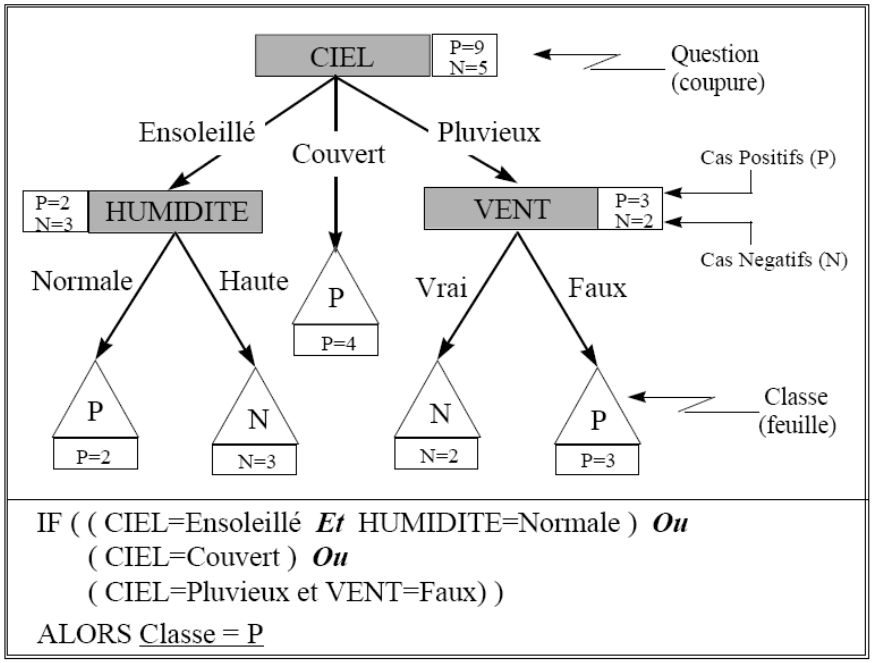


Figure 11. Exemple d'Arbre de Décision Simple (Santos Osório 1998)

La Figure 11 donne un exemple d'arbre de décision pour le classement d'un ensemble de cas, avec un test d'appartenance à une classe (Santos Osório 1998). Dans ce cas particulier, les cas dits *positifs* sont ceux qui appartiennent à la classe et les cas dits *négatifs* sont ceux qui n'y appartiennent pas.

Tableau 1. Ensemble d'Apprentissage : Conditions Météorologiques

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

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

2.2. Les machines à vecteurs support (SVM)

Les machines à vecteurs de support ou séparateurs à vaste marge (en anglais *Support Vector Machine*, SVM), développées originellement par Vapnik dans les années 1980 et 1990, sont un ensemble de techniques d'apprentissage supervisé destinées à résoudre des problèmes de discrimination et de régression. Les SVM sont une généralisation des classificateurs linéaires.

Le classifieur SVM a été conçu pour une séparation de deux ensembles de données. Il est considéré donc comme un classificateur binaire. Le but des SVM est de trouver un hyperplan qui va séparer et maximiser la marge de séparation entre deux classes.

A l'origine conçues pour les tâches de classification ou reconnaissance de formes, les SVM permettent également de traiter les problèmes de régression non linéaire.

Le principe théorique des SVM comporte deux points fondamentaux :

- La transformation non linéaire (Φ) des exemples de l'espace d'entrée vers un espace dit de re-description de grande dimension muni d'un produit scalaire (espace de Hilbert),

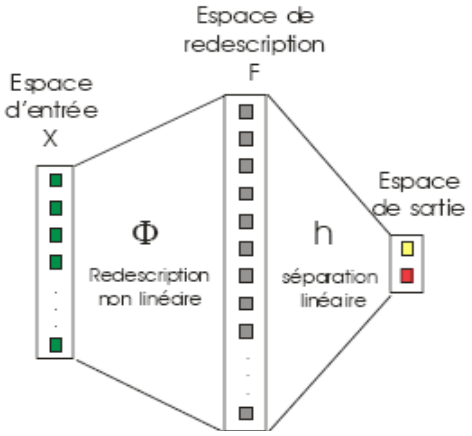


Figure 12. La transformation non linéaire des données peut permettre une séparation linéaire des exemples dans un nouvel espace. Adapté de Cornuéjols et al. 2002.

- La détermination d'un hyperplan permettant une séparation linéaire optimale dans cet espace de grande dimension.

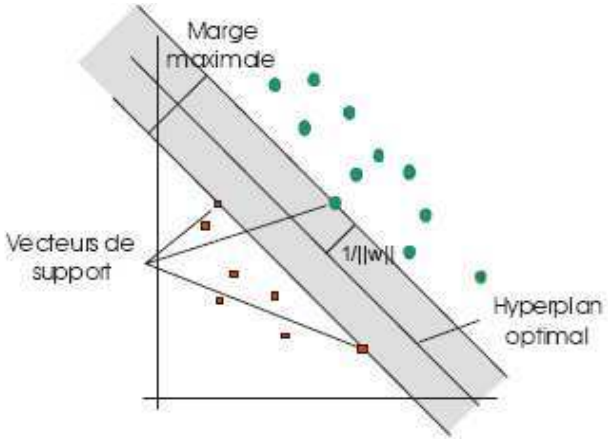


Figure 13. L'hyperplan séparateur optimal est celui qui maximise la marge dans l'espace de redescription.

Le problème de recherche de l'hyperplan séparateur possède une formulation duale. Ceci est particulièrement intéressant car, sous cette formulation duale, le problème peut être résolu au moyen de méthode d'optimisation quadratique.

2.3. Les réseaux de neurones

Les réseaux de neurones sont des assemblages fortement connectés d'unités de calcul, les *neurones formels*. Ils ont pour origine un modèle du *neurone biologique* et ont vocation à imiter certains mécanismes du cerveau humain. *Un neurone formel* est une fonction algébrique non linéaire et bornée, dont la valeur dépend de paramètres appelés coefficients ou poids. Les variables de cette fonction sont habituellement appelées "entrées" du neurone, et la valeur de la fonction est appelée sa "sortie".

Un neurone est donc avant tout un opérateur mathématique, dont on peut calculer la valeur numérique. Il est courant de représenter graphiquement un neurone comme indiqué sur la Figure14.

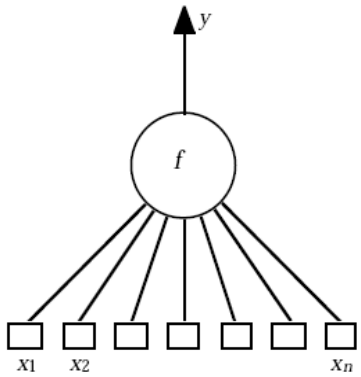


Figure 14. Un neurone formel

Un neurone réalise une fonction non linéaire bornée $y = f(x1, x2, ...xn ; c1, c2, ... c_p)$ où les $\{x_i\}$ sont les variables et les $\{c_j\}$ sont des paramètres (Dreyfus et al. 2004). Pour des raisons que nous expliquerons plus loin, les neurones les plus fréquemment utilisés sont ceux pour lesquels la fonction f est une fonction non linéaire (généralement une tangente hyperbolique) d'une combinaison linéaire des entrées :

$$y = \tanh\left[\sum_{i=1}^{n} c_i x_i\right] \qquad \textbf{(1)}$$

Les $\{x_i\}$ sont les variables (ou entrées) du neurone, les $\{c_i\}$ sont des paramètres ajustables. Un neurone formel ne réalise donc rien d'autre qu'une somme pondérée suivie d'une non-linéarité.

C'est l'association de tels éléments simples sous la forme de *réseaux* qui permet de réaliser des fonctions utiles pour des applications industrielles.

On distingue deux grands types d'architectures de réseaux de neurones : les réseaux de neurones *non bouclés* et les réseaux de neurones *bouclés.*

2.3.1. Les réseaux de neurones non bouclés :

Un réseau de neurones non bouclé est représenté graphiquement par un ensemble de neurones "connectés" entre eux, l'information circulant des entrées vers les sorties sans "retour en arrière" ; si l'on représente le réseau comme un graphe dont les nœuds sont les neurones et les arêtes les "connexions" entre ceux-ci, le graphe d'un réseau non bouclé est *acyclique.*

La Figure 15 représente un réseau de neurones non bouclé qui a une structure particulière, très fréquemment utilisée : il comprend des entrées, une ou plusieurs couches de neurones "cachés" et des neurones de sortie. Les neurones de la même couche cachée ne sont pas connectés entre eux. Cette structure est appelée *Perceptron multicouche.*

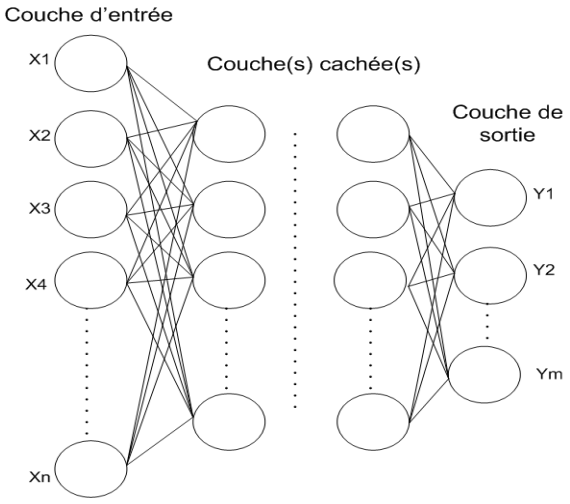


Figure 15. Un modèle de réseau de neurones

Les réseaux de neurones non bouclés sont des systèmes statiques, utilisés principalement pour effectuer des tâches de classification, ou de modélisation statique de processus ; l'opérateur réalisé par un réseau de neurones non bouclé (relation entrées-sorties) est une fonction algébrique.

2.3.2. Les réseaux de neurones bouclés :

Contrairement aux réseaux de neurones non bouclés dont le graphe de connexions est acyclique les réseaux bouclés, dont le graphe des connexions contient des cycles (Figure 16) ; ce sont des systèmes dynamiques, utilisés comme filtres non linéaires, ainsi que pour la modélisation et la commande de processus ; l'opérateur réalisé par un réseau bouclé est un ensemble d'équations aux différences couplées.

Un réseau de neurones bouclé à temps discret est régi par une (ou plusieurs) équations aux différences non linéaires, résultant de la composition des fonctions réalisées par chacun des neurones.

La forme la plus générale des équations régissant un réseau de neurones bouclé est appelée *forme canonique ;*

$$x(k+1) = \varphi[x(k), u(k)] \quad \textbf{(2)}$$

$$Y(k) = \psi[x(k), u(k)] \quad \textbf{(3)}$$

Où φ et ψ sont des fonctions non linéaires réalisées par un réseau de neurones non bouclé (par exemple, mais pas obligatoirement, un Perceptron multicouche), et où k désigne le temps (discret).

La forme canonique est représentée sur la Figure 16. Tout réseau de neurones, aussi compliqué soit-il, peut être mis sous cette forme canonique, de manière complètement automatique.

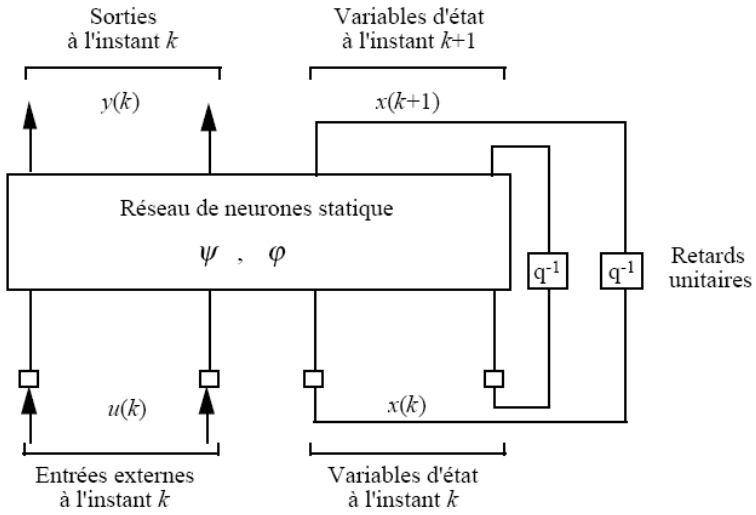


Figure 16. Forme canonique d'un réseau de neurones bouclé

Les réseaux de neurones bouclés sont utilisés pour effectuer des tâches de *modélisation de systèmes dynamiques, de commande de processus, ou de filtrage.*

Afin que le réseau de neurones soit capable de prédire les valeurs de sorties en fonction des valeurs d'entrées, il doit passer par une phase d'apprentissage.

2.3.3. L'apprentissage des réseaux de neurones

L'apprentissage est en général un processus graduel, itératif, où les pondérations des connexions entre les neurones du réseau de neurones : poids, sont modifiés plusieurs fois selon une règle d'apprentissage avant d'atteindre leurs valeurs finales. Les approches d'apprentissage connexionniste peuvent être reparties en trois grandes classes, selon le degré de contrôle donné à l'utilisateur :

i) Apprentissage Supervisé :

L'utilisateur dispose d'un comportement de référence précis qu'il désire faire apprendre au réseau. Le réseau est donc capable de mesurer la différence entre son comportement actuel et le

comportement de référence, et de corriger ses poids de façon à réduire cette erreur. L'apprentissage supervisé utilise des connaissances empiriques, habituellement représentées par des ensembles d'exemples étiquetés par la classe à laquelle ils appartiennent.

***ii) Apprentissage Semi-Supervisé* :**

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

***iii) Apprentissage non-supervisé* :**

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

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

2.4. Méthodes statistiques : Naive Bayes

L'apprentissage statistique désigne un vaste ensemble de méthodes et d'algorithmes permettant dans un sens général d'apprendre des logiques (comportements) à partir d'exemples. Cet apprentissage se base sur l'évaluation mathématique des exemples. Des stratégies de réduction d'erreurs sont ensuite calculées à partir de cette évaluation.

L'algorithme le plus connu de cette catégorie est "***Naive Bayes***". Il permet de faire de l'apprentissage supervisé en se basant sur le théorème de Bayes:

$$P(A, B) = P(A) * P(B|A) \quad \textbf{(4)}$$

$$\text{De là } P(B|A) = P(B) * P(A|B) / P(A) \quad \textbf{(5)}$$

Lorsqu'on tente de prédire une "classe" B en fonction d'attributs connus A (qu'on suppose indépendants) on énumère les cas correspondants figurant parmi les exemples connus.

Les réseaux probabilistes bayesiens, également appropriés à l'apprentissage dans les systèmes multi-agents (Maes et al. 2004), cependant leur apprentissage est très coûteux en termes de temps de calcul.

2.5. Méthodes Hybrides d'Apprentissage Automatique

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

2.5.1. **Systèmes Symbolique-Flous** :

Les systèmes symbolique-flous intègrent la logique floue (*fuzzy-logic*) et les systèmes à base de connaissances (*KBS*). Ce type de système est en réalité une extension des systèmes à base de connaissances, dans lesquels on ajoute la possibilité de représenter des règles floues et de les manipuler à travers des mécanismes d'inférence floue (Santos Osório 1998).

2.5.2. Systèmes Symbolique-Génétiques :

Ces systèmes sont normalement composés d'un module génétique responsable de l'acquisition de connaissances à partir des données (apprentissage), et d'un module symbolique responsable du moteur d'inférence symbolique (raisonnement).

2.5.3. Systèmes Neuro-Flous :

Les systèmes hybrides neuro-flous sont une des catégories de systèmes hybrides les plus développées car la logique floue et les réseaux connexionnistes ont beaucoup de points en commun. Les systèmes neuro-flous sont principalement de trois types : systèmes qui intègrent des règles floues dans des réseaux (le modèle RBF étant un des plus utilisés) ; les systèmes qui font l'extraction de règles floues à partir des réseaux ; et les systèmes qui implémentent des neurones flous .

Et beaucoup d'autres approches hybrides dont nous ne pouvons pas donner une liste exhaustive telles que les systèmes qui intègrent un réseau connexionniste et un système de raisonnent fondé sur des cas (Systèmes Neuro-CBR), les systèmes qui intègrent des réseaux connexionnistes et des systèmes à base de connaissances (KBS) (Systèmes Neuro-KBS ou SHNS), etc (Santos Osório 1998).

3. L'apprentissage basé sur la simulation pour le pilotage des systèmes de production

3.1. Le rôle de la simulation dans l'apprentissage pour le pilotage des systèmes de productions

Dans le cadre de nos recherches, nous nous attacherons essentiellement aux méthodes issues de l'apprentissage permettant d'extraire des règles de fonctionnement sur le comportement des systèmes de production. Cependant la difficulté connue de ces approches d'apprentissage automatique réside dans l'élicitation des connaissances dont doivent être dotée ces systèmes. Et la recherche des bons exemples qui reflètent au mieux le comportement étudié du système pour piloter efficacement, constitue un écueil majeur pour la mise en œuvre du pilotage intelligent d'ateliers. Il est très difficile d'acquérir des exemples d'apprentissage qui sont adaptés au problème étudié dans le système de production.

Néanmoins, les travaux existants dans la littérature se basent globalement sur des exemples d'apprentissage pour générer ces connaissances.

Plusieurs approches ont été proposées pour constituer des ensembles d'apprentissage à partir desquels on peut apprendre :

(1) méthode d'échantillonnage (Pierreval et al. 1990, Li et al. 2007).

(2) connexion avec un algorithme d'optimisation (Huyet et al. 2004).

(3) Exemples provenant des pratiques des opérateurs des systèmes réels (Zhao et al., 2001).

De même, l'usage de la simulation permet d'être à la source de génération de connaissances nouvelles. Plusieurs auteurs ont montré que la simulation était capable de fournir des exemples d'apprentissage, mettant en relation des stratégies de décision avec des performances.

En effet, nous avons besoin de générer des exemples qui reflètent le fonctionnement réel des systèmes de production pendant une période définie de production, sans pour autant dépenser le temps réel de son fonctionnement.

La simulation permet l'étude des systèmes complexes (interactions complexes entre entités, événements aléatoires, etc) où le régime transitoire et son effet sur les performances sont pris en compte. D'une manière générale, la simulation consiste à étudier et à analyser l'évolution de l'état d'un système à travers la variable temps. Le temps peut être vu comme continu ou discret, selon que les états du système sont spécifiés de manière dénombrable ou non. Dans ce cas, on parle de simulation discrète ou continue (Law and Kelton 2000). Un survol des travaux de recherche existants montre que la simulation à événements discrets est d'un usage plus large que la simulation continue (Pierreval, 2006, Pierreval et al. 2007). En effet, choisir de construire un modèle discret ou continu dépend avant tout des objectifs spécifiques de l'étude.

De plus les potentiels de la simulation sont vastes. La simulation peut couvrir tous les flux de l'entreprise, puisqu'elle est capable de représenter : les flux physiques pour lesquels, d'ailleurs, elle est le plus souvent utilisée, mais également les flux informationnels et les flux décisionnels associés à ces flux physiques. Cet outil informatique est ainsi capable de reproduire sur ordinateur, le comportement dynamique et stochastique d'une machine, d'un atelier, d'une ligne de production, d'une usine, ... et donc, l'évolution de l'état du système en fonction des informations suivies et des décisions prises.

La simulation permet de rassembler les données utiles à la production autant pour un atelier existant que pour un atelier en projet. Ces données seront ainsi formalisées en vue de leur exploitation dans un modèle. Par exemple, la démarche de faire une simulation peut aider à formaliser les caractéristiques globales d'un atelier, en définissant par exemple : le nombre d'opérateurs, le nombre de machines, le dimensionnement des stocks, les fréquences des pannes et autres temps d'arrêt, la liste des règles de gestion, ...

Plus formellement, un modèle de simulation est une représentation informatique d'un système réel permettant d'obtenir, à partir de variables d'entrée x_i : les variables de décision du système, une estimation des variables de sortie y_i : les indicateurs de performance considérés pour le système étudié et des variables de résultats (par exemple, le détail d'une solution de dimensionnement). (Figure 17).

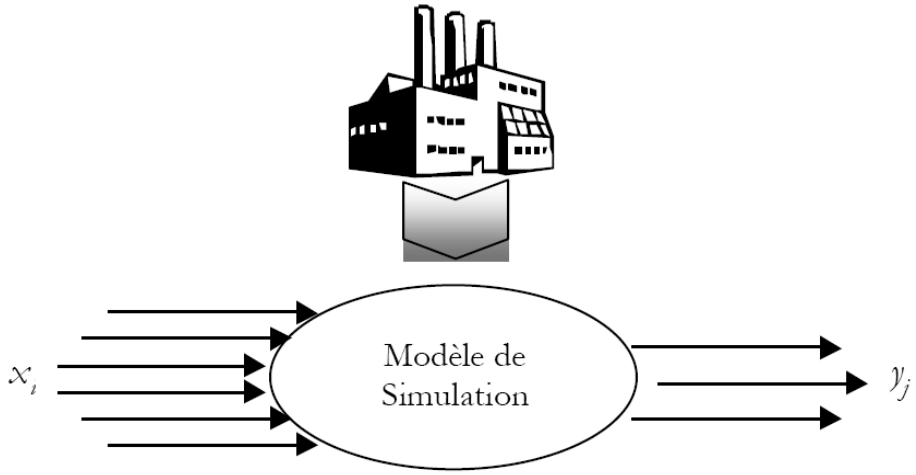


Figure 17. Principe d'un modèle de simulation

Ainsi le modèle de simulation nous permet d'évaluer la performance du système de production étudié selon les valeurs des paramètres d'entrée.

Cependant, la simulation reste un outil d'évaluation de la performance d'un système de production en fonction de ses paramètres d'entrée (Tamani et al. 2006, Habchi et al. 1994). Elle ne permet pas d'analyser et de comprendre directement les relations entre les paramètres et la performance ni d'identifier quelles interactions de ces paramètres influencent la performance globale du système.

Dans la section suivante nous allons citer quelques travaux qui relèvent de l'apprentissage automatique dans les systèmes de production à partir de la simulation.

3.2. Les travaux relatifs à l'apprentissage automatique à partir de la simulation des systèmes de production

Il existe plusieurs travaux sur l'apprentissage automatique dans les systèmes de production à partir de la simulation (Mouelhi et al. 2007a). A partir de résultats de simulation obtenus aléatoirement, l'objectif est d'extraire des connaissances à partir de données simulées sur le comportement des systèmes de production. (Pierreval, 1992) et (Pierreval et Ralambondrainy,

1992) mettent en évidence l'intérêt de disposer de connaissances, présentées notamment sous la forme de règles de production, directement exploitables par un système expert ou par un décideur.

Nombreux sont les travaux qui se basent sur l'apprentissage des réseaux de neurones pour construire des métamodèles qui représentent les systèmes de production. Les métamodèles sont capables de prédire les sorties d'un modèle de simulation d'un système de production à partir de ses entrées.

En premier temps, nous rappelons la définition des métamodèles (Kleijnen, 1979). Soit X_j le vecteur des variables qui influencent la sortie du système réel (j=1, 2, …, s), et soit Y la réponse du système, la relation entre la sortie Y et les variables d'entrée X_j peut être représentée comme suit :

$$Y=f_1(X_1, X_2,\ldots, X_s) \quad \textbf{(6)}$$

Un modèle de simulation est une abstraction du système réel, nous n'y considérons que l'ensemble des variables d'entrée choisies {Xj/j=1, 2, ..., r} où r est significativement inférieure à s qui est inconnu. La réponse Y' de la simulation est alors définie comme une fonction f_2 représentée comme suit :

$$Y'= f_2(X_1, X_2,\ldots, X_r, v) \quad \textbf{(7)}$$

Où v représente un vecteur construit aléatoirement qui représente l'influence des données d'entrée que nous avons exclu.

Un métamodèle est maintenant une nouvelle abstraction, dans laquelle nous choisissons un sous-ensemble de variables d'entrée de la simulation $\{X_j/j=1, 2, ..., m; m\leq r\}$ et décrivent le système comme suit :

$$Y''= f_3(X_1, X_2, \ldots, X_m)+\varepsilon \quad \textbf{(8)}$$

Où ε représente une erreur d'adaptation, qui tend vers zéro. La figure 18 récapitule la méthodologie de construction des métamodèles à partir des réseaux de neurones.

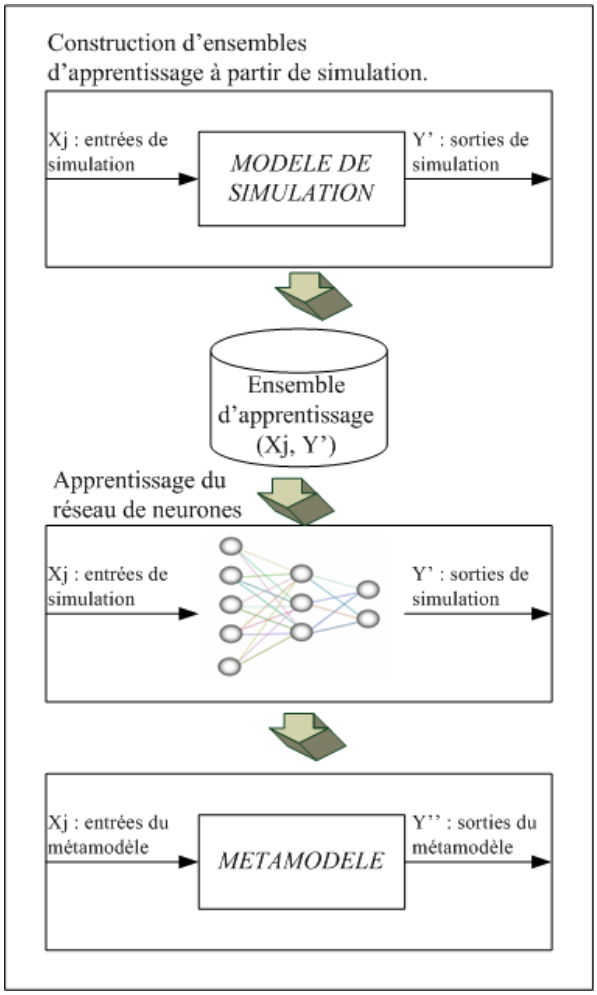


Figure 18. Développement d'un métamodèle à partir d'un réseau de neurones

Parmi les travaux qui mettent en application les métamodèles basés sur les réseaux de neurones dans le domaine des systèmes de production nous citons ; (Pierreval et Huntsinger 1992) qui utilisent le potentiel des réseaux de neurones comme des métamodèles dans deux exemples de modèles : un exemple de modèle discret (un atelier Jobshop) et un exemple de modèle continu. De même, Wang et Yih (1997) utilisent un métamodèle basé sur un réseau de neurones pour mesurer les performances des stratégies de contrôle des automated storage and retrieval systems (AS/RS). Puis, ils choisissent la performance la plus efficace. Plus tard, Kilmer, et al. (1999) ont utilisé un ensemble d'entrées/sorties générées à partir d'un modèle de simulation des stocks d'un atelier pour construire deux réseaux de neurones, un pour évaluer le coût total moyen des stocks et l'autre pour estimer la variance du coût total. Ils ont utilisé ces deux estimations calculées par les réseaux de neurones pour calculer des intervalles de confiance du coût de stockage.

Plus tard, Fonseca et Navaresse (2002) ont proposé un metamodèle basé sur un réseau de neurones pour remplacer un modèle de simulation d'un atelier jobshop. Leur metamodèle était capable d'évaluer les délais (lead times) des commandes traitées simultanément dans un atelier jobshop composé de quatre machines.

Les métamodèles basés sur les réseaux de neurones sont aussi utilisés dans des domaines très spécifiques ; par exemple, Kwak et al. (2005) ont proposé une méthode qui utilise un metamodèle basé sur un réseau de neurones pour résoudre le problème d'un processus de moules d'injection à plusieurs variables. Les techniques d'injection à haute précision sont exigées pour la fabrication des lentilles optiques. Dans cette étude, le réseau de neurones prédit la réduction d'épaisseur et le taux d'altération volumétrique des conditions d'injection.

Il existe plusieurs travaux qui utilisent les métamodèles basés sur les réseaux de neurones couplés avec d'autres approches. Nous citons par exemple Chan et Spedding (2001) qui ont développé un métamodèle basé sur un réseau de neurones d'un système d'assemblage de produits optroniques et ils ont utilisé ce metamodèle couplé avec une approche de surface de réponse pour prédire la performance du système. Aussi, Wang et al. (2005) qui ont développé un modèle hybride de découverte de connaissance, en combinant un arbre de décision et un métamodèle basé sur les réseaux de neurones pour déterminer la bonne règle de production en utilisant des données de production bruitées et prédire sa performance.

Les métamodèles basés sur les réseaux de neurones sont aussi utilisés dans les processus d'optimisation pour évaluer (plus rapidement qu'un modèle de simulation) des solutions proposées par les méthodes d'optimisation. Hurrion (1997) a utilisé un métamodèle basé sur un réseau de neurones dans un programme de recherche combinatoire, avec une étude comparative pour trouver le nombre optimum de kanbans dans un système de production. Wang (2005) a développé une stratégie hybride qui combine l'algorithme génétique et les réseaux de neurones pour résoudre un problème d'optimisation via simulation. La performance des réseaux de neurones étant des bons approximateurs et l'efficacité potentielle des algorithmes génétiques sont combinées pour trouver des conceptions optimales des systèmes.

Cependant, peu d'auteurs ont utilisé les réseaux de neurones pour le pilotage d'atelier en le considérant comme « décideur » dans un système d'aide à la décision (Figure 19) et non pas comme un métamodèle, c'est-à-dire, ils ont considéré le réseau de neurones comme étant un

module dans le modèle de simulation pour l'aide à la décision soit en proposant les décisions possibles aux décideurs pour aider au mieux au pilotage du système, soit en choisissant directement la décision qui optimise la performance visée dans le système de production.

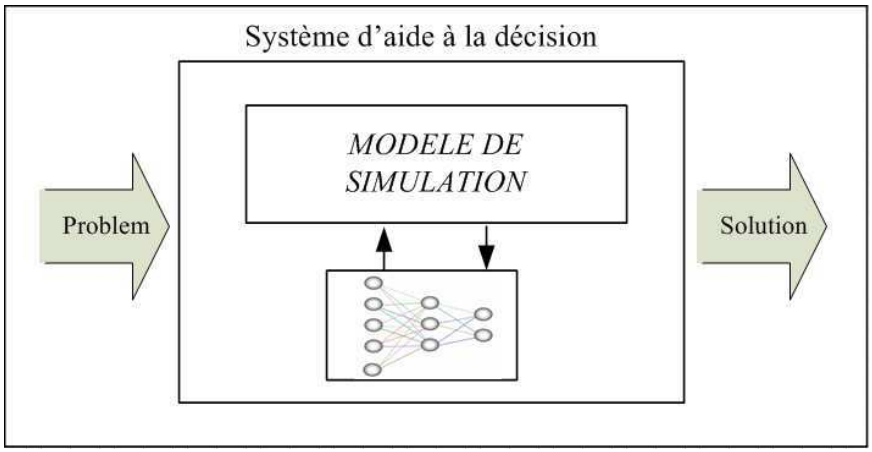


Figure 19. Intégration du réseau de neurones dans le modèle de simulation pour l'aide à la décision.

Parmi les travaux qui intègrent les réseaux de neurones dans les modèles de simulation nous citons par exemple (Pierreval 1992) qui a développé un modèle de simulation pour évaluer les performances des règles de priorité à partir des différents états de l'atelier. Ensuite, il a crée à partir de ces données simulées un ensemble d'apprentissage inverse qui détermine pour chaque état du système la combinaison de règles de priorité adéquate. Enfin, et en se basant sur cet ensemble d'apprentissage, il entraine le réseau de neurones pour prédire la meilleure combinaison de règles qui sera prise en compte dans le modèle de simulation du système de production. En 2006, (Yildirim et al. 2006), ont aussi développé deux réseaux de neurones en se basant sur le même principe, l'un prédit le nombre de machines à utiliser dans chaque centre de travail de leur atelier et l'autre prédit la date de livraison de chaque commande, et cela à partir de la performance donnée par les managers. Enfin, et pour s'assurer de la performance des choix des deux réseaux de neurones ils relancent la simulation et vérifient la performance trouvée avec la performance attendue.

D'autres auteurs se sont intéressés au problème de choix des règles de priorité dans les systèmes de production en se basant sur des méthodes d'apprentissage. Il y a ceux qui choisissent les règles de priorité statiquement, c'est-à-dire, ils testent toutes les règles de priorité sur toute la période de simulation ensuite ils choisissent la règle la plus performante par rapport à la performance choisie

(figure 20). Puis, il y a ceux qui choisissent les règles de priorité périodiquement ; en évaluant les règles de priorité à appliquer dans l'atelier chaque période de temps Δt bien déterminée (figure 21). Et il y a ceux qui choisissent les règles de priorité dans l'atelier dynamiquement à chaque instant de prise de décision soit en fixant un seuil de contrôle qui une fois atteint une nouvelle règle de priorité est choisie, soit un événement déclencheur se passe tel que la libération d'une machine ou l'entrée d'une nouvelle commande, etc (figure 22).

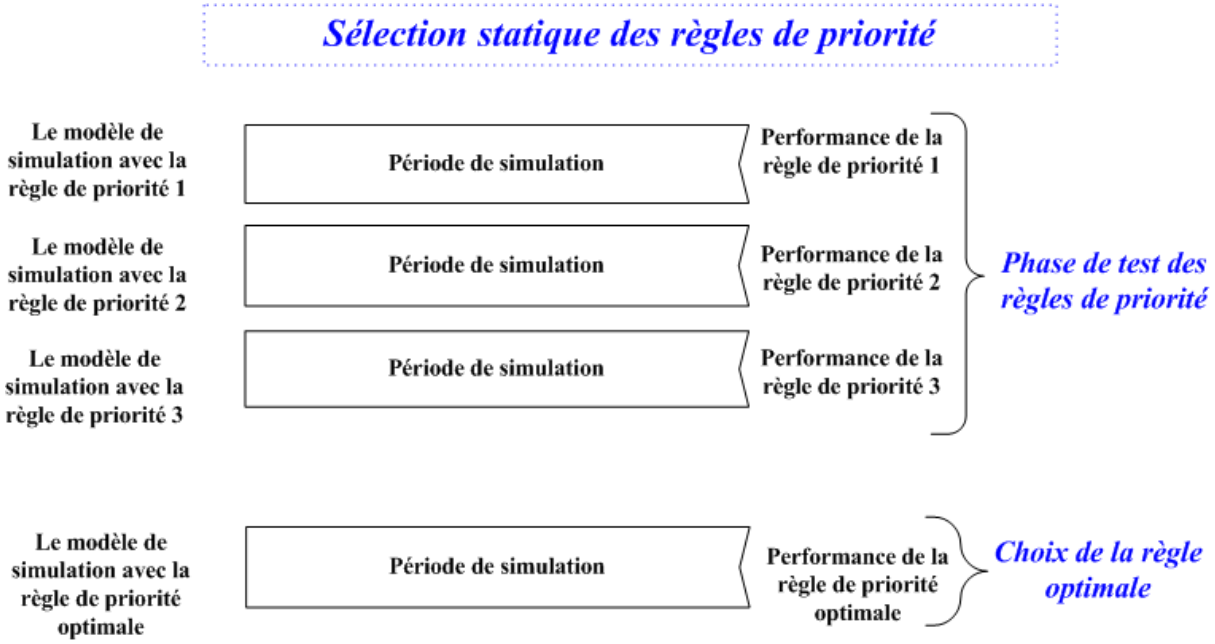


Figure 20. Choix des règles de priorité statiquement.

Parmi les travaux qui choisissent statiquement les règles de priorité nous citons, Li et al., (2007) qui ont utilisé une approche basée sur les réseaux de neurones pour résoudre des problèmes de choix de règles de priorité dans des ateliers flexibles (FMS) à partir d'un nombre de données réelles réduit, en effet, il proposent une technique qui l'ont appelé « Mega-Trend-Diffusion » pour évaluer un petit jeu de données et produire des échantillons artificiels pour former le réseau de neurones en se basant sur la simulation.

Par ailleurs, depuis les années 90 plusieurs travaux sont venus s'inscrire dans le cadre de l'application de l'apprentissage automatique par les méthodes statistiques dans les prises de décisions au sein des systèmes de production. Nous pouvons citer par exemple les travaux de Pierreval (Pierreval 1988) qui utilisent l'analyse de correspondances multiples pour fournir des règles de production à partir de résultats de simulation, ces règles seront éventuellement insérées

dans une base de connaissances (1988), et pour étudier les différentes stratégies d'ordonnancements dynamiques à partir de résultats de simulation (Pierreval 1994).

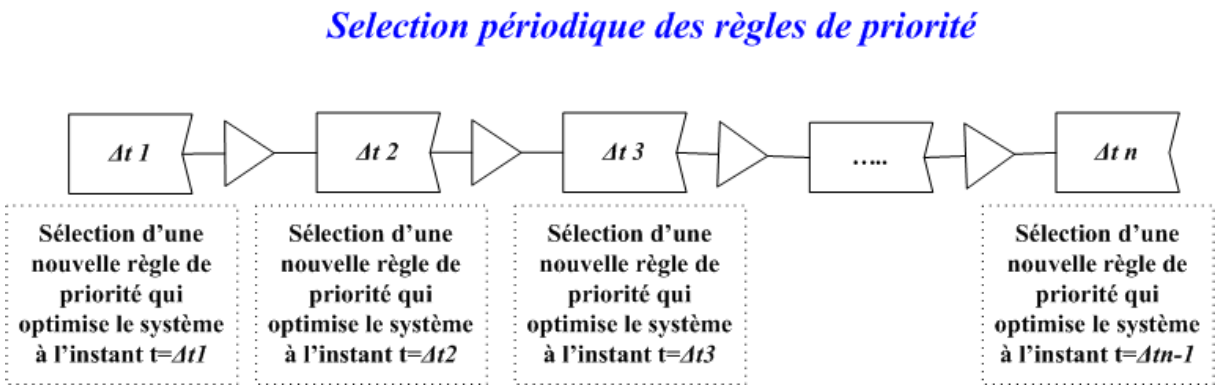


Figure 21. Choix des règles de priorité périodiquement

Parmi les travaux qui choisissent les règles de priorité périodiquement nous citons : (Sun et al. 2004) qui ont présenté une méthode d'ordonnancement basée sur un apprentissage automatique itératif, dans leur système ; les auteurs sélectionnent des règles de priorité dans les files d'attente, ces règles sont incorporées dans le modèle avec des paramètres réglables et, selon la performance globale obtenue de la dernière période de simulation, ces paramètres sont mis à jour afin d'optimiser la valeur de la fonction d'objectif pendant la période suivante.

Shiue et al. (Shiue et al. 2006) ont utilisé une méthode hybride qui lie un algorithme génétique à un réseau de neurones appelée l'approche d'ordonnancement adaptative multi-périodes, qui est capable de découvrir l'état courant du système et selon cet état elle choisie la règle de priorité adéquate pour la période d'ordonnancement qui suit.

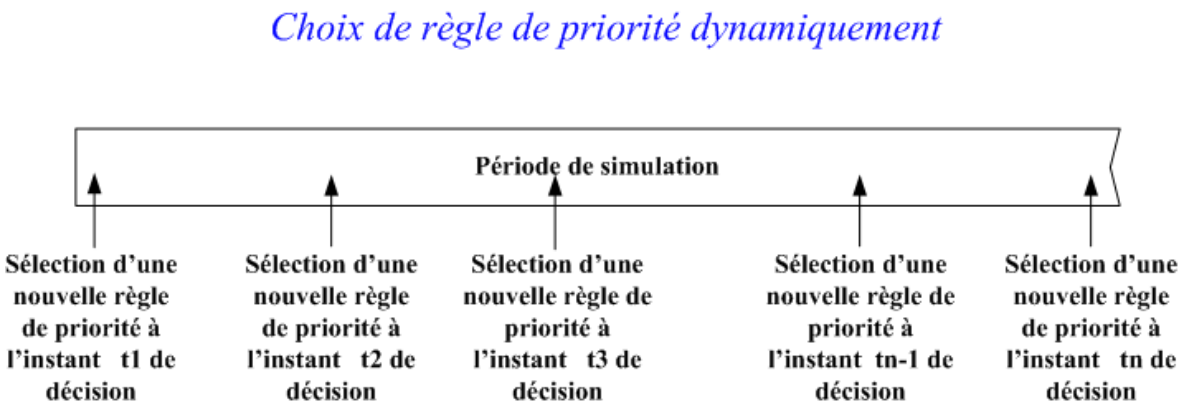


Figure 22. Choix des règles de priorité dynamiquement

Parmi les travaux qui choisissent dynamiquement les règles de priorité nous citons (Metan et al. 2009) qui, pour considérer plus efficacement les changements d'état du système pendant une période donnée, ont ajouté une carte de contrôle pour mieux surveiller ces changements. Ils proposent un système d'ordonnancement basé sur l'apprentissage automatique pour choisir les règles de priorité dans les files d'attente de leur système à chaque fois la carte de contrôle détecte le seuil fixé, l'objectif est d'apprendre à partir des caractéristiques de l'atelier et construire un arbre de décision et ensuite sélectionner les règles de priorité en ligne à partir de l'arbre construit pendant une période d'ordonnancement. Le système utilise une carte de contrôle pour contrôler la performance de l'arbre de décision, qui est automatiquement mis à jour chaque fois que ceci est nécessaire. La performance de cette méthode dépend du paramétrage du système de contrôle et comment l'arbre de décision est construit.

El-Bouri et Shah (El-Bouri et al. 2006) ont utilisé deux réseaux de neurones pour choisir une combinaison de règles de priorité dans un atelier Jobshop ; l'un pour minimiser le makespan et l'autre pour minimiser le flowtime moyen. Les règles sont changées dynamiquement à chaque fois que le seuil de congestion fixé par les auteurs est atteint. Les deux réseaux de neurones sont entrainés à partir d'ensembles d'apprentissage obtenus du modèle de simulation de l'atelier.

Récemment, Lee (Lee 2008) a proposé une approche basée sur les règles floues pour résoudre un problème d'ordonnancement adaptatif ; il choisit et applique dynamiquement la règle de priorité la plus adéquate selon l'état de l'environnement d'ordonnancement. Ces règles floues sont construites à partir d'ensembles d'apprentissage issus de la simulation.

D'autres travaux ont été menés en appliquant des méthodes basées sur l'apprentissage automatique dans le domaine des systèmes de production tel qu'Héritier (Héritier 1991) qui utilise de l'analyse de données par simulation pour effectuer de l'aide à la conception de systèmes de production. Et nous pouvons citer aussi d'autres approches qui sont basées sur les principes de classification comme l'approche présentée par Bonneau et Proth (Bonneau et al. 1985) pour choisir les règles de gestion d'un système de production.

D'autres travaux sont basés sur différentes méthodes d'apprentissage automatique tel que : (Lereno et al. 2001) qui ont construit des arbres de décision à partir des résultats de simulation choisis aléatoirement afin d'aider à la décision dans un logiciel d'ordonnancement. Creighton et Nahavandi (Creighton et al. 2002) ont optimisé les performances d'une ligne de production en

série par apprentissage par renforcement. Cette stratégie leur permet d'optimiser le système plus rapidement et plus efficacement.

De même dans ses travaux Huyet (Huyet 2006) a proposé une méthodologie basée sur la synergie d'une approche d'optimisation évolutionniste et une approche d'apprentissage par les graphes d'induction. Avec cette méthodologie, il est possible d'extraire et d'utiliser des connaissances à partir des meilleures solutions proposées par le système de production étudié et analyser son comportement pour en extraire les bonnes règles du graphe d'induction. L'auteur a appliqué cette méthodologie sur un atelier jobshop à cinq machines.

Récemment, (Aissani et al. 2008) ont proposé un système de commande industriel où les décisions prises par le système sont le résultat d'un travail de groupe d'agents, ces agents assurent une amélioration continue de la performance globale du système, grâce à l'apprentissage par renforcement. Cette technique d'apprentissage permet aux agents d'apprendre le meilleur comportement dans leurs divers rôles (l'auto-organisation, le planning des tâches, etc) sans altérer la qualité des décisions prises en temps réel.

Bilan

Nous avons présenté dans ce qui précède, plusieurs travaux qui se basent sur la simulation pour générer des connaissances. Plusieurs auteurs ont montré que la simulation était capable de fournir des exemples, mettant en relation des stratégies de décision avec des performances, de telle sorte que des outils d'apprentissage automatique puissent en extraire des connaissances utiles et construire les ensembles d'apprentissage.

Malheureusement, l'usage d'ensembles d'apprentissage nécessite que l'on soit capable de générer par simulation des exemples distinguant les bonnes décisions de mauvaises décisions, au regard d'un objectif de performance, puisque les méthodes d'apprentissage automatique sont très sensibles à la qualité de l'ensemble de données d'apprentissage, comme les réseaux de neurones, les règles floues, les arbres de décision, etc. (Osei-Bryson 2004). Ainsi, les résultats d'apprentissage obtenus restent très dépendants des données sur lesquelles les connaissances sont extraites.

Ce problème s'accentue encore plus lorsqu'il s'agit de prendre des décisions en temps réel ; en effet, lorsque l'on considère des décisions de pilotage en temps réel, l'impact sur la performance des décisions prises est difficile à mesurer et à apprécier immédiatement. C'est pour cette raison que la plupart des travaux existants considèrent les performances obtenues sur une période d'étude suffisamment longue ou sur plusieurs périodes relativement courtes dans l'optique d'améliorer la prise en compte des changements d'états dans l'atelier au cours du temps. Or, même en considérant de telles périodes il est difficile de capter tous les changements dans l'atelier au cours de chaque période considérée et qui peuvent ainsi amener à la prise de nouvelles décisions (ex. ajouter une nouvelle machine, réaffecter des opérateurs, changer les règles de priorité dans les files d'attente, etc.).

Cependant notre objectif est de générer des connaissances relatives au fonctionnement de l'atelier en temps réel et en considérant chaque changement d'état dans l'atelier à chaque instant de prise de décision et non sur des périodes données.

Autrement dit, nous cherchons à générer des connaissances par simulation sans utiliser des ensembles d'apprentissage préalablement déterminés. Nous parlons alors d'acquisition autonome de connaissance (Mouelhi et al. 2007b). Des travaux possédant quelques liens avec l'approche que nous proposons sont présentés dans (Geiger *et al.* 2006) pour générer une nouvelle règle de priorité de passage de pièce. Les principes exposés ici diffèrent fortement de l'approche par programmation génétique proposée par ces auteurs. Par ailleurs, nos propositions ont vocation à s'appliquer à des problèmes beaucoup plus large de pilotage d'atelier.

4. Conclusion

Dans la synthèse de la littérature que nous avons présenté dans cet état de l'art nous avons mis l'accent sur l'utilisation de la simulation pour l'apprentissage des méthodes d'apprentissage automatique (ex. les graphes d'induction, les réseaux de neurones, les méthodes statistiques, etc.) dans les systèmes de production. La plupart des travaux utilisent des ensembles d'apprentissage issus des modèles de simulation des systèmes de production étudiés, mais ceci présente des inconvénients si on considère les changements de l'état du système au cours du temps, la prise en compte de ces changements est indispensable pour les changements des stratégies de production au cours du temps. En effet, la prise en compte de tels changements en temps réel dans les systèmes de production est nécessaire dans des systèmes qui sont confrontés à des imprévus qui

peuvent survenir au cours du processus de production (commandes urgentes, manque de matière première, pannes de machine, absentéisme, etc) et qui peuvent nuire aux objectifs de la production.

Dans ce sens, plusieurs travaux ont essayé de prendre en compte ce type des changements. Les chercheurs ont proposé des solutions de décomposition des temps de production pour les simuler sur des périodes courtes, d'autres ont proposé des seuils de contrôle à partir duquel ils révisent l'état du système et prennent en compte les changements qui ont amené à atteindre ces seuils. Malheureusement, la solution de décomposition ne résout pas totalement l'aspect temps réel puisque au cours même de la courte période simulée l'état du système peut changer et ce dernier ne peut être pris en compte que dans la période qui suit. De même pour les seuils de contrôle, puisqu' il est très difficile de déterminer le bon seuil à partir duquel la prise d'une nouvelle décision de production est nécessaire.

A notre connaissance il n'existe pas d'approche qui assure un apprentissage prenant en compte les changements d'état dans les systèmes de production en temps réel.

Dans la suite de ce manuscrit nous allons donner notre formulation du problème et nous allons présenter notre approche

Chapitre III

APPRENTISSAGE AUTONOME DES RESEAUX DE NEURONES BASE SUR OPTIMISATION VIA SIMULATION

1. Introduction

Comme nous l'avons expliqué dans les chapitres précédents, le pilotage des systèmes de production est un problème qui nécessite une prise de décision très complexe d'une part, et une poursuite de plus en plus fine d'objectifs associés à des niveaux de performance élevés d'autre part. En effet, très dépendant de l'expertise du décideur, le pilotage en temps réel est devenu l'enjeu de plusieurs travaux de recherche qui essayent d'acquérir cette expertise pour en déduire les bonnes décisions des mauvaises et satisfaire ainsi la performance visée.

Malheureusement, l'acquisition de l'expertise du décideur nécessite l'usage d'ensembles d'apprentissage comme nous l'avons présenté dans le chapitre II. Et nous avons montré qu'il est nécessaire que l'on soit capable de générer par simulation les exemples distinguant les bonnes décisions des mauvaises décisions, au regard d'un objectif de performance. Cependant, décider si la décision prise est-elle bonne ou mauvaise au cours d'un pilotage en temps réel est très difficile si cela n'est pas impossible. En effet, est-il possible pour un décideur dans un système de production en temps réel de dire si le fait d'affecter l'opérateur X à une machine Y à l'instant *t* améliore la production future ? Ou bien quelle commande urgente doit être passée à cet instant *t* (temps réel) parmi un ensemble de commandes urgentes dans une file d'attente d'une machine qui minimisera le retard moyen dans l'atelier à la fin de la production ?

2. Formulation du problème : description et objectifs

Le problème abordé dans notre étude consiste à déterminer comment obtenir les connaissances nécessaires pour le pilotage en temps réel. Ces connaissances doivent notamment être capables de déterminer les bonnes actions de pilotage pour aboutir aux objectifs de la production (e. g. minimiser les en-cours, minimiser le retard moyen, etc.) à partir de l'état courant du système de production (e. g. le nombre de machines libres, le nombre de pièces dans chaque file d'attentes, etc). Nous proposons dans ce qui suit une formulation de ce problème.

Considérons d'une part, un ensemble d'actions de pilotage en temps réel *A*, tels que choisir une règle de priorité, choisir un opérateur, ou des outils, etc. Où

$$A = \{A_1, A_2, A_3, \ldots, A_k\} \quad \textbf{(9)}$$

Soit Å(t) la fonction permettant de déterminer une action A_i selon l'état courant du système à chaque instant t de prise de décision.

$$Å(t) : t \rightarrow A_i \text{, où } i \in \{1, 2, \ldots, k\} \quad \textbf{(10)}$$

D'autre part, soit le vecteur *X* qui décrit le système à l'un instant *t* de prise de décision dont les composantes sont par exemple le nombre de machines libres, les opérateurs disponibles pour une machine donnée, etc.

Nous considérons aussi la fonction objectif du système de production *f* que nous cherchons à optimiser (minimiser ou maximiser). Soit ∑ la séquence des actions prises lors des décisions pendant toute la période d'étude. La fonction *f* évalue la séquence des actions ∑ des actions de pilotage prises tout au long du processus de production du système par la fonction Å(t) en se basant à chaque fois sur l'état courant du système *X*. On souhaite trouver la meilleure séquence ∑ tel que

$$Min \text{ ou } Max \quad [f(\sum)] \quad \textbf{(11)}$$

Nous abordons ce problème en recherchant une fonction non linéaire G approximant Å qui nous permettra à chaque instant *t* donné de prise de décision de déterminer l'action de pilotage à

choisir parmi l'ensemble de pilotage A, à partir du vecteur *X* caractérisant l'état du système dans le but d'optimiser la fonction objectif *f*. Toutes les étapes pour déterminer cette fonction sont données au chapitre suivant.

Les variables à prendre en compte dans la prise de décision

Le vecteur *X* inclus des paramètres et des variables du système qui reflètent son état courant. En effet, dans un tel contexte, chaque décideur a besoin de ces informations (transmises par le vecteur *X*) pour prendre les bonnes décisions qui optimisent les performances de l'atelier. Alors nous estimons qu'il est nécessaire de bien étudier le choix de ces variables. Elles sont un facteur clef dans l'élaboration de l'approche que nous proposons par la suite, puisqu' elles influencent le pilotage du système.

En effet, il existe plusieurs variables qui peuvent être transmises au décideur à chaque instant *t* de prise de décision pour pouvoir choisir une des actions de pilotage. Ces informations peuvent être utiles comme inutiles pour la prise de décision. La question qui se pose alors est la suivante : « que doit-on inclure, et que doit-on exclure? ». En fait, si des éléments importants sont exclus, il y a de forte chance pour que le décideur n'ait pas assez d'informations pour prendre les bonnes décisions. Inversement, trop de détails pris en compte dans le modèle créent un bruitage inutile qui biaise la décision du décideur. Le niveau de détail à transmettre à ce dernier dépend des objectifs fixés, du détail des données acquises,...

Nous définissons deux vecteurs *S(t)* et *D* qui composent le vecteur *X* représentant l'état du système à l'instant *t*.

$$X = (S(t), D) \tag{13}$$

Le vecteur *S(t)* représente les variables de décisions qui changent dynamiquement tout au long de la période de l'étude de l'atelier, nous les appelons les variables d'état, où *t* est l'instant auquel le décideur prend sa décision. Ces variables caractérisent l'état courant de l'atelier au temps réel, parmi ces variables nous citons par exemples le nombre de pièces dans chaque file d'attente, le nombre de machines libres à l'instant *t*, les opérateurs disponibles pour une machine donnée à l'instant *t*, etc.

Et le vecteur $D = (D_1, D_2, \ldots, D_l)$ représente les paramètres de l'atelier qui doivent être pris en considération pour prendre les décisions, mais qui ne changent pas pendant la période de l'étude de l'atelier. Par exemple : le nombre de machines disponibles dans un workcenter donné, le taux de charge de l'atelier, le nombre d'opérateurs opérants sur chaque machine, etc.

Avec, $X = (S(t), D)$

$$S(t) = (S(t1), S(t2), \ldots, S(tk)), \quad t = t1, t2, \ldots, tk \qquad \textbf{(14)}$$

$$D = (D_1, D_2, \ldots, D_l), \qquad \textbf{(15)}$$

Ces deux vecteurs $S(t)$ et D sont construits en recensant les paramètres et les variables qui reflètent au mieux l'état du système pour résoudre un problème bien précis.

Les paramètres de l'atelier qui sont représentés par le vecteur D sont comme nous l'avons déjà expliqué invariables et ne changent pas au cours du temps, alors leur détermination ne nécessite pas une longue étude des caractéristiques de l'atelier ou des problèmes que nous voulons résoudre dans un atelier donné. Cependant, les variables d'état $S(t)$ qui changent au cours du temps et qui reflètent l'état courant du système sont difficiles à déterminer et leur choix est étroitement lié aux types de problèmes de pilotage que nous considérons dans l'atelier à étudier. Il existe des travaux dans la littérature qui ont été menés pour désigner quelques variables pour décrire l'atelier à un instant donné de prise de décision et qui reflètent au mieux son état pour un problème donné ou bien pour optimiser une performance considérée dans un atelier.

Nous citons les travaux de Shiue et Guh, (Shiue et al. 2006) qui présentent un ensemble d'attributs et de variables qu'ils prennent en compte pour décrire l'état de l'atelier à un instant donné. Le tableau 2 représente toutes les variables d'état que Shiue et Guh ont déterminé pour décrire leur système.

Tableau 2. Variables d'état d'un système de production (Shiue et Guh, 2006)

System attribute	Description	Mathematical definition
Njob	Number of the jobs in the system	$\lvert \boldsymbol{Jw} \rvert$
MeUM	The mean utilization of machines	$\frac{\sum_{k} WP_k}{K}$
SdUM	The standard deviation of machine utilization	$\sqrt{\sum_{k} \frac{[(WP_k - \text{MeUM}]^2}{K-1}}$
MeUL	The mean utilization of load/unload stations	$\frac{\sum_{u} WP_l}{U}$
MeUB	The mean utilization of pallet buffers	$\frac{\sum_{b} WP_b}{B}$
MeUA	The mean utilization of AGVs	$\frac{\sum_{a} WP_a}{A}$
MiOT	The minimum imminent operation time of candidate jobs within the system	$\underset{i \in Jw}{Min}\{pt_{ij}\}$
MaOT	The maximum imminent operation time of candidate jobs within the system	$\underset{i \in Jw}{Max}\{pt_{ij}\}$
MeOT	The mean imminent operation time of candidate jobs within the system	$\frac{\sum_{i \in Jw} pt_{ij}}{\lvert Jw \rvert}$
SdOT	The standard deviation of the imminent operation time of candidate jobs within the system	$\sqrt{\sum_{i \in Jw} \frac{[pt_{ij} - \text{MeOT}]^2}{\lvert Jw \rvert - 1}}$
MiPT	The minimum total processing time of candidate jobs within the system	$\underset{i \in Jw}{Min}\left\{\sum_{j} pt_{ij}\right\}$
MaPT	The maximum total processing time of candidate jobs within the system	$\underset{i \in Jw}{Max}\left\{\sum_{j} pt_{ij}\right\}$
MePT	The mean total processing time of candidate jobs within the system	$\frac{\sum_{i \in Jw} \sum_{j} pt_{ij}}{\lvert Jw \rvert}$
SdPT	The standard deviation of the total processing time of candidate jobs within the system	$\sqrt{\sum_{i \in Jw} \frac{\left[\left(\sum_{j} pt_{ij}\right) - \text{MePT}\right]^2}{\lvert Jw \rvert - 1}}$
MiRT	The minimum remaining processing time of candidate jobs within the system	$\underset{i \in Jw}{Min}\left\{\sum_{j \in R_i} pt_{ij}\right\}$
MaRT	The maximum remaining processing time of candidate jobs within the system	$\underset{i \in Jw}{Max}\left\{\sum_{j \in R_i} pt_{ij}\right\}$
MeRT	The mean remaining processing time of candidate jobs within the system	$\frac{\sum_{i \in Jw} \sum_{j \in R_i} pt_{ij}}{\lvert Jw \rvert}$
SdRT	The standard deviation of the remaining processing time of candidate jobs within the system	$\sqrt{\sum_{i \in Jw} \frac{\left[\left(\sum_{j \in R_i} pt_{ij}\right) - \text{MeRT}\right]^2}{\lvert Jw \rvert - 1}}$
MiST	The minimum slack time of candidate jobs within the system	$\underset{i \in Jw}{Min}\{ST_i\}$
MeST	The mean slack time of candidate jobs within the system	$\frac{\sum_{i \in Jw} \{ST_i\}}{\lvert Jw \rvert}$
SdST	The standard deviation of the slack time of candidate jobs within the system	$\sqrt{\sum_{i \in Jw} \frac{[(ST_i) - \text{MeST}]^2}{\lvert Jw - 1 \rvert}}$
MaTA	The maximum tardiness of candidate jobs within the system	$\underset{i \in Jw}{Max}\{TT_i\}$
MeTA	The mean tardiness of candidate jobs within the system	$\frac{\sum_{i \in Jw} \{TT_i\}}{\lvert Jw \rvert}$
SdTA	The standard deviation of the tardiness of candidate jobs within the system	$\sqrt{\sum_{i \in Jw} \frac{[TT_i - \text{MeTA}]^2}{\lvert Jw \rvert - 1}}$
MaWL	The maximum workload in front of any machine/station within the system	$\underset{k \cup l}{Max}\left\{\sum_{i \in Jw} \sum_{j \in R_i} pt_{ij}^k \cup \sum_{i \in Jw} \sum_{j \in R_i} pt_{ij}^l\right\}$
ToWL	The total workload in front of any machine/station within the system	$\sum_{i \in Jw} \sum_{j \in R_i} pt_{ij}$
MeSO	The mean sojourn time of candidate jobs within the system	$\sum_{i \in Jw} \frac{t - AT_i}{\lvert Jw \rvert}$
SdSO	The standard deviation of the sojourn time of candidate jobs within the system	$\sqrt{\sum_{i \in Jw} \frac{[(t - AT_i) - \text{MeSO}]^2}{\lvert Jw - 1 \rvert}}$
MeTD	The mean time now until due date of candidate jobs within the system	$\sum_{i \in Jw} \frac{DT_i - t}{\lvert Jw \rvert}$
SdTD	The standard deviation of the time now until due date of candidate jobs within the system	$\sqrt{\sum_{i \in Jw} \frac{[(DT_i - t) - \text{MeTD}]^2}{\lvert Jw - 1 \rvert}}$

Notre objectif est donc d'approximer la fonction non linéaire *G* qui représente la relation entre le vecteur *X* de l'état courant du système de production aux instants de prise de décision *t* aux actions de pilotage qui leurs correspondent pour optimiser la fonction objectif *f*.

Dans ce qui suit, nous présentons dans un premier temps les propriétés qui nous ont amené à choisir le réseau de neurones pour approximer la fonction *G*. Dans un second temps, nous expliquons le principe de prise de décisions en temps réel par le réseau de neurones. Et enfin, nous présentons une méthode d'apprentissage du réseau de neurones basée sur l'optimisation via simulation.

3. L'approche proposée

Dans la mesure où le pilotage en temps réel s'effectue sur la base d'un état du système changeant dynamiquement au cours du temps, il est donc très difficile de déterminer la fonction *G* qui lie les états courants du système de production aux bonnes actions de pilotage qui leurs correspondent, aux instants de prise de décision pour optimiser la fonction objectif.

Nous proposons dans cette étude une approche qui permettra d'approximer la fonction non linéaire *G* qui représente la logique du pilotage que nous cherchons à déterminer. Notre approche est composée de deux phases principales, une phase de construction de la logique du pilotage et d'approximation de la fonction *G*, et une phase de l'intégration de cette logique dans le système à piloté.

Pour approximer la fonction *G* et construire la logique du pilotage nous avons utilisé un réseau de neurones. En fait, les réseaux de neurones possèdent des propriétés, que nous détaillerons par la suite, qui leur permettent d'approximer n'importe quelle fonction non linéaire. Ce réseau de neurones apprendra à prendre les bonnes décisions du pilotage en temps réel en se basant sur un apprentissage qui vise à extraire de façon autonome des connaissances d'un modèle de simulation, sans utiliser ni ensembles d'apprentissage ni expertise préalable. Dans la démarche présentée, l'apprentissage s'effectue par optimisation via simulation (Wetter *et al.* 2003) des caractéristiques d'un réseau de neurones par rapport à la fonction objectif *f*. Après optimisation, le réseau possède la capacité de décider quelle action de pilotage entreprendre, en fonction du vecteur *X* de l'état de l'atelier, pour satisfaire à l'objectif *f* visé pour le pilotage. Une fois que le

réseau de neurones a appris, il peut être alors intégré dans le système de pilotage global du système de production considéré.

Dans ce qui suit, nous présentons dans un premier temps les propriétés qui nous ont amené à choisir le réseau de neurones pour approximer la fonction *G*.

Ensuite, nous expliquons comment nous nous somme basé sur l'optimisation via simulation pour procéder à l'apprentissage autonome du réseau de neurones.

3.1. **Une approche basée sur les réseaux de neurones**

3.1.1. Les réseaux de neurones : approximateurs universels parcimonieux

Les réseaux de neurones, tels que nous les avons définis dans le chapitre précédent, possèdent une propriété remarquable qui est à l'origine de leur intérêt pratique dans des domaines très divers: ce sont *des approximateurs universels parcimonieux* (Dreyfus et al. 2004).

Sans entrer dans les détails mathématiques, la propriété d'approximation peut être énoncée de la manière suivante : *toute fonction bornée suffisamment régulière peut être approchée avec une précision arbitraire, dans un domaine fini de l'espace de ses variables, par un réseau de neurones comportant une couche de neurones cachés en nombre fini, possédant tous la même fonction d'activation, et un neurone de sortie linéaire* (Hornik et al. 1989). Cette propriété n'est pas spécifique aux réseaux de neurones : il existe bien d'autres familles de fonctions paramétrées possédant cette propriété ; c'est le cas notamment des ondelettes, des fonctions radiales, des fonctions splines, par exemple. La spécificité des réseaux de neurones réside dans le caractère *parcimonieux* de l'approximation : à précision égale, les réseaux de neurones nécessitent moins de paramètres ajustables (les poids des connexions) que les approximateurs universels couramment utilisés ; plus précisément, le nombre de poids varie *linéairement* avec le nombre de variables de la fonction à approcher, alors qu'il varie *exponentiellement* pour la plupart des autres approximateurs.

Nous verrons dans ce qui suit que c'est cette remarquable parcimonie qui justifie l'intérêt industriel des réseaux de neurones. En pratique, dès qu'un problème fait intervenir plus de deux variables, les réseaux de neurones sont, en général, préférables aux autres méthodes.

Qualitativement, la propriété de parcimonie peut se comprendre de la manière suivante : lorsque l'approximation est une combinaison *linéaire* de fonctions élémentaires fixées (des monômes par exemple, où des gaussiennes à centres et écarts-types fixes), on ne peut ajuster que les coefficients de la combinaison, en revanche, lorsque l'approximation est une combinaison linéaire de fonctions non linéaires *à paramètres ajustables* (un Perceptron multicouche par exemple), on ajuste à la fois les coefficients de la combinaison *et la forme des fonctions que l'on combine.* Ainsi, dans un Perceptron multicouche, les poids de la première couche déterminent la forme de chacune des sigmoïdes réalisées par les neurones cachés, et les poids de la seconde couche déterminent une combinaison linéaire de ces fonctions. On conçoit facilement que cette souplesse supplémentaire, conférée par le fait que l'on ajuste la forme des fonctions que l'on superpose, permet d'utiliser un plus petit nombre de fonctions élémentaires, donc un plus petit nombre de paramètres ajustables. Nous allons voir dans ce qui suit pourquoi cette propriété de *parcimonie* est précieuse dans les applications industrielles.

Dans la pratique, on n'utilise pas les réseaux de neurones pour réaliser des approximations de fonctions *connues*. Le plus souvent, le problème qui se pose à l'ingénieur est le suivant : il dispose d'un ensemble de mesures de variables d'un processus de nature quelconque (physique, chimique, économique, financier, ...), et du résultat de ce processus ; il suppose qu'il existe une relation déterministe entre ces variables et ce résultat, et il cherche une forme mathématique de cette relation, valable dans le domaine où les mesures ont été effectuées, sachant que (1) les mesures sont en nombre fini, (2) elles sont certainement entachées de bruit, et (3) toutes les variables qui déterminent le résultat du processus ne sont pas forcément mesurées. En d'autres termes, l'ingénieur cherche un *modèle* du processus qu'il étudie, à partir des mesures dont il dispose, et d'elles seules : on dit qu'il effectue une modélisation "boîte noire". Dans le jargon des réseaux de neurones, les données à partir desquelles on cherche à construire le modèle s'appellent des *exemples.*

En quoi la propriété d'approximation parcimonieuse peut-elle être utile pour résoudre ce genre de problèmes ? Ce que l'ingénieur cherche à obtenir à l'aide de son modèle, c'est la "vraie" fonction qui relie la grandeur y_p que l'on veut modéliser aux variables $\{x\}$ qui la déterminent, c'est à- dire la fonction que l'on obtiendrait en faisant une infinité de mesures de y_p pour chaque valeur possible de $\{x\}$: en termes de statistiques, l'ingénieur cherche la *fonction de régression* de la grandeur à modéliser. Cette fonction est inconnue, mais on peut en chercher une approximation à partir des mesures disponibles : les réseaux de neurones sont donc de bons candidats pour cela, si

la fonction de régression cherchée est non linéaire. Cette approximation est obtenue en estimant les paramètres d'un réseau de neurones au cours de la phase *d'apprentissage*.

C'est ici que la propriété d'approximation parcimonieuse des réseaux de neurones est précieuse : en effet, le nombre de mesures nécessaires pour estimer les paramètres de manière significative est d'autant plus grand que le nombre de paramètres est grand. Ainsi, pour modéliser une grandeur avec une précision donnée à l'aide d'un réseau de neurones, il faut *moins de données* que pour la modéliser, avec une précision comparable, à l'aide d'une régression linéaire multiple ; de manière équivalente, un réseau de neurones permet, avec les mêmes données disponibles, de réaliser une approximation plus précise qu'une régression linéaire multiple (Dreyfus, 2004).

De manière générale, un réseau de neurones *permet donc de faire un meilleur usage des mesures disponibles que les méthodes de régression non linéaires conventionnelles*. Ce gain peut être considérable lorsque le processus à modéliser dépend de plusieurs variables : en effet le nombre de paramètres (donc de mesures) varie *exponentiellement* pour les méthodes conventionnelles de régression non linéaire, alors qu'elle varie *linéairement* pour les réseaux de neurones.

Ainsi, à la lumière de cette propriété fondamentale, la technique des réseaux de neurones apparaît comme une puissante *méthode de régression non linéaire* : ce n'est donc rien d'autre qu'une extension des méthodes de régression linéaire ou multilinéaires (Dreyfus, 2004).

3.1.2. La prise de décision par le réseau de neurones

Dans l'approche que nous proposons dans cette thèse, le réseau de neurones est utilisé en tant que décideur (figure 3.1) dans le système de production, ses décisions se basent sur des caractéristiques importantes de l'atelier (par exemple : nombre de machines disponibles, taux de charge, etc.) ainsi que son état à l'instant t où une décision doit être prise (par exemple le nombre des pièces d'un certain type en attente pour une ressource donnée, disponibilité d'opérateurs, disponibilité des machines, etc.). Le réseau de neurones décide alors d'une action (ou plusieurs actions) à effectuer parmi les actions possibles de l'ensemble d'actions de pilotage $\boldsymbol{A}$ par exemple : affectation d'un opérateur donné à une machine donnée, permutation de machines, etc. Les décisions prises par le réseau de neurones doivent concourir à la satisfaction des objectifs de production de l'atelier, par exemple la minimisation des retards ou des en-cours.

Ces décisions sont basées sur une logique qui est le raisonnement du réseau de neurones, et ce raisonnement dépend des caractéristiques du réseau de neurones : sa topologie (nombre de couches cachées, nombre de neurones dans chaque couche, …), ses poids et ses paramètres (le type des fonctions d'activation qu'ils utilisent, les paramètres de ces fonctions,…), que l'on caractérise par un vecteur numérique noté *W*. Nous considérons alors la fonction qui représente la logique du réseau de neurones comme suit :

$$W = G\,(\,E\,(f\,(\Sigma)\,)\,) \qquad \textbf{(16)}$$

Cependant, comment pouvons-nous construire cette logique qui lie l'état courant du système de production aux instants de prise de décision aux bonnes actions de pilotage qui leurs correspondent et que le réseau de neurones prendra pour optimiser la fonction objectif *f* ? En d'autres termes, comment pouvons-nous déterminer la fonction *G* qui permettra au réseau de neurones de prendre les bonnes décisions au regard de la performance visée ?

Notre objectif est d'optimiser cette fonction *G* tout au long de la phase d'apprentissage du réseau de neurones, c'est-à-dire optimiser ses caractéristiques pendant cette première phase de l'approche.

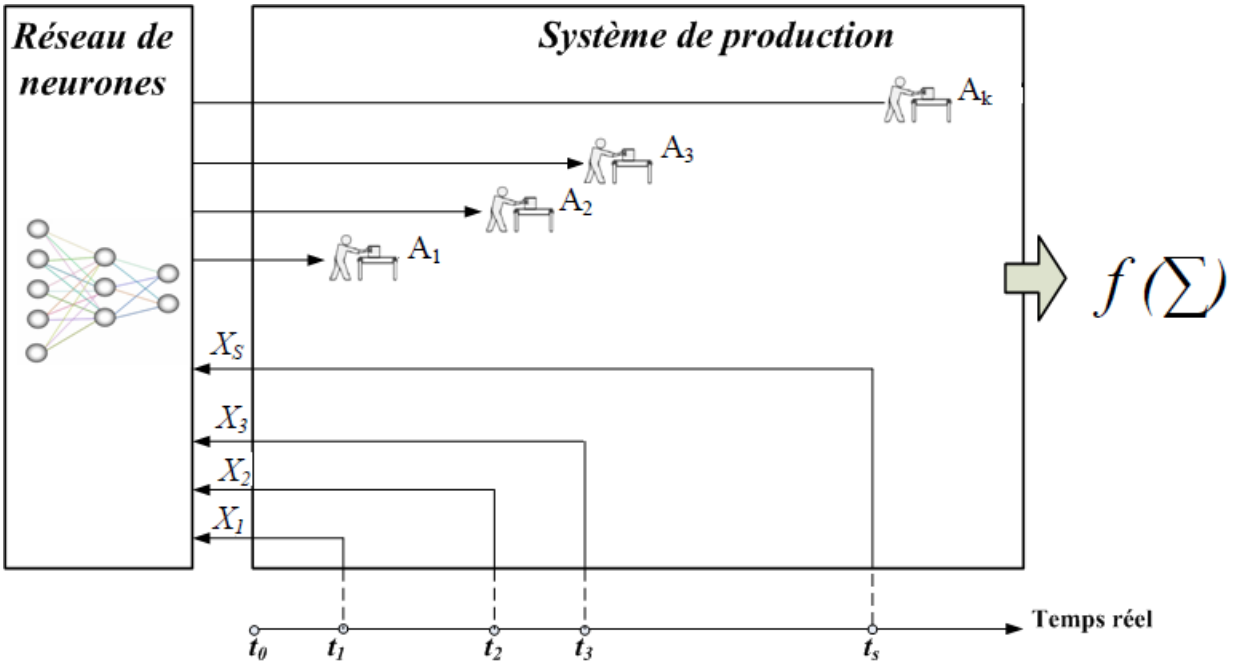


Figure 23. La prise de décision par le réseau de neurones dans un système de production

3.2. Auto-apprentissage des réseaux de neurones par optimisation via simulation

L'apprentissage classique des réseaux de neurones multicouches de type Perceptron est basé généralement sur des ensembles d'apprentissage pour en extraire les connaissances. Cependant, dans le contexte de notre étude nous ne pouvons pas obtenir ces ensembles d'apprentissage, comme nous l'avons expliqué dans les chapitres précédents. Alors, nous nous sommes intéressés à l'application de la simulation pour l'extraction des connaissances à partir de l'optimisation via simulation.

Dans notre contexte, l'optimisation via simulation consiste à améliorer progressivement la logique décisionnelle du réseau de neurones, et ainsi, déterminer le vecteur W qui permettra au réseau de neurones de prendre les bonnes décisions au regard de la performance visée.

Dans la section qui suit nous rappelons brièvement les principes de l'optimisation via simulation et son intérêt pour notre démarche d'auto apprentissage du réseau de neurones.

3.2.1. L'optimisation via simulation

L'optimisation des systèmes de production consiste souvent à déterminer la meilleure combinaison des paramètres qui les caractérisent. Certains de ces paramètres sont numériques comme les capacités de stockage, le nombre de boucles kanban, la taille des lots de transport ou non numériques comme les règles de priorité.

Il y a souvent plusieurs interactions entre ces paramètres (par exemple, certaines tailles de lots peuvent engendrer de bons résultats avec une règle de priorité donnée et de mauvais résultats avec une autre).

Le but et n'est pas seulement d'évaluer ces paramètres mais aussi de les optimiser, il est difficile d'utiliser des approches analytiques sans faire d'hypothèses restrictives. La solution la plus adaptée pour permettre leur évaluation est souvent la simulation car elle permet de prendre en

compte au mieux les diverses caractéristiques des systèmes de production (Law and Kelton, 2000).

Toutefois, la simulation n'est qu'un outil d'évaluation de la performance d'un système en fonction de ses paramètres. Il faut donc utiliser la simulation conjointement à d'autres méthodes afin de rechercher les solutions optimales ; d'où l'optimisation via simulation.

Le principe de l'optimisation via simulation (Azadivar 1992, Fu 1994, Andradottir 1998, Fu 2002) consiste en une interaction entre les deux modules : le module de simulation et le module d'optimisation ; où le module d'optimisation propose successivement des solutions que l'on espère de plus en plus performantes au module de simulation ; celui-ci les évalue en calculant le comportement du système ainsi paramétré, ceci jusqu'à la satisfaction d'un critère d'arrêt (nombre de générations, valeur de l'optimum atteint,…). Le problème considéré est donc la maximisation ou la minimisation de la valeur de la fonction objectif.

Ce problème peut être exprimé de la façon suivante :

$$(\min)\max_{X\in\Theta} H(X) \qquad \textbf{(17)}$$

Où $H(X)= L(X, \varepsilon)$ est la mesure de performance du système,

X est un vecteur de p facteurs contrôlables,

Θ est l'ensemble des contraintes de X,

ε représente les effets stochastiques du système,

De nombreux travaux ont été menés en optimisation via simulation, ils diffèrent souvent par la méthode d'optimisation employée qui est étroitement liée au contexte et aux objectifs de l'étude. Nous citons par exemple les études menées qui cherchent à configurer au mieux selon divers critères de performance tels que le temps de fabrication, le nombre d'en-cours ou le nombre de demandes mises en attente (Krajewski et al. 1987, Paris et al.2001).

Plusieurs méthodes d'optimisation peuvent être utilisées conjointement à la simulation. Les méthodes d'optimisation peuvent être classées de différentes manières : nous les classerons en méthodes déterministes et méthodes non-déterministes. Les méthodes déterministes sont généralement efficaces quand l'évaluation de la fonction est très rapide, ou quand la forme de la fonction est connue à priori. Les cas plus complexes (temps de calcul important, nombreux optima locaux, fonctions non-dérivables, …) seront souvent traités plus efficacement par des méthodes non-déterministes. Les méthodes non-déterministes font appel à des tirages de nombres aléatoires. Elles permettent d'explorer l'espace de recherche plus efficacement.

Dans le cas de l'optimisation via simulation, la fonction n'est pas calculée mais c'est le résultat d'une simulation. Alors, le temps de calcul peut être très important.

Dans ce qui suit nous rappelons quelques méthodes d'optimisation non-déterministes.

3.2.1.1.Le recuit simulé

La méthode du recuit simulé a été introduite par (Kirkpatrick et al.1983). Le recuit simulé est une métaheuristique inspirée d'un processus utilisé en métallurgie. Il est basé sur l'utilisation de cycles de chauffage et de refroidissement d'un métal qui tendent à améliorer sa qualité en jouant sur l'organisation des molécules afin d'obtenir une structure régulière. On parle alors d'arrangement cristallin. Il est aujourd'hui utilisé en optimisation pour rechercher les optimas locaux d'une fonction.

Cette méthode d'optimisation s'appuie sur les travaux de (Metropolis 1953) qui permettent de décrire l'évolution de l'équilibre thermodynamique d'un système, la fonction « objectif » à minimiser étant l'énergie E du matériau. Partant d'une solution initiale, on génère une solution proche de manière aléatoire afin d'obtenir un voisin de l'état de départ. Soit celui-ci améliore le critère que l'on cherche à optimiser, soit il le dégrade.

Si la solution améliore la fonction « objectif », cette dernière est automatiquement acceptée. En réitérant le processus, on tend ainsi à chercher l'optimum dans le voisinage de la solution de départ. Si le voisin ne rentre pas dans la fonction « objectif », il peut quand même être accepté suivant une probabilité p. L'acceptation d'une « mauvaise » solution permet alors d'explorer une plus grande partie de l'espace de solution et tend à éviter de s'enfermer trop vite dans la

recherche d'un optimum local. On introduit également un paramètre fictif, la température T. Cette dernière est plutôt élevée au début de l'algorithme et permet de se dégager des optimas locaux. Au fur et à mesure que les itérations de l'algorithme se font, la température diminue selon une stratégie propre au problème que l'on souhaite résoudre afin d'intensifier la recherche du point optimal mais cette fois-ci localement.

Collins et al. (Collins et al. 1988) et Hajek (Hajek 1988) ont proposé plusieurs températures. Des informations complémentaires peuvent être trouvées, par exemple, dans Van Laarhoven et Aarts (Van Laarhoven et al. 1987) ou Eglese (Eglese 1990). Le recuit simulé est largement utilisé en optimisation via simulation. Manz et al. (Manz et al. 1989) optimisent un système de production automatisé par recuit simulé. Haddock et Mittenthal (Haddock et al. 1992) utilisent le recuit simulé pour maximiser le profit dans un système de production automatisé comprenant 4 machines desservies par un carrousel. Dans les travaux de Jayaraman et Ross (Jayaraman et al. 2003), le recuit simulé est utilisé pour optimiser la conception d'un système de distribution ainsi que ses futures stratégies d'utilisation.

3.2.1.2. Les algorithmes évolutionnistes

Les algorithmes évolutionnistes (AE) sont des méthodes heuristiques de recherche qui mettent en application les idées basées sur le processus d'évolution naturelle (Spear et al. 1993, Reeves 1993). Les algorithmes évolutionnistes recouvrent diverses méthodes telles que les algorithmes génétiques (Holland 1975, Goldberg 1989), la programmation génétique, les stratégies évolutionnistes ou la programmation évolutionniste. Il n'existe cependant aucune liste précise décrivant l'ensemble des AE dans la mesure où il n'existe aucune définition standard (Bäck 1996). Contrairement aux solutions simples utilisées dans des méthodes traditionnelles, les AE travaillent sur une population des solutions de façon à ce que les solutions faibles disparaissent, tandis que les bonnes solutions évoluent pour atteindre l'optimum. Chaque solution, appelée aussi « individu », est codée via un vecteur X_n (appelé «chromosome » et constitué de n «gènes») auquel sera associée une performance, appelée «fitness ». Dans le cas d'une approche d'optimisation via simulation, ce vecteur est constitué des paramètres d'entrée du modèle de simulation et sa performance est un résumé du comportement du système ainsi paramétré (Pflug 1984, Pierreval et Tautou 1997). Classiquement, ces algorithmes commencent leur recherche d'une solution optimale à partir d'un ensemble de solutions potentielles, appelé « population ». Les AE vont faire évoluer cette population vers une population supposée contenir la meilleure

solution. Pour cela, ils utilisent le cycle évaluation-reproduction suivant : à chaque itération, appelée « génération », certaines solutions sont sélectionnées en fonction de leur performance à partir de la population courante. Ces solutions vont être soumises à des opérateurs de mutation et de recombinaison, processus appelé «reproduction». La recombinaison va permettre un mélange des informations portées par les parents qui seront transmises aux descendants ; la mutation va introduire des éléments nouveaux dans la population. D'une génération à l'autre, la population est constituée de solutions globalement plus performantes. Il y a un intérêt important pour les AE dans le cadre de l'optimisation via simulation notamment parce qu'ils n'exigent pas d'hypothèses restrictives ou de connaissances à priori sur la forme de la surface de réponse (Bäck et Schwefel 1993). Parmi les différentes études dans lesquelles les algorithmes évolutionnistes obtiennent de bons résultats, on peut citer Tautou et Pierreval (1995) ou Dolgui et al. (2000). Dans Azadivar et Tompkins (1999), on voit que l'algorithme génétique utilisé permet l'optimisation non seulement de variables numériques classiques comme le nombre de machines mais aussi de variables qualitatives ou même de la structure même du système. Paris et al. (2001) utilisent la parallélisation d'algorithmes évolutionnistes pour optimiser la configuration d'un système kanban multi-produit. Bertel et Billaut (2003) comparent une approximation par borne inférieure, un algorithme 'liste' et un algorithme génétique pour la résolution d'un problème d'ordonnancement d'un flow-shop hybride à deux machines avec re-circulation. Ces travaux soulignent l'efficacité de l'algorithme génétique. Alam et al. (2003) optimisent la planification du process d'une entreprise de fabrication de moules à injecter avec un algorithme génétique. Pierreval et al. (2003) font une étude des diverses approches évolutionnistes utilisées dans la conception et l'organisation de systèmes de production. De plus, il existe des adaptations d'algorithmes génétiques pour l'optimisation multiobjectif comme dans Alberto et al. (2002). Cependant, les AE sont aussi reconnus pour être assez lents et pour fonctionner comme des 'boîtes noires' dans le sens où ils n'apportent rien d'autre qu'une solution que l'on espère optimale ou presque (Huyet, 2004).

3.2.1.3. La recherche tabou

La recherche tabou est une procédure de recherche contrainte, où chaque étape consiste à résoudre un deuxième problème d'optimisation. En effet, à chaque étape, la procédure de recherche exclut un sous-ensemble de l'espace de solutions. Ce sous-ensemble change durant l'exécution de l'algorithme et est habituellement constitué par les solutions précédemment considérées. Des informations complémentaires peuvent être trouvées dans Glover (1989) ou

dans Glover et Laguna (1997). 'Un certain nombre de travaux utilisent des procédures de recherche tabou pour l'optimisation via simulation. Hu (1992) étudie la fiabilité et l'efficacité d'un algorithme de recherche tabou sur quelques fonctions de test standard et signale qu'il surpasse la recherche aléatoire et l'algorithme génétique utilisé sur ces tests. Garcia et Bolivar (1999) développent un système de simulation qu'ils appellent 'simulateur de système d'inventaire stochastique' et optimisent cinq modèles stochastiques d'inventaire avec différentes distributions de probabilité de demandes et de délais d'exécution par recherche tabou. Lutz et al. (1998) traitent le problème du lieu et la taille de stockage dans une ligne de fabrication et l'optimisent par recherche tabou. Martin et al. (1998) implémentent quatre variantes de la recherche tabou pour déterminer le nombre de kanbans et les tailles de lots dans un système kanban générique. Dengiz et Alabas (2000) appliquent également un algorithme de recherche tabou à un modèle de simulation d'un système JIT pour trouver le nombre optimal de kanbans (Huyet, 2004).

3.2.1.4. Algorithme d'échantillonnage

Cet algorithme a été développé par Kushner (1963). La stratégie de recherche est la suivante : à chaque itération, la prochaine solution est choisie pour être le point qui maximise la probabilité de ne pas dépasser la valeur précédente d'une certaine constante positive Yn. Pour un problème de minimisation, cette méthode choisit des solutions dans les zones où la valeur de la fonction d'évaluation est basse. Au début, Y_n est petite mais augmente à mesure que la recherche devient plus locale afin d'accélérer la convergence et d'éviter d'être piégée. Stuckman et Easom (1992) donnent un aperçu des méthodes existantes basées sur ce principe d'échantillonnage (i.e., la méthode de Stuckman, la méthode de Mockus, la méthode de Perttunen) et les comparent à d'autres méthodes telles que l'analyse de sensibilité sur des fonctions de variables continues. Dans leurs résultats, il s'avère que ces méthodes surpassent l'analyse de sensibilité (Huyet, 2004).

3.2.1.5. Méthode de surface de gradient

Ho et al. (1992) ont proposé la méthode de surface de gradient pour l'optimisation des systèmes dynamiques à événements discrets. Cette méthode diffère d'autres techniques globales de recherche notamment parce qu'elle emploie des méthodes 'traditionnelles' de recherche pour explorer globalement une surface de réponse. Elle combine les avantages de la méthodologie de surface de réponse et des techniques d'estimation de la dérivée efficaces telles que l'analyse de perturbation et les estimateurs du maximum de vraisemblance en conjonction avec des

algorithmes d'approximation stochastique. Dans la méthode de surface de gradient, l'estimation du gradient est obtenue par analyse de perturbation ou par estimateurs du maximum de vraisemblance, et la surface de gradient est déduite de ces estimations par des méthodes des moindres carrés comme dans la méthodologie de surface de réponse. Les points zéro de la surface de gradient sont alors pris comme solution optimale. La méthode de surface de gradient est une méthode de recherche globale parce qu'à chaque itération, elle emploie l'information de tous les points plutôt que juste le gradient local. L'avantage de cette méthode est qu'elle obtient les estimations du gradient en une seule exécution, et elle permet d'accéder rapidement à la proximité de la solution optimale en raison de son orientation globale. Ho et al. utilisent cette méthode sur six exemples de réseaux de files d'attente, mais ne fournissent pas de résultats comparatifs par rapport à d'autres méthodes (Huyet, 2004).

3.2.2. Le principe d'auto apprentissage des réseaux de neurones

Dans notre approche, nous n'avons pas seulement des seuils numériques à optimiser, mais plus globalement l'ensemble de la logique de décision que nous avons représentée par le vecteur numérique *W,* ce vecteur peut être optimisé par une méthode classique d'optimisation dans R^n parmi celles que nous avons cité dans la section précédente tel que le recuit simulé, la recherche tabou, les algorithmes génétiques, etc.

Cependant, pour évaluer la logique exprimée par le réseau de neurones nous ne pouvons pas nous baser sur une fonction explicite ; la complexité et la difficulté de représenter de telles décisions de pilotage dans les systèmes de production, ne nous permet pas de déterminer une fonction mathématique fixe et efficace pour mesurer la performance des décisions du réseau de neurones. Pour cela, la performance de la logique du réseau de neurones à prendre les décisions de pilotage d'un atelier donné sera toujours évaluée via simulation.

Ainsi, pour optimiser la logique exprimée par le réseau de neurones, nous nous basons sur: l'optimisation via simulation. Dans notre approche, nous recherchons la meilleure logique définie par *W* de prise de décision du réseau de neurones (Figure 24) au regard de notre fonction objectif *f*.

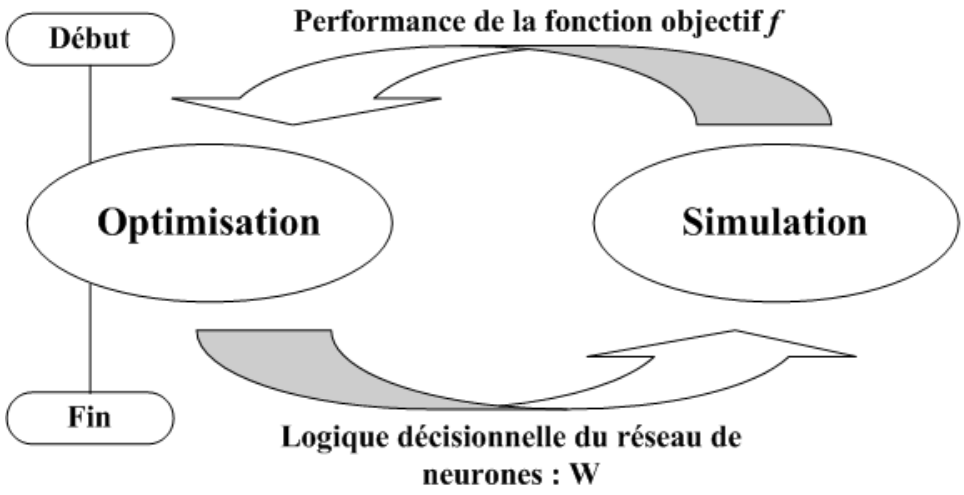


Figure 24. Optimisation de la logique de prise de décision via simulation

L'approche consiste donc à «extraire », à partir du comportement dynamique d'un modèle de simulation de l'atelier, des connaissances de pilotage. Le modèle de simulation permettant d'évaluer si les actions de pilotage sont bonnes, les corrections nécessaires peuvent s'effectuer au niveau des caractéristiques *W* (logique) du réseau de neurones. L'apprentissage consiste donc à adapter les caractéristiques du réseau de neurones en fonction des performances des décisions prises, évaluées par simulation, que l'on va chercher à optimiser. Ainsi, l'apprentissage peut se voir comme un problème d'optimisation via simulation, dans la mesure où il vise à minimiser ou maximiser une ou plusieurs fonction(s) objectifs qui caractérise(nt) la performance de l'atelier, calculées par simulation, en utilisant les caractéristiques du réseau de neurones comme variables de décision du problème (Wetter *et al.*, 2003). Une fois que le réseau de neurones a appris, il peut être alors intégré dans le système de pilotage global du système de production considéré. La démarche globale est résumée dans la figure 25.

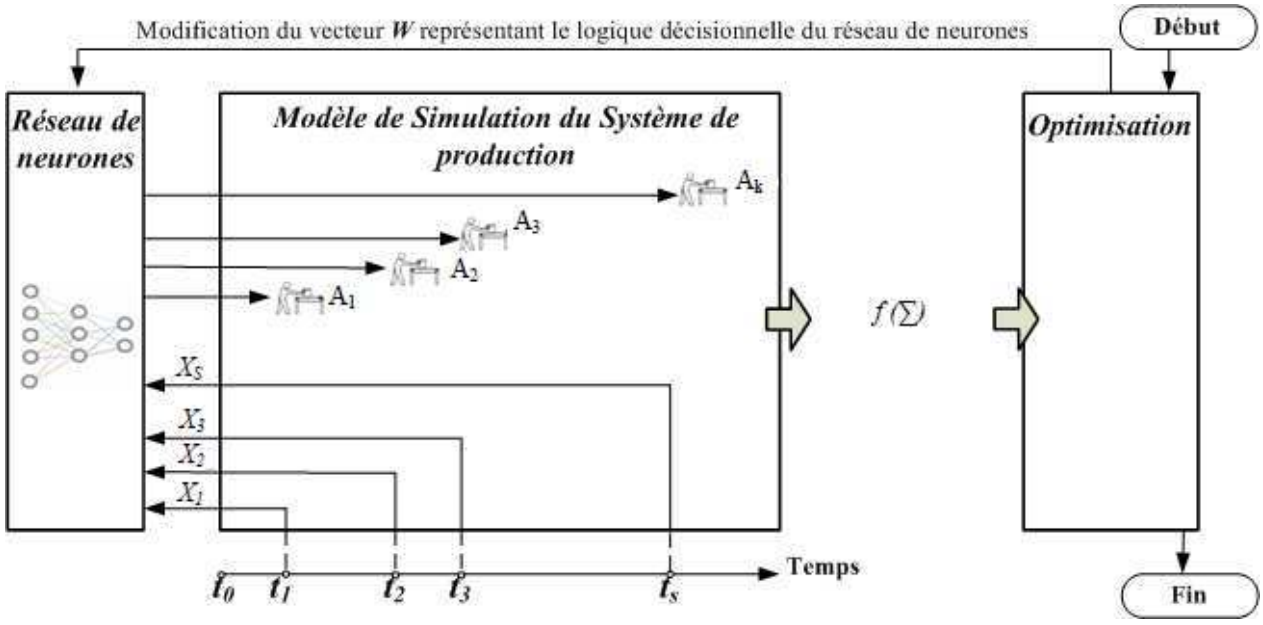


Figure 25. La méthode globale de l'apprentissage des réseaux de neurones par optimisation via simulation.

4. Implémentation

Nous distinguons trois modules : le module du réseau de neurones, le module de la simulation et le module de l'optimisation. Dans la première phase de notre approche de construction de la logique décisionnelle du réseau de neurones, les trois modules interagissent (figure 26). Cependant, dans la deuxième phase d'intégration de la logique décisionnelle dans le système de production pour le pilotage, seulement le module de simulation du système de production étudié et le module du réseau de neurones communiquent (figure 27).

A chaque fois qu'une décision de pilotage doit être prise à un instant t, la simulation communique au réseau de neurones les caractéristiques du système et son état W. Ce dernier lui renvoie sa décision A_i. Alors, la simulation applique cette décision. A la fin de la simulation, la valeur de la fonction objectif est calculée et communiquée au module d'optimisation qui modifie ainsi les caractéristiques du réseau de neurones, pour optimiser la fonction objectif (figure 25).

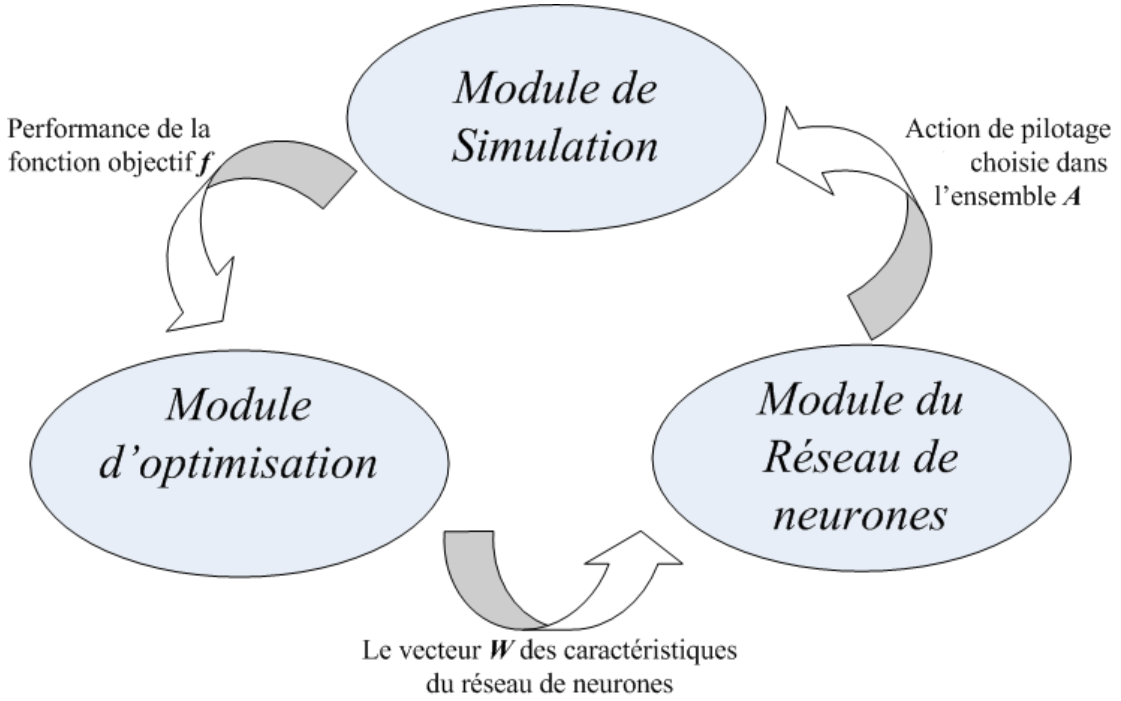


Figure 26. Interaction entre les trois modules (la première phase de l'approche proposée)

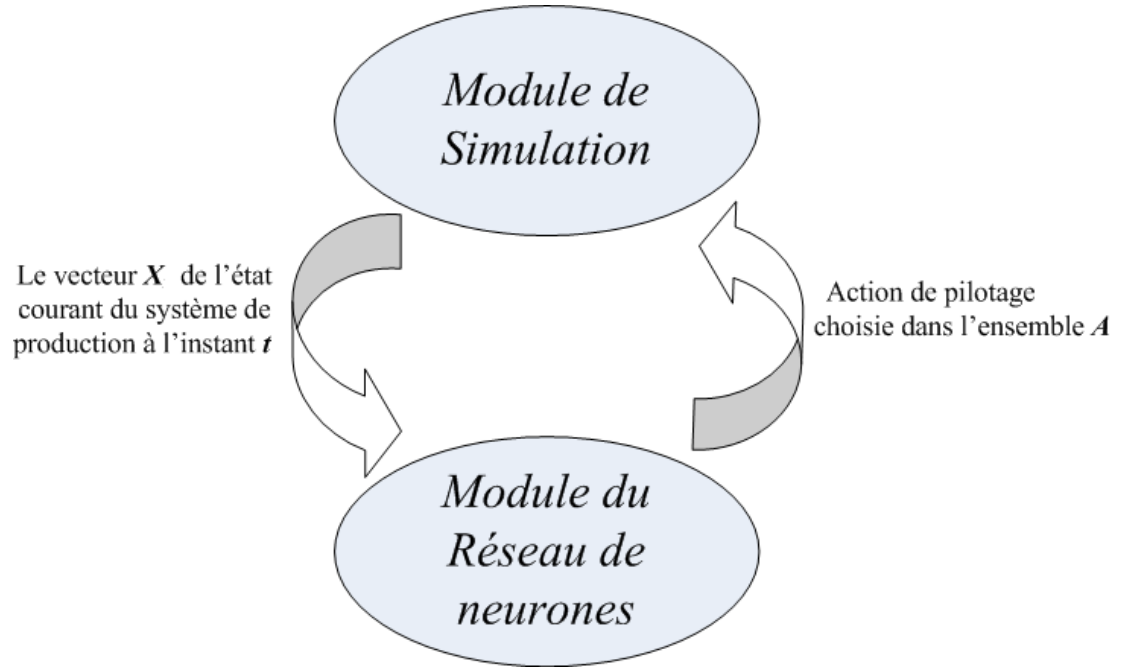


Figure 27. Interaction entre les deux modules (la deuxième phase de l'approche proposée)

4.1. Module de simulation

Pour réaliser et construire le module de simulation nous avons utilisé le logiciel de simulation ARENA dans sa version 10.

La description du modèle du système simulé se fait à l'aide d'un assemblage constitué de mise en série, en parallèle ou en *feedback* de différents blocs fonctionnels, issus de bibliothèques (*templates*) d'ARENA. Une telle approche de modélisation permet d'obtenir une structure du modèle proche de celle du système (réel) à simuler.

Dans ce qui suit nous présentons quelques notions de base d'utilisation du logiciel ARENA pour la construction d'un modèle de simulation d'un système de production donné.

4.1.1. Notions de base

Entité

Une *entité* est un objet qui évolue dans les différents blocs fonctionnels constituant le modèle du système. Elle correspond en général à un objet concret, par exemple, une personne ou une pièce dans un atelier. Le déplacement des entités au sein des différents blocs - par exemple le déplacement de pièces dans un atelier - provoque un changement d'état du modèle de simulation.

Attribut

Un *attribut* est une variable associée *individuellement* aux entités (la variable est locale) pour représenter leurs états ou des paramètres qui leur sont propres. Par exemple, chaque entité, représentant une pièce circulant dans un atelier, peut avoir les attributs suivants :

- *Type_de_piece* afin de désigner le type d'une pièce (par exemple, *Type_de_piece =A* ou *B*) ;

- *Indice_de_priorite* afin de désigner l'indice de priorité d'une pièce (par exemple, *Indice_de_priorite = faible* ou *importante*) ;

- *Date_arrivee_ds_le_modele* (par exemple, *Date_arrivee_ds_le_modele = TNOW*).

Variable globale

Une *variable globale* concerne l'ensemble du modèle. Par exemple, la variable *TNOW* (variable prédéfinie dans SIMAN) désigne la date à laquelle se trouve la simulation, c'est le temps courant - mis à jour à chaque avancée dans l'échéancier des événements – s'écoulant durant une simulation du modèle.

Le principe de fonctionnement du logiciel ARENA est de suivre *chacune des entités* évoluant d'un bloc fonctionnel vers un autre dans le modèle, de sa création à sa destruction. L'ordonnancement dans le temps des différents événements rattachés à l'évolution des entités dans les blocs constituant le modèle se fait au travers d'un *échéancier*.

Quand une entité est introduite dans un bloc fonctionnel, elle déclenche le « service » qui lui est associé, ce qui provoque une modification de l'état du modèle. Un « service » peut agir :

- Sur l'entité au travers de la valeur de ses attributs. Par exemple, à travers un bloc ***Assign***, on peut affecter à l'attribut *indice_de_priorite* d'une entité représentant une pièce, présente dans le bloc, la valeur *importante* ;

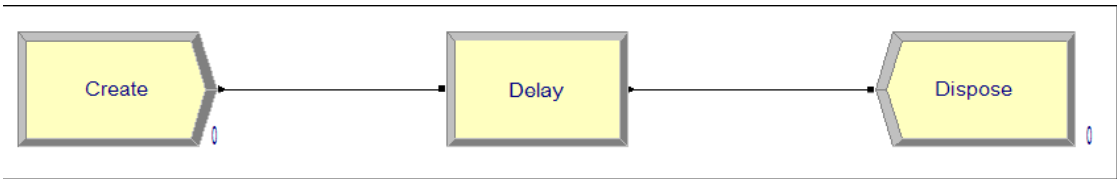
- Sur les variables globales du modèle. Par exemple, le passage d'une entité dans un bloc ***Delay*** provoque un retard pur, ce qui aura une conséquence sur la variable *TNOW*.

Un modèle élaboré avec ARENA est sauvegardé dans un fichier ayant pour extension *.doe* et est constitué :

- D'une partie *modèle*, qui représente l'algorithme décrivant les caractéristiques statiques et dynamiques des différents blocs fonctionnels composant le modèle ;

- Du *cadre expérimental*, qui regroupe les données précisant les paramètres spécifiques à une simulation donnée (conditions initiales, durée de la simulation, …).

En fait, les entités traversent uniquement les blocs fonctionnels de la partie *modèle*. Considérons un simple tapis roulant, ayant un temps de transport de 3 unités de temps, représenté par le modèle décrit comme suit :



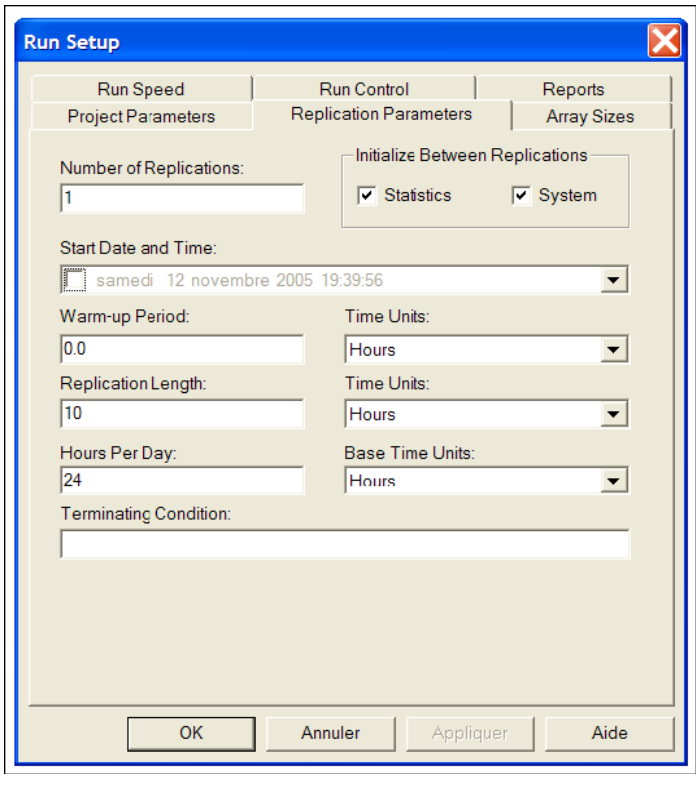
Le bloc ***Create***, issu du *template Basic Process*, est tel qu'une entité est créée à partir de l'instant 0, ceci toutes les 2 unités de temps.

Le bloc ***Delay***, issu du *template Advanced Process*, force une entité à séjourner 3 unités de temps dans le bloc.

Le bloc ***Dispose***, issu du *template Basic Process*, détruit toute entité entrant dans le bloc. A travers le menu *Run/Setup/Replication Parameters*, on peut notamment fixer :

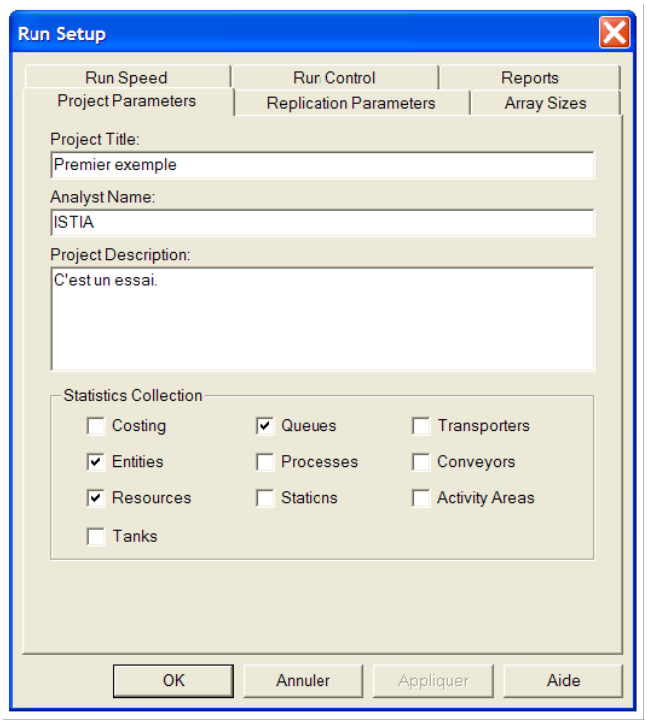
- Le nombre de réplications (champ *Number of Replications*),

- Le temps où se termine une réplication (champ *Replication Length*).



A travers le menu *Run/Setup/Project Parameters*, on peut notamment donner :

- Un titre au projet (champ *Project Title*),
- Le nom du programmeur (champ *Analyst Name*),
- Un commentaire (champ *Project Description*).



Les 2 fichiers générés par SIMAN-ARENA (au format *txt*) sont accessibles *via* le menu *Run/SIMAN/View* (voir ci-dessous un listage partiel de ces fichiers).

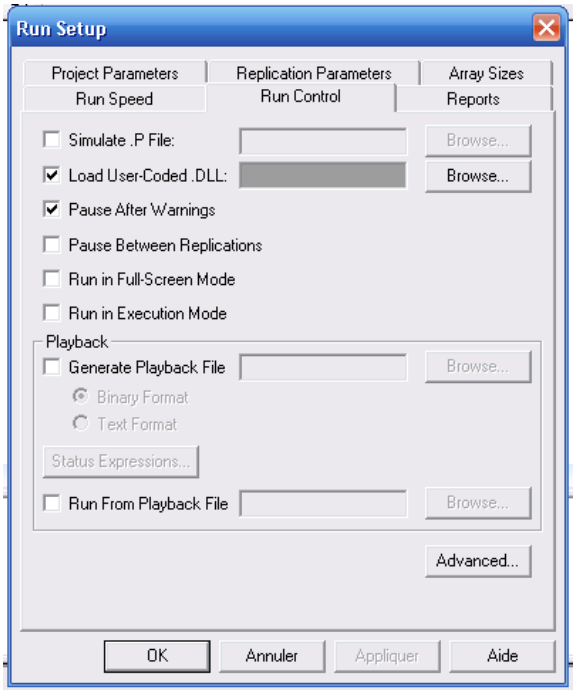
- Fichier ***Exemple.mod* : (*partie modèle*).**
- Fichier ***Exemple.exp* : (*cadre expérimental*).**

ARENA permet de construire un modèle en proposant des primitives de représentation (appelées par la suite, blocs ou modules) plus ou moins détaillées. Il permet également de créer des animations graphiques pour visualiser le comportement du modèle durant la simulation. Les blocs sont regroupés dans différentes bibliothèques (*templates*).

Afin de bénéficier d'une animation, nous allons utiliser les *templates Basic Process* et *Advanced Process*. Sont décrits en « Annexe A » les *templates Blocks* et *Elements* lesquels contiennent des blocs plus élémentaires (à chacun de ces blocs correspond une ligne dans les fichiers générés par SIMAN-ARENA). Nous présentons plus de détails sur l'utilisation du logiciel ARENA dans l'Annexe A.

4.1.2. Lien entre les modèles ARENA et les fichiers DLL

Une des particularités du logiciel ARENA que nous utilisons dans notre approche est sa capacité de communiquer avec des programmes et des logiciels externes.
A partir de l'environnement d'ARENA nous manipulons des fichiers DLL écrits à partir d'un environnement Visual Studios C++. En effet, le logiciel permet l'appel des fichiers DLL pour l'exécution d'un certain nombre de calculs externes qui sont effectués pendant la simulation et qui peuvent utiliser plusieurs données telles que les variables globales, les attributs des pièces traitées dans le modèles, des variables système comme TNOW, etc.
Pour établir la connexion entre le modèle ARENA et les fichiers DLL le logiciel offre la fonctionnalité « Load user-coded .DLL », cette option nous permet d'incorporer des modules codés par l'utilisateur dans l'environnement d'ARENA pour l'utiliser dans des modèles particuliers (ex : calculs externes, importation de données calculées parallèlement lors de la simulation). Une fois cette option sélectionnée nous devons spécifier le nom de notre fichier DLL dans la case qui correspond.



Pour mettre en œuvre cette option, ARENA offre des fonctions prédéfinies qui permettent la communication entre le fichier DLL et le modèle de simulation. Nous énumérons ci-dessous quelques fonctions et leurs utilités :

- « **cevent** » : Elle a comme argument le numéro du bloc « **Event** » du modèle ARENA et l'index de l'entité à traitée, son exécution est valable lors du passage d'une entité par ce bloc « **Event** ».
- « **cprime** » : Cette fonction est appelée par le modèle ARENA au début de chaque réplication de la simulation pour établir les conditions initiales.

Alors, pour créer notre fichier DLL, nous nous sommes basé sur la propriété d'ARENA, d'interagir avec l'environnement C++.

4.2. Module du réseau de neurones

Un des problèmes majeurs des réseaux de neurones a trait à la difficulté de décider de leur architecture : nombre de neurones cachés, nombre de couches cachées, leur poids, leurs fonctions d'activation, et leurs poids. Devant une tâche à résoudre par un réseau de neurones il faut décider quelles seront ses caractéristiques. Ceci se fait souvent de façon ad hoc ; on essaye plusieurs combinaisons de caractéristiques pour un problème donné et on choisit la meilleure combinaison. En effet, aucune méthode n'est connue pour déterminer l'architecture optimale pour un problème donné (Cornuéjols et Miclet, 2003).

Dans cette section nous décrivons le réseau de neurones que nous avons utilisé ainsi que les caractéristiques que nous avons choisies. Nous détaillons son architecture, la ou les fonction(s) d'activation que nous avons utilisées et nous décrivons ses entrées et ses sorties pour le problème que nous traitons.

4.2.1. Architecture du réseau de neurones

Pour construire un réseau de neurones, il faut tout d'abord définir, le nombre de couches qui le composent, les liaisons existantes entre les différents neurones et le nombre de neurones présents dans chaque couche.

Notre réseau de neurones est constitué de trois couches (voir figure 28):

- *Une couche d'entrée* composée des neurones qui représentent le vecteur X des paramètres D et de l'état courant de l'atelier $S(t)$ lors de la prise de décision et qui sont communiqués par de module de la simulation,

- *Une couche cachée* : Le nombre de neurones cachés joue un rôle crucial dans le contrôle de la capacité du réseau de neurones. Si le nombre de neurones cachés est trop petit, alors le réseau possède trop peu de poids et ne peut capter toutes les dépendances qui servent à modéliser et prédire les valeurs du processus observé. À l'inverse, si l'on augmente le nombre de neurones cachés, alors le nombre de poids du modèle augmente et il devient possible, pendant la phase d'optimisation, de modéliser certaines relations qui ne sont que

le fruit de fluctuations statistiques propres à l'ensemble d'entraînement utilisé plutôt que des relations fondamentales de dépendance entre les variables.

- *Une couche de sortie* composée des neurones qui représentent respectivement les actions A_i à transmettre au module de la simulation.

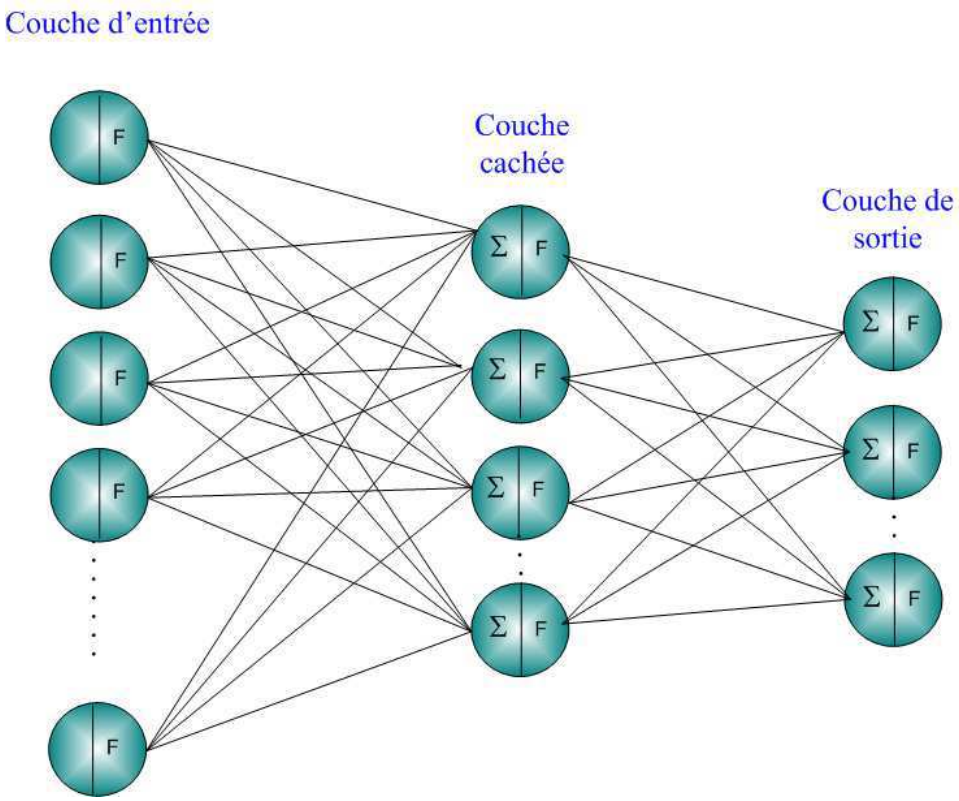


Figure 28. Architecture du réseau de neurone utilisé dans notre approche.

Les liaisons entre les neurones d'une couche à la couche suivante ont été choisies comme complètes : chaque neurone de la couche N est relié à tous les neurones de la couche N+1. Les couches sont reliées entre elles par des poids qui constituent notre vecteur W qu'on cherche à optimiser. En effet, pour chaque couche, les données d'entrées X sont multipliées avec le poids synaptique de chaque arc w_{ij} (voir figure 29), liant le neurone i de la couche N au neurone j de la couche N+1. Après cette multiplication, nous additionnons les produits liés au même neurone de la couche N+1, pour obtenir la valeur des neurones de la couche N+1. Ensuite, la valeur est transformée à travers une fonction d'activation φ qui limite les valeurs dans un intervalle borné, ses bornes dépendent de la fonction d'activation utilisée (voir figure 29).

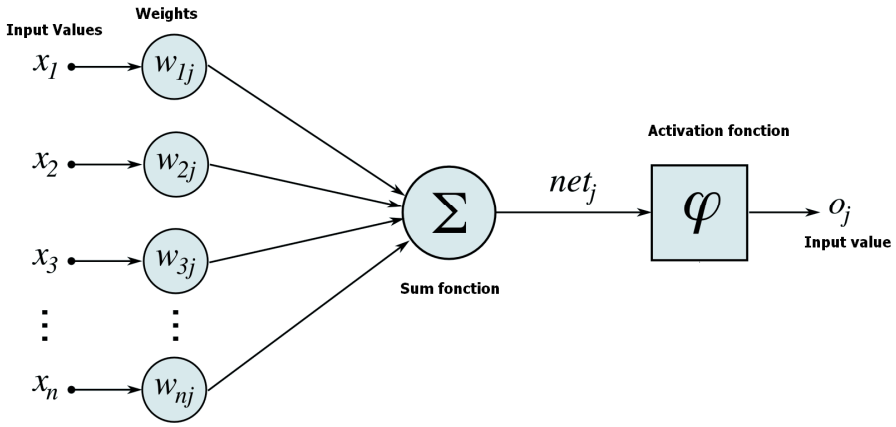
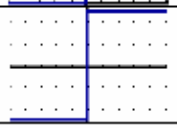
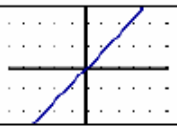
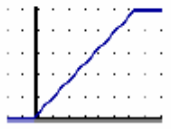
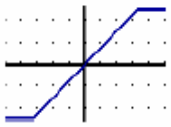
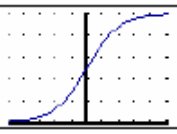
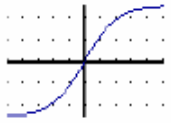


Figure 29. Le Principe du calcul du réseau de neurones

4.2.2. Fonction d'activation

Les types de fonctions d'activation (ou fonctions de transfert) sont très nombreux et dépendent du type des données et de l'architecture du réseau. Voici un récapitulatif des fonctions d'activation les plus utilisées :

Tableau 3. Tableau récapitulatif des différents types de fonctions d'activation les plus utilisées

Catégorie	Type	Relation	Allure	Dérivées
Seuil	Binaire (fonction de Heaviside)	$f(x)=\begin{cases}0 \text{ si } x<0\\1 \text{ si } x\geq 0\end{cases}$		-
	Signe	$f(x)=\begin{cases}-1 \text{ si } x<0\\1 \text{ si } x\geq 0\end{cases}$		-
	Stochastique (binaire ou signe) *(T est appelé température)*	$f(x,T)=1$ avec la probabilité $\frac{1}{1+e^{-\frac{x}{T}}}$	-	-
Linéaire	Identité	$f(x,k)=k.x$		$f'(x,k)=k$
	Saturé positif	$f(x,k)=\begin{cases}0 \text{ si } x<0\\1 \text{ si } x\geq \frac{1}{k}\\k.x\end{cases}$		$f(x,k)=\begin{cases}0 \text{ si } x<0\\0 \text{ si } x\geq \frac{1}{k}\\k\end{cases}$
	Saturé symétrique	$f(x,k)=\begin{cases}-1 \text{ si } x<-\frac{1}{k}\\1 \text{ si } x\geq \frac{1}{k}\\k.x\end{cases}$		$f(x,k)=\begin{cases}0 \text{ si } x<-\frac{1}{k}\\0 \text{ si } x\geq \frac{1}{k}\\k\end{cases}$
Sigmoïde	Positive (type logistique)	$f(x,k)=\frac{1}{1+e^{-k.x}}$		$f'(x,k)=\frac{k}{2+e^{-k.x}+e^{k.x}}$
	Symétrique (type tanh)	$f(x,k)=\frac{2}{1+e^{-k.x}}-1$		$f'(x,k)=\frac{2.k}{2+e^{-k.x}+e^{k.x}}$

Nous utiliserons une fonction tangentielle comme fonction d'activation :

$$f(x,k)=\frac{2}{1+e^{-kx}}-1 \qquad \textbf{(18)}$$

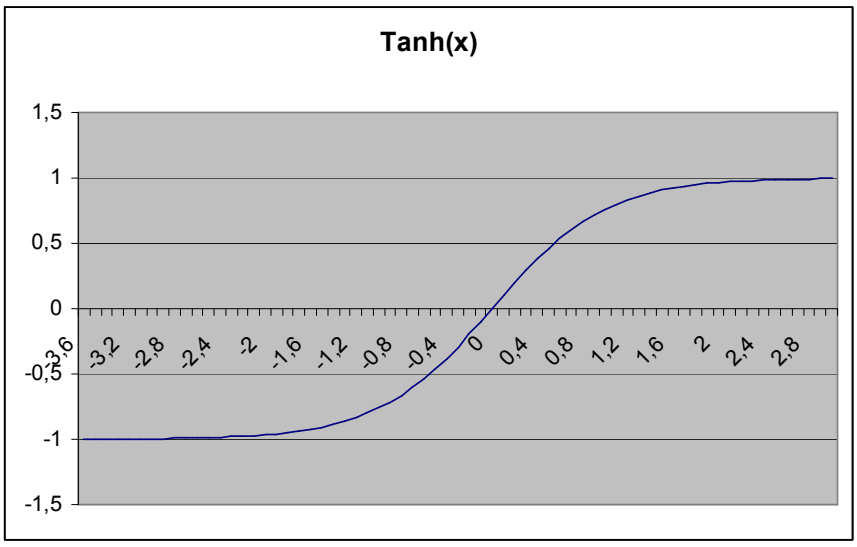


Figure 30. Courbe de la tangentielle : y = tanh(x)

Le module du réseau de neurones est développé avec Microsoft Visual C++ et connecté directement au modèle de simulation de l'atelier.

4.2.3. Adaptation des entrées/sorties du réseau de neurones aux données de la simulation

Les entrées et les sorties du réseau de neurones sont étroitement liées au type de problème traité et dans notre cas ils sont étroitement liées aux modèles de simulation puisque le réseau de neurones reçoit les données à partir du modèle de simulation et communique ses décisions aussi au modèle de simulation sous la forme qui convient. Nous décrivons dans ce qui suit les entrées du réseau de neurones ainsi que ses sorties.

Les entrées

Il est important d'évaluer distinctement les variables d'entrée du réseau de neurones issues du modèle de simulation. En effet, les données peuvent avoir besoin d'être converties sous une autre forme pour être significatives pour un réseau de neurones.

La bonne compréhension du problème pour connaître quel type d'information est appropriée, est la clef pour déterminer les variables d'entrée adéquates. Comment nos données sont représentées

et traduites joue également un rôle important dans la capacité du réseau de prendre les décisions convenables.

Les sorties

La troisième couche : la couche de sortie donne le résultat obtenu après compilation par le réseau de neurones des données d'entrée dans la première couche. Sa taille est directement déterminée par le nombre de variables qu'on veut en sortie. De même pour un problème de détection de client à risque ou pas dans un système bancaire, la couche de sortie peut être composée d'un seul neurone à valeurs binaires « 0 » pour dire que le client ne présente pas de risque pour le système et « 1 » sinon.

4.3. Module de l'optimisation

Dans le contexte d'optimisation dans lequel nous nous situons dans cette étude, plusieurs méthodes d'optimisation peuvent être utilisées, telles que le recuit simulé, les algorithmes évolutionnistes ou la recherche tabou. De plus, ces méthodes sont déjà largement exploitées dans le domaine avec succès. Une façon de choisir une méthode plutôt qu'une autre pourrait être de prendre en considération si elles ont ou non des paramètres extérieurs à régler.

Dans notre approche nous avons choisi la méthode du recuit simulé. Comme nous l'avons déjà définie dans la section 3 de ce chapitre cette méthode est basée sur une analogie avec un principe thermodynamique. A partir d'une solution initiale, une solution voisine est choisie et évaluée. Si la fonction d'évaluation est améliorée, cette solution devient la solution courante. Sinon, elle devient quand même la solution courante mais avec une probabilité qui décroît suivant une certaine loi et en fonction du nombre de solutions déjà étudiées. Ainsi, si la fonction d'évaluation est détériorée, la règle d'acceptation d'une solution Y voisine de Y' est basée sur une température T qui dépend du nombre de solutions **m** étudiées et s'exprime de la manière suivante :

Si un nombre choisi entre 0 et 1 est inférieur à $\exp\left(\frac{F(Y')-F(Y)}{T(m)}\right)$ alors on accepte la solution Y. La solution Y' est remplacée par Y qui devient la nouvelle solution courante.

Pour optimiser les caractéristiques *W* du réseau de neurones nous citons ci-dessous les différentes étapes de l'algorithme du recuit simulé que nous avons utilisé.

La solution initiale W_0 est déterminée aléatoirement. La perturbation des solutions est basée sur un principe de voisinage dynamique. Soit W_l le vecteur courant des caractéristiques du réseau de neurones, et l est l'itération courante du recuit simulé.

La solution voisine W_l**'** est définie en deux étapes tel que :

Etape 1 : sélectionner aléatoirement ***g(l)*** éléments *de* W_l

Etape 2 : pour chaque élément sélectionné, nous ajoutant un nombre aléatoire à partir d'une loi normale **N(0, *s*(T_l))**.

Dans l'algorithme que nous utilisons :

- $$g(l) = E\left[1 + \left((a-1) * e^{-l^2/T_0 * 10}\right)\right] \quad \textbf{(19)}$$

- $$s(T_l) = \left[\left(\sigma_{\max} - \sigma_{\min}\right)/\left(T_0 - \varepsilon\right)\right] * \left(T_l - T_0\right) + \sigma_{\max} \quad \textbf{(20)}$$

Avec:

s est une fonction qui prend ses valeurs dans l'intervalle [σ_{min} , σ_{max}],

E[z] est la partie entière du nombre réel z,

a est le nombre total des caractéristiques du réseau de neurones que nous voulons optimiser,

T_0 est la température initiale du recuit simulé,

T_l est la température courante,

ε est la condition d'arrêt de l'algorithme qui est initialement choisie.

Les fonctions ***g(l)*** et ***s(T_l)*** diminuent progressivement, tout au long des itérations du recuit simulé, avec la diminution de la température. Ainsi, les changements effectués au début de l'optimisation sont beaucoup plus importants que les changements effectués à la fin.

Cette méthode assure la connectivité de l'espace de recherche. Le taux de refroidissement et la température initiale ont été empiriquement choisis après plusieurs expériences. Le recuit simulé s'arrête quand la température est inférieure à **ε.**

Pour appliquer l'algorithme du recuit simulé à notre approche, il est évident de l'adapter pour qu'il interagisse avec le reste des modules.

D'abord, puisque la fonction objectif ***f*** n'est pas calculée par le recuit simulé mais communiquée par le module de simulation, il est donc nécessaire de gérer cet échange sur le plan informatique ; nous proposons alors un échange par un fichier externe « fichier performance » dans le quel la simulation écrit la performance calculée de ***f*** à chaque fois la simulation est terminée. Ce fichier sera alors lu par le recuit simulé en tant que valeur de la fonction à optimiser.

Ensuite, nous proposons aussi un échange par un fichier externe pour gérer l'échange entre le recuit simulé et le module du réseau de neurones ; en effet, le module du réseau de neurones utilise pour chaque calcul d'un choix d'action demandé par la simulation son fichier (externe) de caractéristiques « fichier caractéristiques du réseau de neurones ». Ce dernier, est modifié à chaque itération du recuit simulé, voir (figure 31).

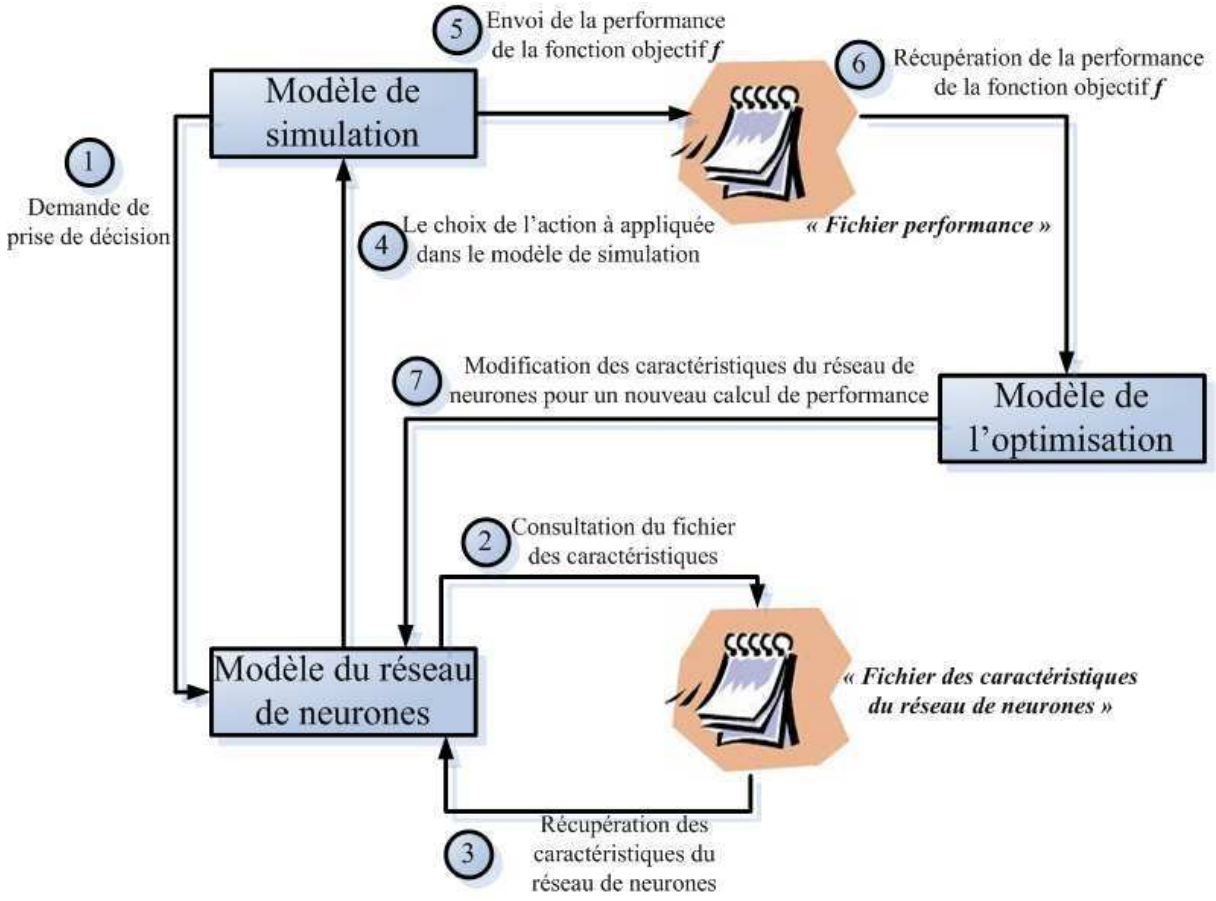


Figure 31. Flux de données entre les fichiers externes et les modules : Optimisation, Réseau de neurones et Simulation.

5. Conclusion

Nous avons donc proposé une approche qui assure un pilotage en temps réel des systèmes de production en se basant sur les réseaux de neurones, cette approche s'appuie sur un apprentissage automatique qui ne nécessite ni exemple d'apprentissage ni une expertise préalable concernant le pilotage de l'atelier que nous voulons optimiser la performance. Comme nous l'avons déjà précisé, le réseau de neurones est considéré comme le décideur (il gère le passage des pièces sur les machines, il affecte les bons opérateurs sur les machines libres,…) et il prend ses décisions en se basant sur l'état réel du système à chaque instant *t* de prise de décision. Notre approche, assure son apprentissage en se basant sur l'optimisation via simulation.

Cette approche est illustrée dans le chapitre suivant.

Chapitre IV

APPLICATIONS

1. Introduction

L'objectif de ce chapitre est de valider la faisabilité de l'approche de pilotage en temps réel basée sur les réseaux de neurones développée dans les précédents chapitres. Les performances du pilotage sont évaluées par simulation sous ARENA 10.0.

Dans la première partie de ce chapitre, nous appliquons notre approche pour le pilotage en temps réel à un atelier Flowshop. Nous avons choisi un atelier qui est déjà utilisé dans la littérature depuis 1986 par Barett et Barmann (Barett et Barman 1986). Dans ce contexte, il s'agit de choisir les stratégies de passage des pièces sur les machines de l'atelier pour optimiser la fonction objectif.

Dans la deuxième partie, nous déclinons et validons notre approche sur un autre exemple inspiré des travaux de Fonseca et Navaresse (Fonseca et al. 2002). C'est un atelier Jobshop sur lequel nous testons les capacités du réseau de neurones du pilotage en temps réel.

2. Exemple 1 : Application de l'approche d'apprentissage autonome des réseaux de neurones pour le pilotage d'un atelier Flowshop

2.1. Description de l'exemple

Le modèle étudié dans ce premier exemple d'application de notre approche est une reprise d'un modèle précédemment réalisé en 1986 par Barrett et Barmann. L'objectif de ces auteurs était alors d'étudier l'influence des règles de priorité sur les files d'attente d'un atelier flowshop. Nous avons repris ce même modèle pour permettre une comparaison des résultats (Mouelhi et al. 2007c).

L'atelier que nous avons considéré est un atelier flowshop constitué de deux centres d'usinage workcenter 1 (WC1) et workcenter 2 (WC2) réalisant l'un après l'autre l'usinage des pièces (Figure 32). Chaque centre de travail est composé de deux machines identiques. Les machines de

chaque centre sont capables de réaliser les mêmes opérations ; cependant, les opérations du workcenter 1 sont différentes de celles de workcenter 2.

À son arrivée dans l'atelier, chaque pièce, est traitée séquentiellement, d'abord dans workcenter 1 ensuite dans workcenter 2. Si les deux machines d'un centre de travail sont occupées alors la pièce attend dans la file d'attente en amont du centre de travail jusqu'à ce qu'une machine soit disponible.

Au niveau du workcenter 2, 10% des pièces usinées doivent subir une retouche sur l'une des deux machines du centre de travail. Ces pièces à retoucher sont prioritaires par rapport aux pièces en attente d'usinage.

Chaque machine après avoir usiné cinquante pièces, reste hors service pendant une durée de 0.5 unités de temps, puis sera remise en service.

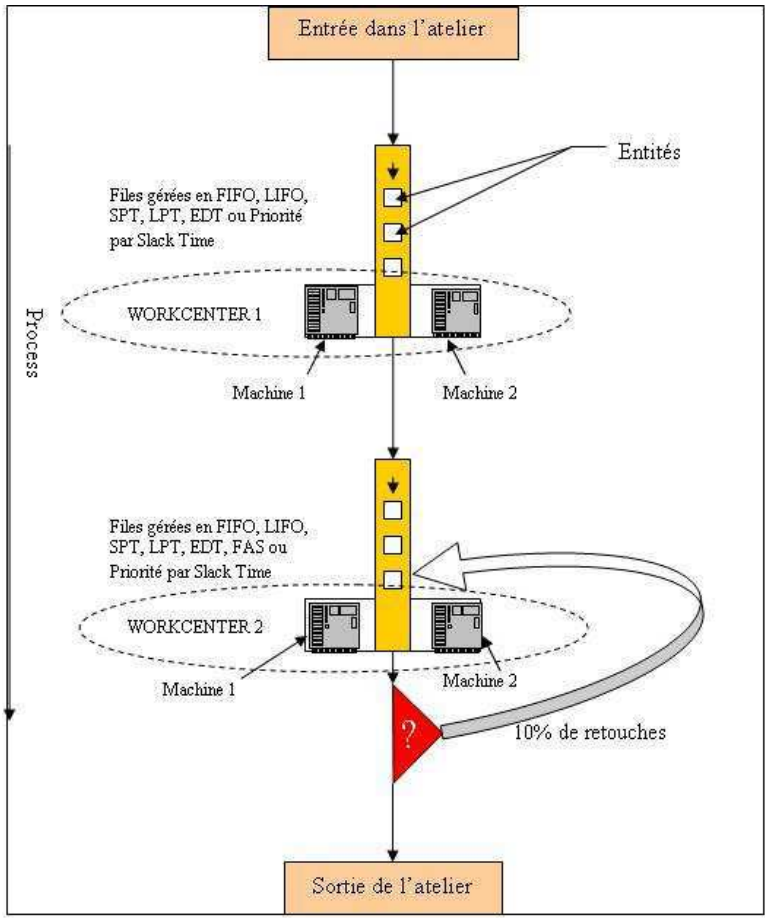


Figure 32. L'atelier flowshop. (Barett et Barman, 1986)

Dans ce système l'objectif est de minimiser le retard moyen (meantardiness) qui est la moyenne des retards obtenus pour les pièces usinées dans cet atelier, en choisissant des règles de priorité optimales pour les files d'attente.

Les actions de pilotage consistent ici à déterminer en temps réel quelles pièces doivent passer sur les centres d'usinage. Pour cela on décide de changer dynamiquement les règles de priorité déterminant l'ordre des pièces (Pierreval, 1992b).

Les règles de priorité utilisées dans cet exemple sont: {FIFO, EDD, SPT, LPT, LIFO} qui sont utilisées dans le premier centre de travail de l'atelier workcenter 1. Et {EDD, SPT, LPT, FAS} qui sont utilisées dans le deuxième centre de travail de l'atelier workcenter 2 avec :

FIFO : *First In First Out*
La pièce entrée en premier dans la file d'attente est traitée en premier. C'est la règle du « premier arrivé, premier servi ».

LIFO : *Last In First Out*
Cette règle est similaire à la précédente mais c'est la dernière pièce arrivée qui est traitée en premier.

SPT : *Shortest Processing Time*
Donne la priorité aux pièces ayant le plus faible temps de process : la pièce ayant le plus faible temps de process sera traitée en premier.

LPT : *Longest Processing Time*
La pièce ayant le plus long temps de process sera traitée en premier.

EDT : *Earliest Due Date* La pièce devant donc être livrée la première sera traitée en premier.

FAS **:** *First Arrived at Shop*

Cette règle permet de traiter en priorité les pièces entrées en premier dans l'atelier. Cette règle n'est donc pas judicieusement applicable à la première file d'attente car identique alors à la règle FIFO.

2.2. Expérimentations

Chaque simulation dure 15000 unités de temps, avec une période transitoire de 7000 unités de temps, ce qui correspond à une simulation de 4 années de travail de production dans l'atelier.

Les attributs des pièces sont définis grâce à des lois stochastiques. Les paramètres de l'atelier sont présentés ci-dessous.

Pour chaque simulation, il est possible de choisir la charge de l'atelier. L'utilisateur peut choisir entre une charge lourde (91%), moyenne (86%) et faible (81%). Il s'agit du **Shopload**.

Le temps de process des pièces peut également varier en changeant la dispersion des commandes. Deux options se présentent : une dispersion forte ou faible ce qui correspond donc à une **Déviation Standard** de 0,3 ou 0,6.

Les lois définissant les caractéristiques des pièces sont les suivantes :

- Loi d'arrivée des pièces dans l'atelier : exp(1)

- Temps de process = Exp(2. Shopload) + N (0, σ)

Avec *Shopload* correspond à la charge de l'atelier et *N(0, σ)* valeur sélectionnée par la distribution de loi normale de moyenne 0 et de déviation standard égal au choix de dispersion.

- Date de livraison $D_i = a_i + \alpha \sum p_{ij}$

Avec a_i correspond au temps d'arrivée dans l'atelier de la pièce

α correspond au facteur de retard (ici nous prendrons 3)

p_{ij} correspond au temps de process d'une entité i sur une machine j.

Le modèle de l'atelier flowshop

La modélisation de l'atelier avec le logiciel ARENA 10.0 est décrite ci-dessous.

La pièce après avoir été créée, est définie par des attributs (temps de process…) puis passe par le poste workcenter 1, puis par le poste workcenter 2. Enfin, ses caractéristiques sont analysées et le meantardiness du système est actualisé, avant d'évacuer la pièce de l'atelier.

Dès que les pièces arrivent dans le workcenter 1, elles sont placées dans une file d'attente avant d'être choisies par la règle de priorité. Une foi choisie, la pièce va être usinée (bloc Delay) puis avant d'être évacuer de la machine (par le bloc Release), elle va être dupliquée pour lancer une analyse de la première file d'attente. Après analyse, les nouvelles données sont mises à jours, et le bloc Event fait appel au réseau de neurones pour choisir la bonne règle de priorité correspondant à ces données d'entrée. La recherche de la prochaine pièce à usiner est alors effectuée et celle-ci est déplacée en première position dans la première file d'attente. Enfin, La pièce traitée est relâchée.

Le principe de fonctionnement de la modélisation de workcenter 2 reste la même que pour workcenter 1. La seule exception, est l'ajout d'un test en fin d'usinage pour savoir si des retouches sont applicables, si oui elles sont réalisées de suite.

Avant d'être évacuée de l'atelier, la pièce est analysée. Nous observons si elle est en retard ou non. Si tel est le cas le meantardiness est mis à jour. La pièce est par la suite supprimée de l'atelier.

2.2.1. Le module du réseau de neurones

Nous avons proposé deux manières différentes de prendre les décisions dans les deux files d'attente:

a) nous avons considéré un seul réseau de neurones qui prend les décisions pour les deux files d'attente. Dans cette première proposition nous indiquons à chaque fois dans le vecteur $\boldsymbol{X}$ des entrées du réseau de neurones le numéro de la file d'attente traitée (1 pour la première file d'attente ou 2 pour la deuxième file d'attente).

b) Nous avons considéré un réseau de neurones pour chaque file d'attente : c'est-à-dire nous considérons deux décideurs chacun choisit une règle de priorité pour la file d'attente qui le concerne.

2.2.1.1. Les entrées du réseau de neurones : un réseau de neurones pour les deux files d'attente

Tableau 4. Variables d'entrée du réseau de neurones communiquées par la simulation (proposition d'un réseau de neurones pour les deux files d'attente)

#	**Variables du vecteur d'entrée du réseau de neurones**
1	Nombre de pièces en attente dans la première file d'attente Q1.
2	Nombre de pièces en attente dans la deuxième file d'attente Q2.
3	Numéro de la file d'attente traitée.
4	Temps opératoire moyen sur le worckcenter 1 des pièces en attente dans la première file d'attente.
5	Temps opératoire minimum sur le worckcenter 1 des pièces en attente dans la première file d'attente.
6	Temps opératoire maximum sur le worckcenter 1 des pièces en attente dans la première file d'attente.
7	Temps opératoire moyen sur le worckcenter 2 des pièces en attente dans la première file d'attente.
8	Temps opératoire minimum sur le worckcenter 2 des pièces en attente dans la première file d'attente.
9	Temps opératoire maximum sur le worckcenter 2 des pièces en attente dans la première file d'attente.
10	Temps opératoire moyen sur le worckcenter 2 des pièces en attente dans la deuxième file d'attente.
11	Temps opératoire minimum sur le worckcenter 2 des pièces en attente dans la deuxième file d'attente.
12	Le temps opératoire maximum sur le worckcenter 2 des pièces en attente dans la deuxième file d'attente.
13	Pourcentage des pièces dans la première file d'attente ayant un temps opératoire sur le worckcenter 1 < à 1.5 unités de temps.
14	Pourcentage des pièces dans la première file d'attente ayant un temps opératoire sur le worckcenter 1 entre 1.5 et 3 unités de temps.
15	Pourcentage des pièces dans la première file d'attente ayant un temps opératoire sur le worckcenter 1 entre 3 et 4.5 unités de temps.

16	Pourcentage des pièces dans la première file d'attente ayant un temps opératoire sur le worckcenter 1 > à 4.5 unités de temps.
17	Pourcentage des pièces dans la première file d'attente ayant un temps opératoire sur le worckcenter 2 < à 1.5 unités de temps.
18	Pourcentage des pièces dans la première file d'attente ayant un temps opératoire sur le worckcenter 2 entre 1.5 et 3 unités de temps.
19	Pourcentage des pièces dans la première file d'attente ayant un temps opératoire sur le worckcenter 2 entre 3 et 4.5 unités de temps.
20	Pourcentage des pièces dans la première file d'attente ayant un temps opératoire sur le worckcenter 2 > à 4.5 unités de temps.
21	Pourcentage des pièces dans la deuxième file d'attente ayant un temps opératoire sur le worckcenter 2 < à 1.5 unités de temps.
22	Pourcentage des pièces dans la deuxième file d'attente ayant un temps opératoire sur le worckcenter 2 entre 1.5 et 3 unités de temps.
23	Pourcentage des pièces dans la deuxième file d'attente ayant un temps opératoire sur le worckcenter 2 entre 3 et 4.5 unités de temps.
24	Pourcentage des pièces dans la deuxième file d'attente ayant un temps opératoire sur le worckcenter 2 > à 4.5 unités de temps.
25	Pourcentage des pièces dans la première file d'attente ayant un slacktime positif.
26	Pourcentage des pièces dans la première file d'attente ayant un slacktime négatif.
27	Pourcentage des pièces dans la deuxième file d'attente ayant un slacktime positif.
28	Pourcentage des pièces dans la deuxième file d'attente ayant un slacktime négatif.

2.2.1.2. Les entrées du réseau de neurones : un réseau de neurones pour chaque file d'attente

Nous utilisons dans cette proposition deux réseaux de neurones, les tableaux :tab 1 et tab 2 , présentent respectivement les entrées du réseau de neurones de la première file d'attente et le réseau de neurones de la deuxième file d'attente.

Tableau 5. Variables d'entrées du premier réseau de neurones de la première file d'attente communiquées par la simulation.

#	**Variables du vecteur d'entrée du réseau de neurones**
1	Nombre de pièces en attente dans la première file d'attente Q1.
2	Nombre de pièces en attente dans la deuxième file d'attente Q2.
3	Temps opératoire moyen sur le worckcenter 1 des pièces en attente dans la première file d'attente.
4	Temps opératoire minimum sur le worckcenter 1 des pièces en attente dans la première file d'attente.
5	Temps opératoire maximum sur le worckcenter 1 des pièces en attente dans la première file d'attente.
6	Temps opératoire moyen sur le worckcenter 2 des pièces en attente dans la première file d'attente.
7	Temps opératoire minimum sur le worckcenter 2 des pièces en attente dans la première file d'attente.
8	Temps opératoire maximum sur le worckcenter 2 des pièces en attente dans la première file d'attente.
9	Temps opératoire moyen sur le worckcenter 2 des pièces en attente dans la deuxième file d'attente.
10	Temps opératoire minimum sur le worckcenter 2 des pièces en attente dans la deuxième file d'attente.
11	Le temps opératoire maximum sur le worckcenter 2 des pièces en attente dans la deuxième file d'attente.
12	Pourcentage des pièces dans la première file d'attente ayant un temps opératoire sur le worckcenter 1 < à 1.5 unités de temps.
13	Pourcentage des pièces dans la première file d'attente ayant un temps opératoire sur le worckcenter 1 entre 1.5 et 3 unités de temps.
14	Pourcentage des pièces dans la première file d'attente ayant un temps opératoire sur le worckcenter 1 entre 3 et 4.5 unités de temps.
15	Pourcentage des pièces dans la première file d'attente ayant un temps opératoire sur le worckcenter 1 > à 4.5 unités de temps.
16	Pourcentage des pièces dans la première file d'attente ayant un temps opératoire sur le worckcenter 2 < à 1.5 unités de temps.
17	Pourcentage des pièces dans la première file d'attente ayant un temps opératoire sur le worckcenter 2 entre 1.5 et 3 unités de temps.
18	Pourcentage des pièces dans la première file d'attente ayant un temps opératoire sur le worckcenter 2 entre 3 et 4.5 unités de temps.

19	Pourcentage des pièces dans la première file d'attente ayant un temps opératoire sur le worckcenter 2 > à 4.5 unités de temps.
20	Pourcentage des pièces dans la deuxième file d'attente ayant un temps opératoire sur le worckcenter 2 < à 1.5 unités de temps.
21	Pourcentage des pièces dans la deuxième file d'attente ayant un temps opératoire sur le worckcenter 2 entre 1.5 et 3 unités de temps.
22	Pourcentage des pièces dans la deuxième file d'attente ayant un temps opératoire sur le worckcenter 2 entre 1.5 et 3 unités de temps.
23	Pourcentage des pièces dans la deuxième file d'attente ayant un temps opératoire sur le worckcenter 2 entre 3 et 4.5 unités de temps.
24	Pourcentage des pièces dans la deuxième file d'attente ayant un temps opératoire sur le worckcenter 2 > à 4.5 unités de temps.
25	Pourcentage des pièces dans la première file d'attente ayant un slacktime positif.
26	Pourcentage des pièces dans la première file d'attente ayant un slacktime négatif.
27	Pourcentage des pièces dans la deuxième file d'attente ayant un slacktime positif.
28	Pourcentage des pièces dans la deuxième file d'attente ayant un slacktime négatif.

Tableau 6. Variables d'entrées du deuxième réseau de neurones de la deuxième file d'attente communiquées par la simulation.

#	**Variables du vecteur d'entrée du réseau de neurones**
1	Nombre de pièces en attente dans la deuxième file d'attente Q2.
2	Pourcentage des pièces dans la deuxième file d'attente ayant un temps opératoire sur le worckcenter 2 < à 1.5 unités de temps.
3	Pourcentage des pièces dans la deuxième file d'attente ayant un temps opératoire sur le worckcenter 2 entre 1.5 et 3 unités de temps.
4	Pourcentage des pièces dans la deuxième file d'attente ayant un temps opératoire sur le worckcenter 2 entre 3 et 4.5 unités de temps.
5	Pourcentage des pièces dans la deuxième file d'attente ayant un temps opératoire sur le worckcenter 2 > à 4.5 unités de temps.
9	Pourcentage des pièces dans la deuxième file d'attente ayant un slacktime positif.
10	Pourcentage des pièces dans la deuxième file d'attente ayant un slacktime négatif.
11	Pourcentage des pièces qui ont un slacktime < à -10
12	Pourcentage des pièces qui ont un slacktime entre -10 et -5
13	Slacktime minimum
14	Slacktime maximum
15	Temps opératoire moyen sur le worckcenter 2 des pièces en attente dans la deuxième file d'attente.

16	Temps opératoire minimum sur le worckcenter 2 des pièces en attente dans la deuxième file d'attente.
17	Le temps opératoire maximum sur le worckcenter 2 des pièces en attente dans la deuxième file d'attente.

2.2.1.3.Sorties du réseau de neurones

Les règles de priorités sont représentées par les numéros de 1 à 5 (ou 6 pour la file 2) pour chaque WC (Tableau 7), cela nous permet de faire la correspondance entre la valeur de sortie calculée et un numéro de règle de priorité.

Tableau 7. Les règles de priorités

Numéro	Règle de priorité	
	File 1	File 2
1	FIFO	FIFO
2	LIFO	LIFO
3	SPT	SPT
4	LPT	LPT
5	EDT	EDT
6		FAS

2.2.1.4. Architecture du réseau de neurones

Nous avons utilisé une architecture de trois couches pour toutes nos expérimentations. Et nous avons considéré différentes configurations pour la couche cachée ; nous avons varié le nombre de neurones dans cette couche, et ceci est pour les deux propositions que nous avons considéré (un réseau de neurones pour les deux files d'attente et un réseau de neurones pour chaque file d'attente).

Le nombre de poids que nous allons optimiser dépend du nombre de neurones que nous choisissons à mettre dans la couche cachée.

2.2.2. Le module d'optimisation

Comme nous l'avons expliqué dans le chapitre précédent, nous utilisons pour notre approche la méthode du recuit simulé pour optimiser le vecteur de poids ***W*** du réseau de neurones.

Nous rappelons ci-dessous le voisinage variable que nous utilisons :

La solution voisine W_l' est définie en deux étapes tel que :

Etape 1 : sélectionner aléatoirement ***g(l)*** éléments *de* W_l

Etape 2 : pour chaque élément sélectionné, nous ajoutant un nombre aléatoire à partir d'une loi normale **N(0, *s*(T_l))**.

Dans l'algorithme que nous utilisons :

- $$g(l) = E\left[1 + \left((a-1) * e^{-l^2/T_0 * 10}\right)\right] \quad \textbf{(21)}$$

- $$s(T_l) = \left[(\sigma_{\max} - \sigma_{\min})/(T_0 - \varepsilon)\right] * (T_l - T_0) + \sigma_{\max} \quad \textbf{(22)}$$

Avec:

s est une fonction qui prend ses valeurs dans l'intervalle [σ_{min} , σ_{max}],

E[z] est la partie entière du nombre réel z,

a est le nombre total des caractéristiques du réseau de neurones que nous voulons optimiser,

T_0 est la température initiale du recuit simulé,

T_l est la température courante,

ε est la condition d'arrêt de l'algorithme qui est initialement choisie.

Les fonctions ***g(l)*** et ***s(T_l)*** diminuent progressivement, tout au long des itérations du recuit simulé, avec la diminution de la température. Ainsi, les changements effectués au début de l'optimisation sont beaucoup plus importants que les changements effectués à la fin.

Plusieurs paramètres dans l'algorithme du recuit simulé que nous utilisons sont définis par l'utilisateur tel que la température initiale T_0, la condition d'arrêt **ε**, etc. Le tableau ci-dessous (Tableau 8) reprend toutes les variables que nous devons initialiser pour chacune de nos expérimentations. Dans la section qui suit (Résultats des expérimentations) nous présenterons à chaque expérimentation les valeurs de ces variables.

Tableau 8. Les paramètres du recuit simulé à initialiser par l'utilisateur

Paramètres du recuit simulé à initialiser par l'utilisateur	**Value**
Température initiale	T_0
Taux de refroidissement	***r***
La borne supérieure de l'intervalle [σ_{min} , σ_{max}] où la fonction ***s*** prend ses valeurs.	σ_{max}
Nombre d'itérations sans baisse de la température pour chaque palier de température traité.	**k**
La borne inférieure de l'intervalle [σ_{min} , σ_{max}] où la fonction ***s*** prend ses valeurs.	σ_{min}
La condition d'arrêt	**ε**

2.3. Résultats des expérimentations

D'abord, beaucoup d'expérimentations ont été menées afin de déterminer une bonne topologie pour le réseau de neurones (nombre de neurones dans la couche cachée) et les bons paramètres pour l'algorithme de recuit simulé. L'optimisation commence à partir d'un vecteur initial de poids obtenus aléatoirement. Pendant l'optimisation de ces poids avec différents nombres de neurones

dans la couche cachée, nous avons considéré seulement 10 réplications pour chaque run de simulation, pour économiser le temps de calcul.

À la fin du processus d'optimisation, une fois les poids sont optimisés, nous avons utilisé 50 réplications pour comparer les décisions des réseaux de neurones expérimentés avec les meilleures combinaisons de règles de priorité trouvées par Barrett et le Barman (1986).

2.3.1. Les meilleures combinaisons de règles de priorité trouvées par Barrett et le Barman (1986)

Nombre de répliques : 50
Le choix arbitraire du coefficient K :2
Shopload :91%

Tableau 9. Retards moyens obtenus à partir des meilleures combinaisons de règles de priorité proposées par Barrett et Barmann

Combinaison des règles	Le retard moyen du modèle
FIFO-FIFO	17.76443
EDD-EDD	11.99342
EDD-SPT	7.27360
SPT-EDD	6.67160
SPT-FIFO	9.33060
SPT-SPT	3.69815
EDD-FIFO	14.00025

En se basant sur les résultats trouvés par Barrett et Barman nous constatons que les deux meilleures combinaisons des règles de priorité pour minimiser le retard moyen sont *SPT-EDD* et *SPT-SPT* qui ont pu réduire le retard moyen de l'atelier respectivement à 6.67160 unités de temps et 3.69815 unités de temps.

Dans la section suivante nous comparons les résultats trouvés par les réseaux de neurones avec les meilleurs résultats trouvés par Barrett et Barman

2.3.2. Les résultats obtenus par le réseau de neurones

Dans un premier temps, nous avons considéré un seul réseau de neurones pour les deux files d'attente, et à chaque fois que notre modèle de simulation fait appel à la décision du réseau de neurones, le numéro de la file concernée est communiqué au réseau.

Dans ce qui suit, nous présentons les différents résultats que nous avons obtenus en variant le nombre de neurones de la couche cachée et les différents paramètres du Recuit Simulé.

Compte tenu des objectifs de ces essais rappelés au début de ce chapitre, la performance du réseau de neurones est évaluée en regard de la minimisation du retard moyen dans notre atelier flowshop.

La contribution de l'approche que nous proposons se mesure essentiellement au regard de la capacité du réseau de neurones à apprendre à prendre les bonnes décisions. Par exemple, dans le tableau 10 nous présentons les résultats obtenus des décisions prises par un réseau de neurones avec trois neurones cachés. En effet, en analysant les résultats, on s'aperçoit que le réseau de neurones a obtenu de très mauvais résultats pendant les premières itérations du recuit simulé, tels qu'ils sont attendus puisque le réseau de neurones n'a pas encore appri.

Dans le tableau 10 nous présentons les résultats obtenus par un réseau de neurones avec trois neurones cachés (le nombre de neurones dans la couche cachée = 3).

Tableau 10. Les résultats des différentes expérimentations de l'apprentissage du réseau de neurones avec nombre de neurones dans la couche cachée est 3.

Les paramètres du Recuit Simulé								**Retard moyen obtenu par le réseau de neurones**	
T_0	r	σ_{max}	σ_{min}	ε	K	**Nombre d'itérations**	**Temps CPU (minutes)**	**Avant apprentissage**	**Après apprentissage**
2500	**0.80**	**5**	**1**	**0.002**	**3**	**125**	**5875**	**17.879820**	**6.185470**
3000	**0.75**	**10**	**2**	**0.001**	**3**	**155**	**7285**	**27.564432**	**4.335690**
4500	**0.85**	**15**	**3**	**0.001**	**2**	**188**	**8836**	**23.944312**	**5.677310**

Ensuite, tout au long de l'optimisation de ses poids par le recuit simulé, le réseau de neurones permet de diminuer le retard moyen dans l'atelier. Nous nous apercevons qu'après optimisation le réseau de neurones arrive à combiner les six règles de priorité dans le but de minimiser le

retard moyen dans l'atelier. Alors, il est devenu capable de choisir les combinaisons des règles sans se baser ni sur une expertise préalable, ni sur un ensemble d'apprentissage.

Les résultats obtenus montrent une nette amélioration des retards calculés à la base des décisions du réseau de neurones au début de son apprentissage par rapport aux retards calculés à la base des décisions du réseau de neurones après son apprentissage. Cette constatation est vérifiée pour toutes les configurations des réseaux de neurones que nous avons utilisés (3 neurones cachés tableau 10, 5 neurones cachés tableau 11, 7 neurones cachés tableau 12 et 12 neurones cachés tableau 13). Ceci s'explique par le fait que pour chaque réseau de neurones nous partons d'une matrice de poids aléatoires.

Dans les tableaux de résultats que nous présentons pour ce premier exemple (tableau 10, tableau 11, tableau 12 et tableau 13) chaque ligne représente un cycle (ou run) d'apprentissage du réseau de neurones, au début de chaque cycle d'apprentissage le réseau de neurones ne réalise pas de bonne performance puisque comme nous l'avons expliqué précédemment il n'a pas encore appris. Cependant une fois qu'il a apprit, nous obtenons des performances très intéressantes.

Dans le tableau ci-dessous (Tableau 11) nous présentons les résultats obtenus par un réseau de neurones avec cinq neurones cachés (le nombre de neurones dans la couche cachée = 5).

Tableau 11. Les résultats des différentes expérimentations de l'apprentissage du réseau de neurones avec nombre de neurones dans la couche cachée est 5.

Les paramètres du Recuit Simulé								**Retard moyen obtenu par le réseau de neurones**	
T_0	r	σ_{max}	σ_{min}	ε	k	**Nombre d'itérations**	**Temps CPU (minutes)**	**Avant apprentissage**	**Après apprentissage**
2500	**0.80**	**5**	**1**	**0.002**	**2**	***125***	**5875**	**19.532210**	**5.177560**
4500	**0.90**	**25**	**5**	**0.0005**	**3**	**455**	**21385**	**21.437230**	**3.61277**
7000	**0.90**	**20**	**5**	**0.0001**	**3**	**514**	**24158**	**25.430000**	**3.44712**
3000	**0.85**	**20**	**5**	**0.001**	**3**	**275**	**12925**	**32.565500**	**4.855220**

Dans le tableau ci-dessous (Tableau 12) nous présentons les résultats obtenus par un réseau de neurones avec sept neurones cachés (le nombre de neurones dans la couche cachée=7).

Tableau 12. Les résultats des différentes expérimentations de l'apprentissage du réseau de neurones avec nombre de neurones dans la couche cachée est 7.

Les paramètres du Recuit Simulé								**Retard moyen obtenu par le réseau de neurones**	
T_0	r	σ_{max}	σ_{min}	ε	k	**Nombre d'itérations**	**Temps CPU (minutes)**	**Avant apprentissage**	**Après apprentissage**
4000	**0.90**	**20**	**5**	**0.0002**	**2**	**319**	**14993**	**26.667120**	**5.244570**
2000	**0.90**	**15**	**3**	**0.008**	**2**	**235**	**11045**	**31.366120**	**4.484630**
4000	**0.85**	**18**	**1**	**0.001**	**2**	**187**	**8695**	**29.787250**	**4.328930**

Dans le tableau ci-dessous (Tableau 13) nous présentons le résultat obtenu par un réseau de neurones avec douze neurones cachés (le nombre de neurones dans la couche cachée = 12).

Tableau 13. Le résultat de l'expérimentation de l'apprentissage du réseau de neurones avec nombre de neurones dans la couche cachée est 12.

Les paramètres du Recuit Simulé								**Retard moyen obtenu par le réseau de neurones**	
T_0	r	σ_{max}	σ_{min}	ε	k	**Nombre d'itérations**	**Temps CPU (minutes)**	**Avant apprentissage**	**Après apprentissage**
5000	**0.90**	**30**	**5**	**0.0005**	**3**	**453**	**21291**	**34.655270**	**7.003410**

Par ailleurs, nous constatons que le choix des paramètres du recuit simulé a un impact considérable sur la qualité d'apprentissage des réseaux de neurones. Nous pouvons déduire d'après les configurations que nous avons réalisées et qui sont présentées dans les quatre tableaux de résultats que le nombre d'itérations dans chaque cycle (run) est étroitement lié à l'amélioration

des décisions prises par les réseaux de neurones (minimisent le retard moyen), en effet, plus le nombre d'itérations est élevé, plus l'apprentissage des réseaux de neurones est amélioré, mais ceci tant que le recuit simulé n'a pas était bloqué dans un minimum local.

En ce qui concerne l'impact de la température initiale $\mathbf{T_0}$ sur les performances des réseaux de neurones nous pouvons constater à travers nos expérimentations empiriques que nous pouvons réaliser un bon apprentissage avec des températures élevées, comme nous pouvons avoir aussi un bon apprentissage avec des températures qui ne sont pas trop élevées, par exemple (tableau 10) : le réseau de neurones à 3 neurones cachés a pu atteindre dans le deuxième cycle d'apprentissage que nous présentons (la deuxième ligne du tableau) un retard moyen de **4.335690** à partir d'une température initiale $\mathbf{T_0 = 3000}$, comme il a atteint dans le troisième cycle d'apprentissage que nous présentons (la troisième ligne du tableau) un retard moyen de **5.677310** à partir d'une température initiale plus élevée $\mathbf{T_0 = 4500}$. En effet, en se basant sur le principe du recuit simulé, nous rappelons que à température nulle, le système évolue à partir d'une configuration vers la, ou les, configurations d'énergie les plus basses par une méthode de type gradient. Par conséquent, dans le cas d'un système frustré, la configuration atteinte est le plus souvent un état métastable, dont l'énergie est supérieure à celle du minimum absolu. Le système est en quelque sorte piégé dans ce minimum local (Figure. 33), suivant la configuration initiale, indiquée par les flèches, la dynamique aboutit à température nulle dans l'un quelconque des minima relatifs, séparés par les barrières indiquées en pointillés. A température élevée, les processus probabilistes permettent au système de sauter les barrières séparant les vallées.

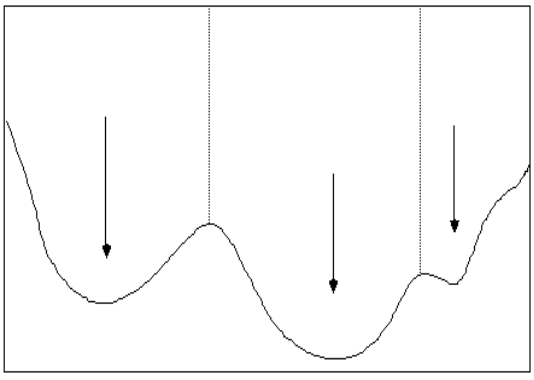


Figure 33. Un paysage d'énergie basé sur le recuit simulé.

En revanche, à température non nulle, le caractère probabiliste des changements de configuration peut permettre au système de remonter la pente et de ressortir du bassin d'attraction d'un minimum relatif pour accéder à un autre bassin plus profond. Idéalement, pour se rapprocher du

minimum, il faudrait que la température soit ***assez grande*** pour permettre au système de sauter les barrières et qu'elle soit ***assez faible*** pour qu'il soit malgré tout attiré vers le fond des vallées.

Le plus intéressant est alors d'envisager une double dynamique: celle de la recherche des minima à température fixée, plus une dynamique de diminution de la température. Si l'on part d'une température élevée, toutes les configurations sont accessibles et le système n'a qu'une faible préférence pour les états de faible énergie. En diminuant progressivement la température par un ratio ***r*** de la baisse de température pas trop élevé, on permet au système de rechercher des bassins d'attraction relativement larges au début, tout en lui évitant d'être piégé par les attracteurs métastables. Ce qui explique l'obtention d'un retard moyen plus performant (la deuxième ligne du tableau : un retard moyen de **4.335690** à partir d'une température initiale $\mathbf{T_0}$ = **3000)** en partant d'une température moins élevée que celui d'un retard moyen d'un apprentissage partant d'une température plus élevée, puisque le premier est basé sur un ratio de baisse de température (**r =0.75**) moins élevé que le second (**r =0.85**).

De plus nous avons obtenu de meilleurs résultats que ceux obtenus par Barrett et Barman (Barrett et Barman, 1986). En effet, en partant d'une matrice aléatoire de poids d'un réseau de neurones à cinq neurones cachés, le retard moyen dans l'atelier a diminué de **21.437230** unités de temps à **3.61277** unités de temps en **455** itérations du recuit simulé. De plus pour la même configuration du réseau de neurones (5 neurones cachés) le retard moyen dans l'atelier a diminué de **25.43** unités de temps à **3.44712** unités de temps en **514** itérations du recuit simulé.

Par ailleurs, si nous considérons les temps de nos calculs (CPU) nous constatons que typiquement, un cycle d'apprentissage de 155 itérations (run d'optimisation via simulation) qui utilise le modèle de simulation de notre atelier flowshop de 10 réplications et une période de simulation de 4 ans, exige 7285 minutes sur un PC avec Intel Pentium® 4, CPU 3,80 GHz et 2 Go de mémoire RAM. C'est-à-dire, 47 minutes pour chaque itération du recuit simulé. Certes, c'est temps important, mais c'est le temps d'apprentissage d'un réseau de neurones. En effet, la phase d'apprentissage nécessite suffisamment de temps pour que le réseau de neurones puisse finir son cycle d'apprentissage et atteint la performance qui lui permettra de prendre les bonnes décisions. L'élément essentiel des réseaux de neurones est qu'ils peuvent capter les dépendances non-linéaires de haut niveau entre les variables d'entrée, ce qui est possible grâce à la présence d'une transformation, elle-même non-linéaire, dans le calcul de la valeur prédite. Cependant, ils ne peuvent acquérir cette dépendance non-linéaire que s'ils ont suffisamment appris, ce qui explique en partie le temps d'apprentissage élevé qu'ils nécessitent.

D'autre part, le but de notre choix d'un atelier déjà existant dans la littérature pour pouvoir comparer et évaluer les performances des réseaux de neurones, nous a obligé de simuler sur une longue durée de quatre années (conformément aux caractéristiques de l'atelier flowshop de Barett et Barmann), ce qui explique en partie le temps d'apprentissage des réseaux de neurones, puisque à chaque itération de recuit simulé dans le cycle d'apprentissage on évalue le retard moyen généré par les nouvelles décisions prises par les réseaux de neurones après optimisation de leurs poids.

Néanmoins, une fois appris, le réseau de neurones prend les bonnes décisions qui optimisent la fonction objectif dans l'atelier en fraction de seconde. En effet, le calcul lui-même de la décision du réseau de neurones ne nécessite pas un temps de calcul (CPU) important mais c'est le temps d'apprentissage qui inclut à chaque fois l'évaluation via le modèle de simulation qui nécessite un temps très important. Dès que nous intégrant le réseau de neurones (sans le module d'apprentissage par optimisation via simulation) dans l'atelier réel pour qu'il prenne ses décisions en temps réel, nous ne parlons plus alors d'un temps de calcul pour la prise de décisions puisqu'il est négligeable.

Dans le tableau qui suit nous présentons les performances de deux réseaux de neurones différents dans notre modèle le premier servira à déterminer le choix de règles de priorité dans la première file d'attente. Alors que le deuxième réseau de neurones servira à déterminer le choix de règles de priorité dans la deuxième file d'attente.

Nombre de neurones dans la couche cachée du premier réseau de neurones = 4

Nombre de neurones dans la couche cachée du deuxième réseau de neurones = 3

Tableau 14. Les résultats des différentes expérimentations de l'apprentissage du réseau de neurones avec nombre de neurones dans la couche cachée est 3.

Les paramètres du Recuit Simulé								Retard moyen obtenu par le réseau de neurones	
T_0	r	σ_{max}	σ_{min}	ε	K	Nombre d'itérations	Temps CPU (minutes)	Avant apprentissage	Après apprentissage
5000	**0.90**	**20**	**3**	**0.002**	**3**	**419**	**19693**	**23.789060**	**3.788980**
3000	**0.80**	**30**	**5**	**0.006**	**2**	**117**	**5499**	**46.694599**	**4.006790**

Les résultats que nous avons obtenu en considérant un réseau de neurones pour chaque file d'attente sont assez performants ; certes ils ne dépassent les résultats que nous avons déjà présentés (un réseau de neurones pour les deux files d'attentes) mais ils sont trop près, pour les deux expérimentations que nous avons retenus et présentés dans le tableau ci-dessus, les réseaux de neurones ont minimisé le retard moyen à **3.788980** et **4.006790**. En effet, nous avons appris à chaque réseau de neurones le comportement de la file d'attente pour laquelle il prend ses décisions et choisit les règles de priorité adéquates, alors nous avons limité le champ de connaissance de chaque réseau de neurones dans le but qu'il garde sa concentration d'apprentissage sur un seul comportement (d'une seule file d'attente), c'est pour cela que nous avons considéré pour chaque réseau de neurones les variables qui sont liées à la file d'attente qui lui correspond (voir les tableaux des variables de décisions dans la section précédente).

3. Exemple 2 : Application de l'approche d'apprentissage autonome des réseaux de neurones pour le pilotage d'un atelier Jobshop

3.1. description de l'exemple

Nous considérons un atelier job shop inspiré du travail de Fonseca et Navaresse (Fonseca et al. 2002) (Mouelhi et al. 2008), fonctionnant de manière stochastique. Cet atelier est constitué de

quatre machines différentes {M1, M2, M3, M4}, et nous avons quatre types de pièces différents {P1, P2, P3, P4}.

Les pièces suivent les cheminements suivants en fonction de leur type :

Tableau 15. Séquences des pièces en fonction de leur type

Type de pièces	Séquences
P1	M1 – M3 – M2
P2	M2- M3 - M1 – M4
P3	M2- M3 - M4
P4	M4- M2- M1

La loi d'arrivée des pièces est du type exponentiel de moyenne 5,5 minutes. Les taux de production des pièces et les lois régissant les temps opératoires par type de pièces et par machine en minutes suivant une loi normale [moyenne, écart type] sont présentés dans le tableau suivant :

Tableau 16. Paramètres des lois des durées opératoires

Type de pièces	Proportion d'arrivée par type (%)	M1	M2	M3	M4
P1	40	[9, 1.3]	[7, 1]	[8, 1.2]	-
P2	15	[7, 1]	[8, 1.2]	[9, 1.3]	[7, 1]
P3	20	-	[9, 1.3]	[7, 1]	[9, 1.5]
P4	25	[9, 1.5]	[4, 1]	-	[8, 1]

Chaque machine dans l'atelier possède sa propre file d'attente. Le problème que nous considérons dans cet atelier est de minimiser le retard moyen. Nous recherchons pour cela des heuristiques capables de déterminer la séquence de passage des pièces sur chaque machine, en considérant l'état réel du système à chaque instant *t* de prise de décision.

Le Passage des pièces dans les files d'attente est assuré par six règles de priorité. Elles ne présentent pas toutes le même intérêt mais elles permettent de réaliser des heuristiques de gestion de production variée.

Les règles que nous avons utilisées pour notre atelier sont :

- *FIFO* (First In First Out), *LIFO* (Last In Last Out),
- *SPT* (Shortest Processing Time), *LPT* (Longest Process Time),
- *EDD* (Earliest Due Date),
- *PPM* : Plus Petite Marge ; la marge correspond à la différence entre la date de livraison et la somme des temps opératoires à effectuer.

Notre modèle de représentation de l'atelier est présenté ci-dessous :

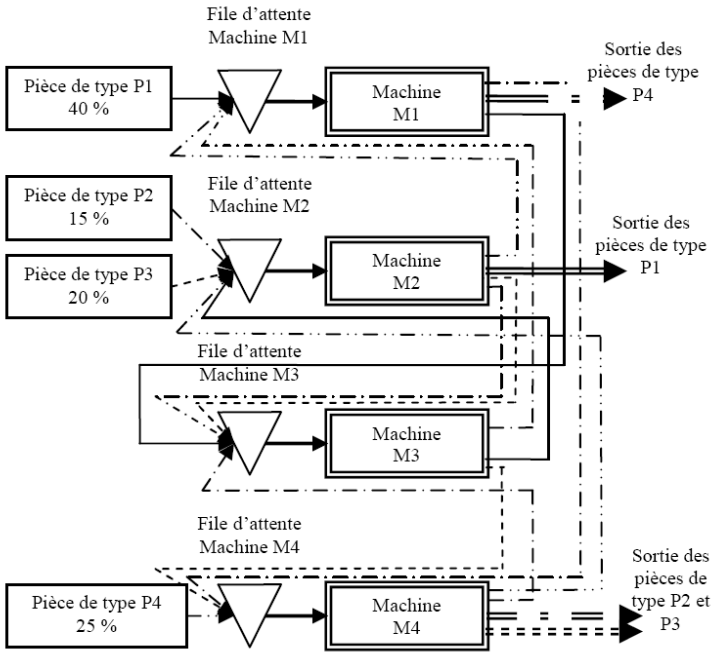


Figure 34. Représentation de l'atelier à cheminements multiples étudié et des différents flux

3.2. Expérimentations

3.2.1. Le module de simulation

Pour modéliser l'atelier nous avons utilisé le logiciel de simulation ARENA 10.0, l'unité de temps de la simulation est la minute et notre atelier fonctionne 24 heures sur 24 et 7 jours sur 7, sans pause même pendant les changements d'équipe. La durée de la simulation est de 40320 minutes soit 1 mois comprenant une période de transition de 1 semaine durant laquelle les statistiques de l'atelier ne sont pas comptabilisées.

Dans notre Job shop, les pannes et les arrêts de maintenance ne sont pas pris en compte et les temps de transfert entre les différentes parties de l'atelier sont considérés comme nuls. On considère qu'il n'y a jamais de rupture d'approvisionnement.

Pour résoudre ce problème, nous proposons la construction d'heuristiques basées sur des règles de priorité à partir d'une logique déterminée par un réseau de neurones. Cette logique sera au départ aléatoire, puis améliorée par optimisation via simulation, afin de devenir progressivement performante.

3.2.2. Le module du réseau de neurones

Les décisions étant basées sur l'état courant de l'atelier, nous définissons le vecteur d'entrée X qui représente les variables d'état de celui-ci. Nous avons étudié les paramètres et les variables de l'atelier qui reflètent au mieux son état courant et qui influent sur les différentes règles de priorité qui composeront nos heuristiques. Les différentes variables de décision que nous avons retenues sont :

1	Pourcentage des pièces de type P1 dans la file concernée.
2	Pourcentage des pièces de type P2 dans la file concernée.
3	Pourcentage des pièces de type P3 dans la file concernée.
4	Pourcentage des pièces de type P4 dans la file concernée.
5	Nombre de pièces dans toutes les autres files.
6	Temps opératoire moyen.
7	Temps opératoire maximum.
8	Temps opératoire minimum.
9	Numéro de la file traitée.
10	Date de livraison minimum.
11	Date de livraison maximum.
12	Date de livraison moyenne.
13	Pourcentage de pièces dans la file qui ont une marge inférieure à 10.

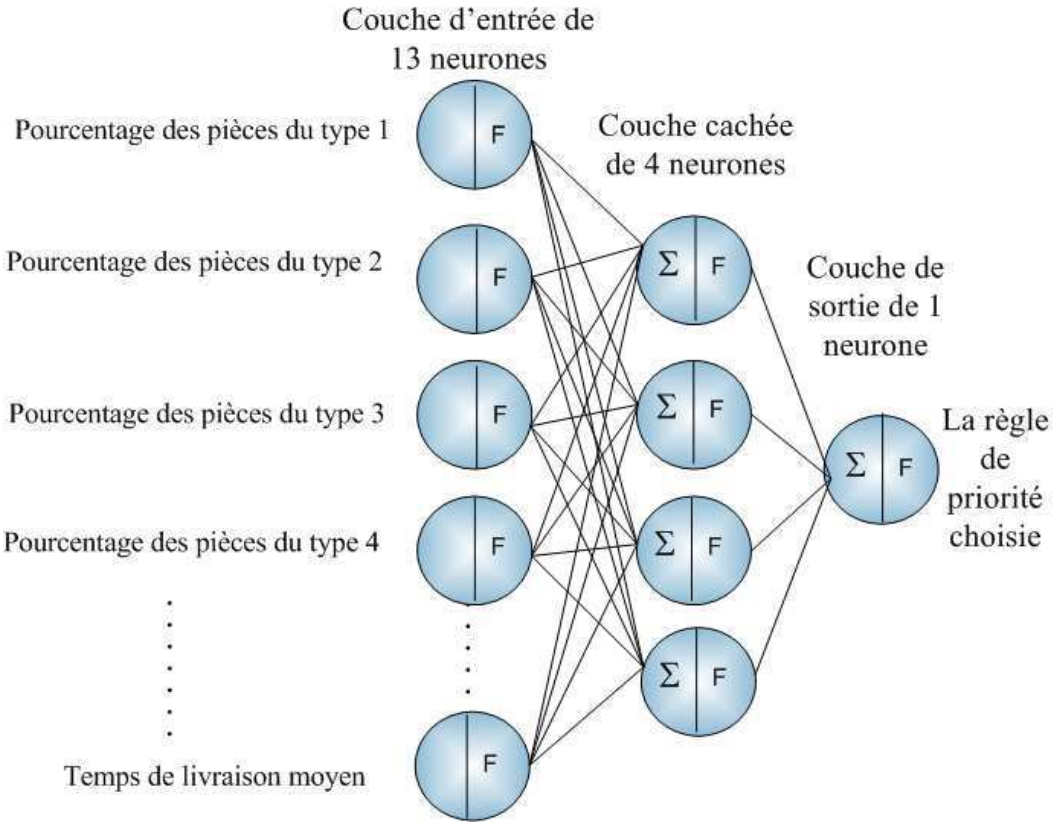


Figure 35. Le réseau de neurones utilisé pour le choix des règles de priorité

Dans les expérimentations que nous avons réalisées, le réseau de neurones est constitué de trois couches : une couche d'entrée de 13 neurones qui représentent le vecteur d'entrée **X** de l'état de l'atelier lors de la prise de décision et qui sont communiqués par la simulation, une couche caché de 4 neurones et une couche de sortie d'un seul neurone qui représente la règle de priorité choisie (figure 35).

Ces couches sont reliées entre elles par 56 connexions qui constituent notre matrice *W* de 56 poids que nous cherchons à optimiser. Nous utiliserons une fonction tangentielle comme fonction d'activation. Le module réseau de neurone et le module d'optimisation ont été développés avec Microsoft Visual C++ et connectés directement avec le modèle de simulation du job shop.

3.2.3. Le module d'optimisation

Dans ces expérimentations, nous avons utilisé un recuit simulé comme méthode d'optimisation de la matrice *W*, exactement comme dans l'exemple précédent de l'ateleier flowshop.

Une solution initiale W_0 est tirée aléatoirement. La perturbation des solutions est basée sur des principes de voisinage variables. Les principes du voisinage du recuit simulé est le même que celui du voisinage du recuit simulé utilisé pour l'optimisation via simulation de l'exemple précédent (voir l'exemple du flowshop).

L'approche choisie assure la connexité de l'espace de recherche. La fonction objectif, le retard moyen, est évalué par la simulation. Après plusieurs expérimentations, le paramètre de la diminution de température r et la température initiale ont été empiriquement déterminés. Le recuit simulé s'arrête quand la température est inférieure à ε, avec ε initialement choisie.

3.3. Les résultats des expérimentations

Nous présentons dans cette section les résultats que nous avons obtenus après les expérimentations menées pour l'apprentissage des réseaux de neurones à prendre les bonnes décisions dans l'atelier jobshop.

D'abord et pour pouvoir comparer les décisions du réseau de neurones aux performances des meilleures règles considérées dans l'atelier (*FIFO*, *LIFO*, *SPT*, *LPT*, *EDD et PPM*), nous avons considéré l'impact de chacune de ces règles sur le retard moyen. Le tableau 3 présente le retard moyen des six règles de priorité associées à chacune des files d'attente des quatre machines de l'atelier.

Tableau 17. Les résultats des six règles de priorité sur les quatre machines de l'atelier jobshop

Règle de priorité	**Le retard moyen**
SPT	298.305
PPM	156.496
EDD	157.813
FIFO	182.128
LPT	581.188
FIFO	412.569

En se basant sur les résultats ci-dessus (tableau 17), nous constatons que les règles PPM et EDD sont les règles les plus performantes pour minimiser le retard moyen dans notre atelier jobshop.

Ensuite, en analysant les résultats que nous avons obtenus nous constatons qu'ils sont similaires à ceux obtenus dans le premier exemple (l'atelier flowshop). En effet, au cours des premières itérations les retards moyens obtenus par les différents réseaux de neurones sont très élevés, tels qu'ils sont attendus puisque les réseaux de neurones n'ont pas encore appris. Puis, tout au long de l'optimisation de ses poids par le recuit simulé, chaque réseau de neurones que nous avons considéré a permis la diminution du retard moyen dans l'atelier. Nous nous apercevons qu'après optimisation le réseau de neurones arrive à combiner les six règles de priorité et proposer une heuristique dans le but de minimiser le retard moyen dans l'atelier. Alors, il est devenu capable de choisir les règles de priorité sans se baser ni sur une expertise préalable, ni sur un ensemble d'apprentissage.

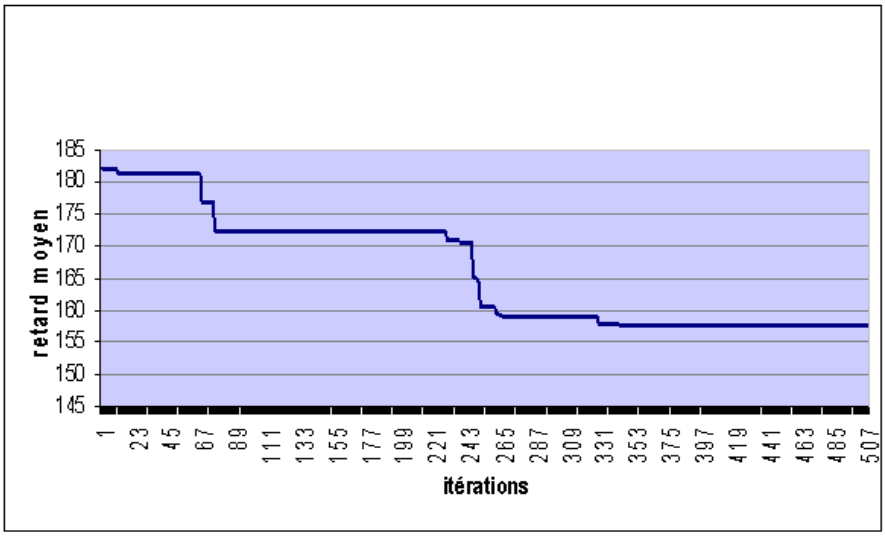


Figure 36. Evolution du retard moyen

La Figure 36 présente la diminution du retard moyen durant les différentes itérations du recuit simulé pour un exemple de cycle d'apprentissage à 507 itérations et une température initiale $\mathbf{T_0}$= **50000** (voir la ligne 3 du tableau 18) ; d'abord, nous avons calculé le retard moyen avec une matrice initiale de poids tirés aléatoirement : le retard moyen initial est égal à **182.669850**. Les résultats obtenus (dans les premières itérations) sont très mauvais. Ensuite, tout au long de l'optimisation de ses poids par le recuit simulé, le réseau de neurones permet de diminue le retard moyen dans l'atelier : il a atteint un retard moyen de **152.664552**. De plus, nous constatons que

cette dernière expérimentation (voir la ligne 3 du tableau 18) a permis l'apprentissage d'un réseau de neurones qui a été capable de dépasser les performances trouvées dans le tableau 17 des règles statiques que nous avons présenté au début de cette section. Ainsi, le réseau de neurones a minimisé le retard moyen de **182.669850** à **152.664552** unités de temps, et a dépassé les meilleures règles testées PPM et EDD qui n'ont minimisé le retard moyen respectivement qu'à **156.496** et **157.813**. L'apprentissage du réseau de neurones lui a permis d'acquérir la connaissance nécessaire à coupler les bonnes règles de priorité aux bons moments du cycle de pilotage, au regard de minimiser le retard moyen.

D'ailleurs, les performances des réseaux de neurones se sont améliorées lorsque nous avons considéré un réseau de neurones pour chaque file d'attente. En effet, nous avons considéré un réseau de neurones différent pour chaque file d'attente, de telle sorte que nous apprenons à chaque réseau de neurones le comportement de chaque file d'attente à part, c'est-à-dire nous avons limité la connaissance qu'il doit acquérir à une seule file d'attente, ceci a permis un apprentissage plus efficace qi nous comparons les retards moyens ainsi obtenus (tableau 19): **159.208801**, **148.669495** et **148.424927.**

Par ailleurs, si nous considérons les temps de nos calculs (CPU) nous constatons que typiquement, un cycle d'apprentissage de 507 itérations (run d'optimisation via simulation) qui utilise le modèle de simulation de notre atelier jobshop de 30 réplications et une période de simulation de 1 mois, exige 4562 secondes pour sur un PC avec Intel Pentium® 4, CPU 3,80 GHz et 2 Go de mémoire RAM. C'est-à-dire, environ 9 secondes pour chaque itération du recuit simulé. Ces temps CPU sont largement inférieurs à ceux constatés pour l'atelier flowshop. En effet, comme nous l'avons expliqué dans le premier exemple (flowshop) le temps de calculs CPU dépend étroitement du temps que la simulation consacre pour évaluer dans chaque itération les performances du réseau de neurones ou des réseaux de neurones (en cas d'un réseau de neurones pour chaque file d'attente). Dans l'atelier jobshop que nous avons choisi, la période simulée correspond à un mois ce qui explique une différence très importante entre le temps de calcul dans l'atelier flowshop et le temps de calcul dans l'atelier jobshop.

Résultats de la proposition d'un réseau de neurones pour les quatre files d'attente

Tableau 18. Les résultats des différentes expérimentations de l'apprentissage du réseau de neurones

Les paramètres du Recuit Simulé								**Retard moyen obtenu par le réseau de neurones**	
T_0	r	σ_{max}	σ_{min}	**ε**	**k**	**Nombre d'itérations**	**Temps CPU (seconde)**	**Avant apprentissage**	**Après apprentissage**
700000	**0.93**	**25**	**5**	**0.00001**	**3**	**1032**	**9288**	**201.093552**	**175.021851**
500000	**0.9**	**30**	**3**	**0.00006**	**4**	**867**	**7803**	**213.677813**	**159.651978**
50000	**0.9**	**20**	**5**	**0.001**	**3**	**507**	**4562**	**182.669850**	**152.664552**
800000	**0.92**	**40**	**10**	**0.00001**	**4**	**1208**	**10872**	**223.445790**	**156.273193**

Résultats de la proposition d'un réseau de neurones pour chaque file d'attente

Dans le tableau qui suit nous présentons les performances de quatre réseaux de neurones différents dans notre modèle chacun servira à déterminer le choix de règles de priorité dans la file d'attente concernée.

Tableau 19. Les résultats des différentes expérimentations de l'apprentissage du réseau de neurones

Les paramètres du Recuit Simulé								**Retard moyen obtenu par le réseau de neurones**	
T_0	r	σ_{max}	σ_{min}	**ε**	**k**	**Nombre d'itérations**	**Temps CPU (seconde)**	**Avant apprentissage**	**Après apprentissage**
10000	**0.9**	**10**	**1**	**0.001**	**3**	**459**	**4131**	**250.920029**	**159.208801**
900000	**0.95**	**15**	**5**	**0.00001**	**3**	**1466**	**13194**	**265.545570**	**148.669495**
500000	**0.91**	**20**	**3**	**0.00006**	**3**	**726**	**6534**	**234.800934**	**148.424927**

4. Conclusion

A travers ce chapitre, nous avons illustré l'approche de l'apprentissage autonome des réseaux de neurones pour le pilotage en temps réel basée sur l'optimisation via simulation.

Ainsi, nous avons détaillé dans un premier temps, les expérimentations que nous avons menées sur le premier exemple : l'atelier flowshop que nous avons choisi à partir des travaux de Barett et Barman (Barett et Barman, 1986). Dans un second temps, nous avons mené une série d'expérimentations sur un deuxième exemple : un atelier Jobshop lui aussi inspiré de la littérature (Fonseca et Navaresse, 2002). Dans les deux exemples, les réseaux de neurones expérimentés ont acquis par l'apprentissage basé sur l'optimisation via simulation la connaissance nécessaire pour prendre les bonnes décisions de pilotage pour les deux ateliers flowshop et jobshop.

CONCLUSION GENERALE ET PERSEPECTIVES

Le pilotage d'ateliers nécessite de prendre des décisions complexes et souvent en temps réel qui sont prises très empiriquement. Toutefois, l'apprentissage offre des possibilités très intéressantes pour ce type de pilotage, et dans la mesure où le pilotage s'effectue sur la base d'un état du système changeant dynamiquement au cours du temps, il est très difficile de disposer d'exemples des « meilleures pratiques ».

Plusieurs auteurs ont montré que la simulation était capable de fournir des exemples, mettant en relation des stratégies de décision avec des performances, de telle sorte que des outils d'apprentissage automatique puissent en extraire des connaissances utiles et construire les ensembles d'apprentissage. Pour cela, la plupart des travaux dans la littérature utilisent des ensembles d'apprentissage issus des modèles de simulation des systèmes de production étudiés, mais ceci présente des inconvénients liés à la prise en compte des changements qui est indispensable pour les changements des stratégies de production au cours du temps. En effet, la prise en compte de tels changements en temps réel dans les systèmes de production est nécessaire dans des systèmes qui sont confrontés à des imprévus qui peuvent survenir au cours du processus de production (commandes urgentes, manque de matière première, pannes de machine, absentéisme, etc) et qui peuvent nuire aux objectifs de la production. Dans ce sens, plusieurs travaux ont essayé de prendre en compte ce type des changements. Les chercheurs ont proposé des solutions de décomposition des temps de production et les simuler sur des périodes courtes, d'autres ont proposé des seuils de contrôle à partir desquels ils révisent l'état du système et prennent en compte les changements qui ont amené à atteindre ce seuil. Malheureusement la solution de décomposition ne résout pas totalement l'aspect temps réel puisque au cours même de la courte période simulée l'état du système peut changer et ce dernier ne peut être pris en compte que dans la période qui suit. De même pour les seuils de contrôle, puisque il est très difficile de déterminer le bon seuil à partir duquel la prise d'une nouvelle décision de production est nécessaire. A notre connaissance il n'existe pas d'approche qui assure un apprentissage des méthodes d'apprentissage automatique et qui prend en compte ces changements d'état dans les systèmes de production en temps réel.

Alors, nous avons présenté dans ces travaux de thèse une approche qui surmonte la difficulté d'élicitation des connaissances à partir des modèles de simulation pour l'apprentissage des

méthodes d'apprentissage automatiques, de plus cette approche prend en compte les changements d'état des systèmes en temps réel.

En effet, nous avons présenté le concept d'apprentissage autonome de réseaux de neurones pour prendre les décisions de pilotage dans un système de production. Ce type d'apprentissage vise à extraire de façon autonome des connaissances d'un modèle de simulation, sans utiliser d'ensembles d'apprentissage, difficiles à constituer dans le cas du pilotage d'ateliers.

Dans la démarche présentée, l'apprentissage s'effectue par optimisation via simulation des paramètres d'un réseau de neurones par rapport à un objectif. Après optimisation via simulation, le réseau de neurones possède la capacité de décider quelle action de pilotage entreprendre, en fonction de l'état de l'atelier, pour satisfaire à l'objectif visé pour le pilotage.

Ainsi, nous avons montré comment un système pouvait acquérir des connaissances de façon autonome, grâce à l'utilisation combinée d'approches de simulation et d'optimisation, et d'un réseau de neurones.

Les connaissances sont acquises sans intervention d'opérateur, sans expertise préalable et sans ensemble d'apprentissage. La propriété des réseaux de neurones en tant qu'approximateurs universels de fonctions complexes et non linéaires, rendent leur usage très attractif dans ce contexte.

Nous avons généré des connaissances relatives au fonctionnement de l'atelier en temps réel et en considérant chaque changement d'état dans l'atelier à chaque instant de prise de décision. Autrement dit, nous cherchons à générer des connaissances par simulation sans utiliser des ensembles d'apprentissage préalablement déterminés. Comme nous l'avons présenté dans les chapitres précédents, des travaux possédant quelques liens avec l'approche que nous proposons sont présentés dans (Geiger *et al.*, 2006) pour générer une nouvelle règle de priorité de passage de pièce. Les principes exposés ici diffèrent fortement de l'approche par programmation génétique proposée par ces auteurs. Par ailleurs, nos propositions ont vocation à s'appliquer à des problèmes beaucoup plus larges de pilotage d'atelier.

Afin d'illustrer l'efficacité et le potentiel de notre méthode, nous l'avons appliquée à un atelier flowshop simplifié, déjà étudié dans la littérature (Barett et Barman, 1986) et à un atelier Jobshop aussi inspiré de la littérature (Fonseca et Navaresse, 2002) ; Dans les deux cas, le réseau de neurones a été capable d'auto-apprendre à piloter le système et à choisir en temps réel les règles

les plus pertinentes en fonction de l'état courant de l'atelier. Les résultats obtenus montrent que le réseau de neurones a été capable d'auto-apprendre à contrôler l'atelier en temps réel, et que les connaissances générées sont extrêmement pertinentes.

Cette approche peut être appliquée à un large spectre de problèmes de pilotage. Une fois entraîné hors ligne, le réseau de neurones peut être installé sur l'atelier réel. Une des conditions essentielles pour l'applicabilité de cette approche est, pour l'instant, que les caractéristiques de l'atelier restent suffisamment stables au cours du temps. Si des changements majeurs devaient intervenir (e. g. ajout de machines supplémentaires), alors il pourrait devenir nécessaire de ré entraîner le réseau de neurones avec un modèle de simulation mis à jour. L'ajustement du réseau en fonction des changements de caractéristiques est une des perspectives de recherche liée à ce travail.

La principale limite de ce type d'approche réside principalement dans les temps de calcul nécessaires. La méthode d'optimisation choisie doit permettre une convergence rapide vers de bonnes solutions. Néanmoins cette approche parait extrêmement prometteuse. Nos recherches futures intègrent de nombreuses directions. Parmi celles-ci on relève l'amélioration de l'optimisation (Siarry *et al.* 1997), une meilleure gestion des problèmes de temps de calcul, une meilleure prise en compte des aspects stochastiques dans la comparaison des solutions lors de l'optimisation via simulation et l'application à d'autres types de problèmes (chaînes logistiques, etc.).

Bibliographie

Aissani N. et Beldjilali B., 2008, Use of machine learning for continuous improvement of the real time heterarchical manufacturing control system performances. *International Journal of Industrial and Systems Engineering*, Vol. 3, No. 4, pp 474-497.

Alam, M.R., Lee K.S., Rahman, M. et Zhang, Y.F., 2003 Process planning optimization for the manufacture of injection moulds using a genetic algorithm. *International Journal of computer Integrated Manufacturing*, 16(3), pp. 181-191.

Alberto, I., Azcàrate, C., Mallor, F, et Mateo, P.M., 2002, Optimization with simulation and multi-objective analysis in industrial decision-making: A case study. *European Journal of Operational Research*, 140(2), pp. 373-383.

Aler, R., Borrajo, D., Isasi, P., 2002, Using genetic programming to learn and improve control knowledge. *Artificial Intelligence*, vol. 141, p.29-56.

Andradottir, S., 1998, Simulation optimization. *Handbook of Simulation*, Chapter 9, Wiley and Sons, New York, p. 307-333.

Archiméde B., 1991, Conception d'une architecture réactive distribuée et hiérarchisée pour le pilotage des systèmes de production, Thèse de doctorat, Université de Bordeaux I, France, 1991.

Azadivar, F., 1992, A tutorial on simulation optimization. *Proceedings of the 1992 Winter Simulation Conference*, p. 198-204.

Azadivar, F. Et Tompkins, G., 1999, Simulation optimization with qualitative variables and structuralmodel changes : A genetic Algorithm approach. *European Jouranl of Operational Research*, 113, pp.169-182.

Bäck T., 1996, Evolutionary Algorithms in Theory and Practice, Oxford University Press.

Bajic E. Y. Sallez, 2007, Proposition de projet transversal exploratoire du GDR MACS : « Système Contrôlé par le Produit », *Journées prospectives STP*, Lyon, 11 & 12 juin.

Bäck T. et Schwefel H.P., 1993, An overview of Evolutionary Algorithms for Parameter Optimization. *Evolutionary Computaion*, 1(1), pp. 1-23.

Barrett, R.T. et Barman, S., 1986, A SLAM II Simulation study of a simplified flow shop. *Simulation* , Vol 13, n°5, p. 181-189.

Blanc P., Demongodin I. et Castagna P., 2008. A holonic approach for manufacturing execution system design: An industrial application. *Engineering Application of Artificial Intelligence*, Vol N° 21, Issue 3, pp. 315-330.

Bourrières J.P., Baptiste P., Bernard A., Lopez P., Morel G., Pierreval H., Portmann M.C., 2007. Comité d'Experts Productique : prospective de recherche, 15 janvier 2007.

Bertel S. et Billaut J.C., 2003, A genetic algorithm for an industrial multiprocessor flow shop scheduling problem with recirculation. *European Journal of Operational Research*

Cardin O. et Castagna P., 2009, Using online simulation in Holonic manufacturing systems. *Engineering Application of Artificial Intelligence*, *Article In Press*.

Chan K.K. et Spedding T.A., 2001, On-line optimization of quality in a manufacturing system. *International Journal of Production Research,* 39:1127-1145, 2001.

Collins N.E., Eglese R.W., et Golden B.L., 1988, Simulated annealing : An annotated bibliography. *American Journal of Mathematical and Management Sciences*, 8 :209.307.

Cornuéjols A., Miclet L., Kodratoff Y., Apprentissage artificiel, Eyrolles, 2002. Chapitres 2 et 9.

Cornuéjols A., Miclet L., Apprentissage artificiel Concepts et algorithmes, Eyrolles, 2003.

Creighton D.C. et Nahavandi S., 2002, Optimizing discrete event simulation models using a reinforcement learning agent. *Proceedings of the 2002 Winter Simulation Conference*, pp. 1945-1950.

Dengiz, B et Alabas, C., 2000, Simulation optimization using tabu serach. *Proceedings of the 2000 Winter Simulation Conference*, pp. 805-810.

Dindeleux E., 1992, Proposition d'un modèle et d'un système interactif d'aide à la décision pour la conduite d'atelier, Thèse de doctorat, Université de Valenciennes, 1992.

Dolgui, A., Ereemev, A., Kolokolov, A., Sigaev, V., 2000, A Genetic Algorithm for Buffer Allocation in Production Line with Unreliable Machine. *Proc. of the international workshop on discrete Optimization methods in scheduling and computer-Aided Design*, pp. 26-32.

Dreyfus G., Martinez J.-M., Samuelides M., Gordon M. B., Badran, F., Thiria S., Hérault L., 2004, Réseaux de neurones: méthodologies et applications. *édition EYROLLES Avril 2004.*

Eglese R.W., 1990, Simulated annealing: a tool for operational research. *European Journal of Operational Research,* **46**: 271–281.

El-Bouri, A. et Shah P., 2006, A neural network for dispatching rule selection in a job shop. *International Journal of Advanced Manufacturing Technology*, 31(3-4), 342-349.

Fonseca D. J. et Navaresse D., 2002, Artificial neural networks for job shop simulation. *Advanced Engineering Informatics*, 16:241-246.

Fu, M. C., 1994, Optimization via simulation: a review. *Annals of Operation research*, 53 p. 199-247.

Fu, M.C., 2002, Optimization for simulation: Theory vs Practice. *INFORMS Journal on Computing*, Vol. 14, N°3, pp. 192-215.

Garcia L.L. et Bolivar A.P., 1999, A simulator that uses Tabu search to approach the optimal solution to stochastic inventory models. *Computers & Industrial Engineering*, 37(1-2), pp. 215-218.

Geiger, C.D., Uzsoy, R., Aytug, H., 2006, Rapid modeling and discovery of priority dispatching rules: an autonomous learning approach. *Journal of Scheduling,* vol. 9, p. 7-37.

Glover, F., 1989, Tabu Search Part I. *ORSA Journal on Computing*, pp. 190-206.

Glover, F. et Laguna, M., 1997, Tabu Search. *Kluwer Academic Publishers*, Norwell, Masachsetts.

Goldberg, D.E., 1989, Genetic Algorithm in Search, Optimization, and Machine Learning. Addition-Wesley, Reading, MA.

Gouyon D., Pétin J.F., Morel G., Control Synthesis For Product-Driven Automation. IFAC WODES'04, *7th Workshop on Discrete Event Systems*, Reims, 22-24 septembre 2004.

Habchi G., 2001, Conceptualisation et Modélisation pour la Simulation des Systèmes de Production, *Habilitation à diriger des recherches, Université de Savoie*, 2001.

Berchet C., 2000, Modélisation pour la simulation d'un système d'aide au pilotage industriel, Thèse de doctorat, INPG, Grenoble, France, 2000.

Habchi G. et Labrune Ch., 1994, Study of lot sizes on job shop systems performance using simulation.Vol. N°2, Issue 6, pp. 277-289.

Haddock, J. And Mittenthal J., 1992, Simulation optimization using simulated annealing. *Computers and Industrial Engineering,* 22:387-395.

Hajek. B., 1988, Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13 :311.329.

Héritier C., 1991, Une aide à la conception des systèmes de production basée sur la simulation et l'analyse de données. Thèse de Doctorat, Ecole Nationale des Mines, saint-Etionne.

HO, Y.C., Shi, L., Dai, L. et gong, W.B., 1992, Optimizing discrete event dynamic systems via the gradient surface method. *Discrete Event Dynamic systems*, 2, pp. 99-120.

Holland, J.H., 1975, Adaptation in Natural and artificial Systems. Ann ardour, MI: University of Michigan Press.

Hornik K., Stinchcombe M., White H.,1989, Multilayer Feedforward Networks are Universal Approximators. *Neural Networks,* Vol. 2, pp. 359-366.

Hu N.F., 1992, Tabu search method with random moves for globally optimal-design. *International Journal for Numerica Methods in Engineering*, 35, pp. 1055-1070.

Hurrion R. D., 1997, An example of simulation optimisation using a neural network metamodel: finding the optimum number of kanbans in a manufacturing system. *Journal of Operational Research Society*, 48: 1105-1112.

Huyet A.-L. (2004). Extraction de connaissances pertinentes sur le comportement des systèmes de production : une approche conjointe par optimisation évolutionniste via simulation et apprentissage. Thèse de doctorat, Université Blaise-Pascal Clermont 2, N°d'ordre 1528, EDSPIC : 305.

Huyet, A-L., Paris, J-L., 2004, Synergy between evolutionary optimization and induction graphs learning for simulated manufacturing systems. *International Journal of Production Research,* vol. 42, n° 20, p.4295– 4313.

Huyet A.L., 2006, Optimization and analysis aid via data-mining for simulated production systems. *European Journal of Operational Research*, 173, pp 827–838.

Jayaraman V. et Ross, A., 2003, A simulated annealing methodology to distribution network design and management. *European Journal of Operational Research*, 144, pp. 629-645.

Kallel G., 1985, Proposition d'une conduite décentralisée coordonnée (CODECO) pour un atelier de fabrication , Thèse de Doctorat en Automatique, Institut National Polytechnique de Grenoble, soutenue le 28 juin 1985.

Key K. Lee, 2008, Fuzzy rule generation for adaptive scheduling in a dynamic manufacturing environment. *Applied Soft Computing*, 8, 1295–1304.

Kilmer R.A., Smith A.E., Shuman L.J., 1999, Computing confidence intervals for stochastic simulation using neural network metamodels. *Computers & Industrial Engineering*, 36: 391-407.

Kirkpatrick S., Gelatt C. D. et Vecchi M.P., 1983, Optimization by Simulated Annealing. *Science*, Number 4598, 13May 1983, volume 220, 4598, p.671-680.

Krajewski L.J. King B.E., Ritzman L.P., Wong D.S., 1987, Kanban, MRP and shaping the manufacturing environment. *Management Science*, 33(1), pp. 39-57.

Kouiss K., Pierreval H. et Merbaki N., 1995, Toward the use of a multi-agent approach to the dynamic scheduling of flexible manufacturing systems. *International Conference on Industrial Engineering and Production Management (IEPM'95)*, Marrakech, Maroc, pp. 118-125.

Kushner, H.J., 1963, A new method for locating the maximum of an arbitrary multipeak curve in the presence of noise. *Proc. of the Joint Automatic Control Conference*.

Kwak T. S., Suzuki T., Bae W. B., Uehara Y. et Ohmori H., 2005, Application of neural network and computer simulation to improve surface profile of injection molding optic lens. *Journal of materials processing technology*, 170:24-31.

Law, A.M. et Kelton W.D., 2000, Simulation Modelling and Analysis, 3rd esition. New York : McGraw-Hill.

Le Moigne J.L., 1990, La théorie du système général : théorie de la modélisation. 2ème édition, Paris, 1990.

Lereno, E., Morello, B. et Baptiste, P., 2001, Systèmes d'aide au paramétrage d'un logiciel en ordonnancement. *Proceedings of MOSIM'01*, pp. 363-369.

Li, D-C., Wu C-S., Tsai T-I. et Lina Y-S., 2007, Using mega-trend-diffusion and artificial samples in small data set learning for early flexible manufacturing system scheduling knowledge. *Computer and Operations Research,* 34, 966-982.

Li, D-C., Wu C-S., Tsai T-I. et Lina Y-S., 2007, Using mega-trend-diffusion and artificial samples in small data set learning for early flexible manufacturing system scheduling knowledge. *Computer and Operations Research*, 34, 966-982.

Lutz, C.M., Davis, K.R et Sun, M.H., 1998, Determining buffer location and size in production lines using tabu search. *European Journal of operational Research*, 106, pp. 301-316.

Maes, S. and Meganck, S. and Manderick, B., 2004, Multi-Agent Identifcation of Causal Effects. *The Second European Workshop on Multi-Agent Systems*, Barcelona, Spain.

Manz E.M., Haddock, J. et Mittenthal, 1989, Optimization of an automated manufacturing system simulation model using simulated annealing. *Proceedings of the 1989 Winter simulation Conference*, pp. 390-395.

Martin, A.D., Chang, T.M., Yih, Y. et Kincaid, R.K., 1998, Using tabu search to determine the number of kanbans and lot sizes in a genetic kanban system. *Annals of Operations research*,78, pp. 201-217.

McFarlane D., Sarma S., Chirn J.L., Ashton K., 2002, The intelligent product in manufacturing control and management, *15th Triennal IFAC World Congress*, Barcelone, Espagne.

Mcfarlane, D., Sarma, S., Chirn, J.L., Wong, C.Y., Ashton, K. 2003: Auto id systems and intelligent manufacturing control. *Engineering Applications of Artificial Intelligence* 16/4: 365–376.

Metan, G. et Sabuncuoglu I., 2005, A simulation based learning mechanism for scheduling systems with continuous control and update structure. *M.E. Kuhl, N.M. Steiger, F.B. Armstrong, and J.A. Joines, Proceedings of the 2005 Winter Simulation Conference*, 2148-2156.

Metropolis N., Rosenbluth A.W., Rosenbluth M.N., Teller A.H., et Teller E., 1953, Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21 :1087–1091.

Metan, Sabuncuoglu et Pierreval (2009) In Press

Monostori L., 2003, AI and machine learning techniques for managing complexity, changes and uncertainties in manufacturing, *Engineering Applications of Artificial Intelligence*, 16 (4) (2003) 277–291.

Morel, G., Valckenaers, P., Faure, J.M., Pereira, C., Diedrich, C., 2007, Survey paper on manufacturing plant control challenges and issues. *Control Engineering Practice*.

Mouelhi, W., Pierreval H. et Gningue M. B., 2007, Dynamic Selection of Dispatching Rules: a Neural Network Approach. I*nternational Conference on Industrial Engineering and Systems management*, 324 CD-ROM.

Mouelhi W., Huyet A.-L. et Pierreval H., 2007, Combining simulation and artificial neural networks : an overview. *6th EUROSIM Congress on Modeling and Simulation*, (CD-ROM).

Mouelhi W. et Pierreval H., 2008, Construction d'heuristiques basées sur des règles de priorité pour job shop par optimisation via simulation. *7ème Conférence Internationale de Modélisation et Simulation (MOSIM)*, CD-ROM, 5 pages.

Mouelhi W. et Pierreval H., 2007, Le concept d'apprentissage autonome et son application au pilotage des systèmes de production. *7ème Congrès international de génie industriel (CIGI)*, CD-ROM, 9 pages.

Mouelhi-Chibani W. et Pierreval H., 2009, Training a neural network to select dispatching rules in real time. *Computers & Industrial Engineering*, *In Press, Corrected Proof*, *Available online 29 March 2009*.

Osei-Bryson K.M., 2004, Evaluation of decision trees : a multi-criteria approach. *Computers &Operations Research*, 31, pp. 1933-1945.

Paris J.-L., Pierreval H., 2001, A Distributed Evolutionary Simulation Optimisation Approach for the Configuration of Multiproduct Kanban Systems. *International Journal of Computer Integrated Manufacturing*, Vol. 14, N° 5, pp. 421-430.

Pétin J.-F., Gouyon D. et Morel G., 2007, Supervi-sory synthesis for product-driven automation and its application to a flexible assembly cell. *Control En-gineering Practice*, **15**(5), 595-614.

Pflug, G.C., 1984, Optimizing simulated systems, *Simuletter*, 15(4), pp. 6-9.

Pierreval, H., Caux, C., Paris, J.L. et Viguier, F., 2003, Evolutionary approaches to design and organization of manufacturing systems. Computers & industrial Engineering, 44, pp. 339-364.

Pierreval H. et Tautou, L., 1997, Using evolutionary algorithms and simulation for the optimization of manufacturing systems. *IIE Transactions*, 29 (3), pp. 181-190.

Pierreval H., 1999, Proposition de typologie des décisions en temps réel agissant sur les flux des systèmes de production, *2ème Congrès MOSIM'99, Annecy*, p.331, octobre 1999.

Pierreval H. Training a neural network by simulation for dispatching problems. *Proceedings of the Third Rensselaer International Conference on Computer Integrated Engineering*, New York, 332-336, 1992.

Pierreval, H., Ralambondrainy, H., 1990, A simulation and learning technique for generating knowledge about manufacturing systems behavior, *Journal of Operational Research Society,* vol. 46, n° 6, p. 461-474.

Pierreval H., 2006, Simulation combinée discret/continu : étude du cas d'une fonderie", *6ème Conférence Francophone de MOdélisation et SIMulation – MOSIM'06*, Rabat, Maroc, 2006.

Pierreval H., Bruniaux R. et Caux C., 2007, A continuous simulation approach for supply chains in the automotive industry. *Simulation Modelling Practice and Theory*, Vol. 15, N° 2, pp. 185–198, 2007.

Pierreval H., 1992, Training a neural network by simulation for dispatching problems. *Proceedings of the Third Rensselaer International Conference on Computer Integrated Engineering*, New York, 332-336, 1992.

Pierreval H. et Huntsinger R., An investigation on neural network capabilities as simulation metamodels. *Proceedings of the 1992 Summer Computer Simulation Conference, California, CA*, 413-417, 1992.

Pierreval. H., 1992, Training a neural network by simulation for dispatching problems. *Proceedings of the Third Rensselaer International Conference on Computer Integrated Engineering*, New York, 332-336, 1992.

Pierreval H., 1988, Data-analysis oriented techniques for learning about manufacturing control with simulation. *Proceedings of the 2nd European Simulation Multi-Conference : Factory of the Future*, pp. 61-66.

Pierreval H., 1994, Using Multiple Correspondence Analysis in the Analysis of Simulation Experiments : a Study of Dynamic Scheduling Strategies. *Int. Trans. Opt. Res.*, 1(2), pp. 147-157.

Pujo P., Ounnar F., 2007, Vers une approche holonique des systèmes mécatroniques complexes – Proposition d'un système de pilotage auto-organisé et isoarchique, *JESA*, 41 (6), p. 673-706.

Pujo P. et Kieffer J.P., 2002, Concepts fondamentaux du pilotage des systèmes de production. *Fondements du pilotage des systèmes de production*, Traité IC2 Productique (Pujo P. and J.P. Kieffer. Ed), p. 25-50, Hermès, Paris.

Pujo P. et Brun-Picard D., 2002, Pilotage sans plan prévisionnel, ni ordonnancement préalable. *Méthodes du pilotage des systèmes de production*, Traité IC2 Productique (Pujo P. and J.P. Kieffer. Ed), p. 129-162, Hermès, Paris.

Reeves, C.R., 1993, Genetic algorithms, *Modern heuristic Techniques for Combinational Problems*, edited by C.R. Reves (orient Longman). Chapter 4, pp. 151-188.

Santos Osório F., 1998, Un système hybride neuro-symbolique pour l'apprentissage automatique constructif. Thèse de doctorat, L'Institut National Polytechnique de Grenoble, 1998.

Roy D., 1998, Une architecture hiérarchisée multi- agents pour le pilotage réactif d'ateliers de production, Thèse de Doctorat en Automatique - Productique, Université de Metz, soutenue le 15 janvier 1998.

Roy, D., Anciaux, D., Vernadat F., 2001, SYROCO : A novel multi-agent shop-floor control system, *Journal of Intelligent Manufacturing*, Vol. 12, n°3, p. 295-307.

Shiue, Y-R. et Guh R-S., 2006, Learning based multi-pass adaptive scheduling for a dynamic manufacturing cell environment. *Robotics and Computer-Integrated Manufacturing*, 22, 203-216.

Sebag M., Gallinari P., 2002, Apprentissage artificiel : acquis, limites et enjeux. *Apprentissage artificiel, Actes des deuxièmes Assises nationales du GDR I3*, Cépaduès Editions, p 303-333.

Siarry, P., Berthiau G., Durbin F., Haussy J., 1997, Enhanced simulated annealing for globally minimizing functions of many-continuous variables. *ACM Transactions on Mathematical Software*, 23(2), p. 209-228

Sonquist J.A. et Morgan J.J., 1963, Problems in the analysis of survey data and proposal**,** *J.A.S.A.* 58, pp. 415-435.

Spear W.M., Bäck K.A., Fogel T., Degaris D.H., 1993, An overview of evolutionary comptation, *Proceedings of the European Conference on Machine Learning*, (Spring: New-York), pp. 442-459.

Stuckman, B.E. et Easmon, E.E., 1992, a comparison of bayesian/sampling global optimization techniques.IEEE Transactions on systems, Man and cybernetics, 22, pp. 1024-1031.

Sun R-L., Ding H., Xiong X. et Du R., 2004, Iterative learning scheduling: a combination of optimization and dispatching rules. *Journal of Manufacturing Technology management,* 15 (3), 298-305.

Tamani K., Boukezzoula R. et Habchi G., 2006, High level Petri nets based approach for analyzing conceptual objects for production systems simulation. *Proceedings Volume from the 12th IFAC Conference,* Saint-Etienne, France, pp. 339-344.

Tautou, L. et Pierreval, H., 1995, Using evolutionary algorithm and simulation to optimize manufacturing systems. Proc. of ETFA'95, 3, pp.509-516.

Thomas A., 2004, De la planification au pilotage pour les chaines logistiques. Habilitation à Diriger des Recherches, Université Henri Poincaré – Nancy 1.

Trentesaux D., 1996, Conception d'un système de pilotage distribué, supervisé et multicritère pour les systèmes automatisés de production », Thèse de Doctorat en automatique- Productique, Institut National Polytechnique de Grenoble*,* soutenue le 24 janvier 1996.

Van Laarhoven P.J.M. et Aarts E.H.L., Simulated Annealing : Theory and Applications. *Kluwer Academic Publishers*, Boston, 1987.

Wang K.-J., Chen J. C. et Lin Y.-S., 2005, A hybrid knowledge discovery model using decision tree and neural network for selecting dispatching rules of a semiconductor final testing factory. *Production Planning and Control,* 16: 665-680, 2005.

Wang K. -J., Chen J. C. et Lin Y.-S., 2005, A hybrid knowledge discovery model using decision tree and neural network for selecting dispatching rules of a semiconductor final testing factory. *Production Planning and Control,* 16: 665-680, 2005.

Wang J-Y. et Yih Y., Using neural networks to select a control strategy for automated storage and retrieval systems (AS/RS). *International Journal of Computer Integrated Manufacturing*, 10:487-495, 1997.

Wetter, M., Polak, E., 2003, A convergent optimization method using pattern search algorithms with adaptive precsion simulation. *Eigth International IBPSA Conference, Eindhoven, Netherlands*, p. 1393-1400.

Yildirim, M.B., Cakar T., Doguc U. et Meza J.C., 2006, Machine number, priority rule, and due date determination in flexible manufacturing systems using artificial neural networks. *Computers and Industrial Engineering*, 50, 185-194.

Youssef A., 1998, Architecture distribuée multi- experts avec contrôle hiérarchique pour le pilotage des systèmes de production, Thèse de Doctorat en Productique Automatisée, Université de Metz, soutenue en juin 1998.

Zhao Z. et De Souza R., 2001, Fuzzy rule learning during simulation of manufacturing resources. *Fuzzy Sets and Systems*, 122, pp. 469-485.

Zighed, D.A. et Rakotomalala, R., 2000, Graphes d'induction, Hermès.

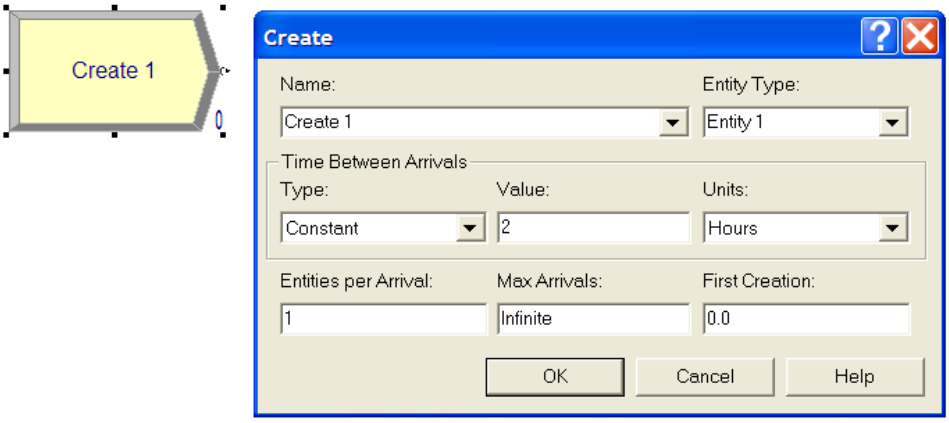
Annexe A

DESCRIPTION DE QUELQUES BLOCS PERMETTANT LA CONSTRUCTION D'UN MODÈLE

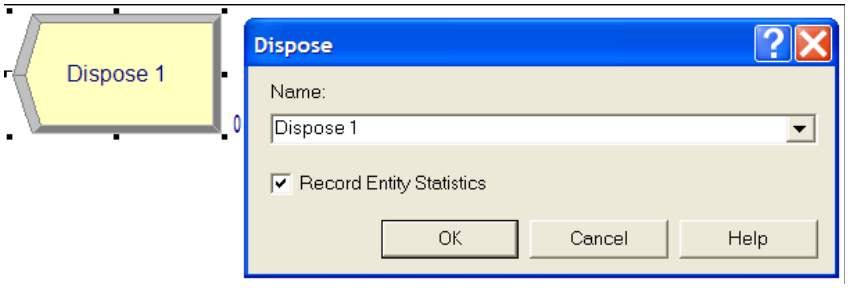
a) *Create* (issu du *template Basic Process*) : Un bloc ***Create*** permet de créer des entités. Celui représenté dans la figure suivante est intitulé *Create* 1 (champ *Name* = *Create* 1). Sont indiqués :

- La période de création des lots d'entités (cadre *Time Between Arrivals*, par exemple : champ *Type* = *Constant,* champ *Value* = 2),
- La taille des lots (champ *Entities per Arrival* = 1),
- Le nombre total de lots à créer (champ *Max Arrivals* = *Infinite*),
- La date de création du premier lot (champ *First Creation* = 0).

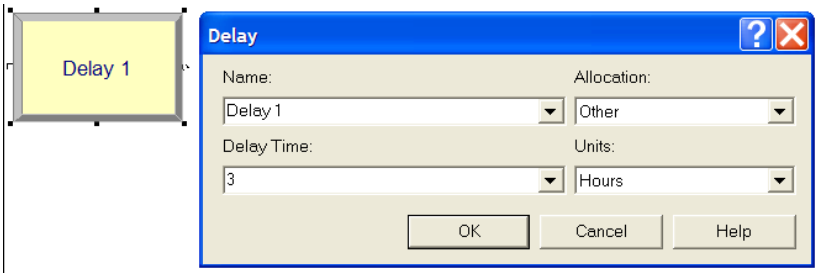
Les valeurs considérées sont telles qu'une entité est créée toute les deux unités de temps à partir de l'instant 0, ceci une infinité de fois.



b) *Dispose* (issu du *template Basic Process*) : Un bloc ***Dispose*** permet de détruire des entités. Celui représenté dans la figure suivante est intitulé *Dispose* 1 (champ *Name* = *Dispose* 1), une entité entrant dans ce bloc est immédiatement détruite.



c) *Delay* (issu du *template Advanced Process*) : Un bloc ***Delay*** permet de retarder le passage d'entités. Celui représenté dans la figure suivante est intitulé *Delay* 1 (champ *Name* = *Delay* 1), quand une entité entre dans ce bloc, elle y reste inconditionnellement pendant la durée (aléatoire ou non) indiquée dans le champ *Delay Time*.



d) *Seize* (issus du *template Advanced Process*) : Une entité présente dans un bloc ***Seize*** ne peut sortir de ce bloc que s'il existe un nombre suffisant de ressources **disponibles** (le nombre et le type de ressources étant spécifiés dans le bloc) ; en attendant l'entité est stockée (« patiente ») dans une file d'attente interne au bloc ***Seize***. Le fait qu'une entité sorte du bloc indique que les ressources, disponibles en nombre suffisant, sont « saisies » (et donc plus disponibles).

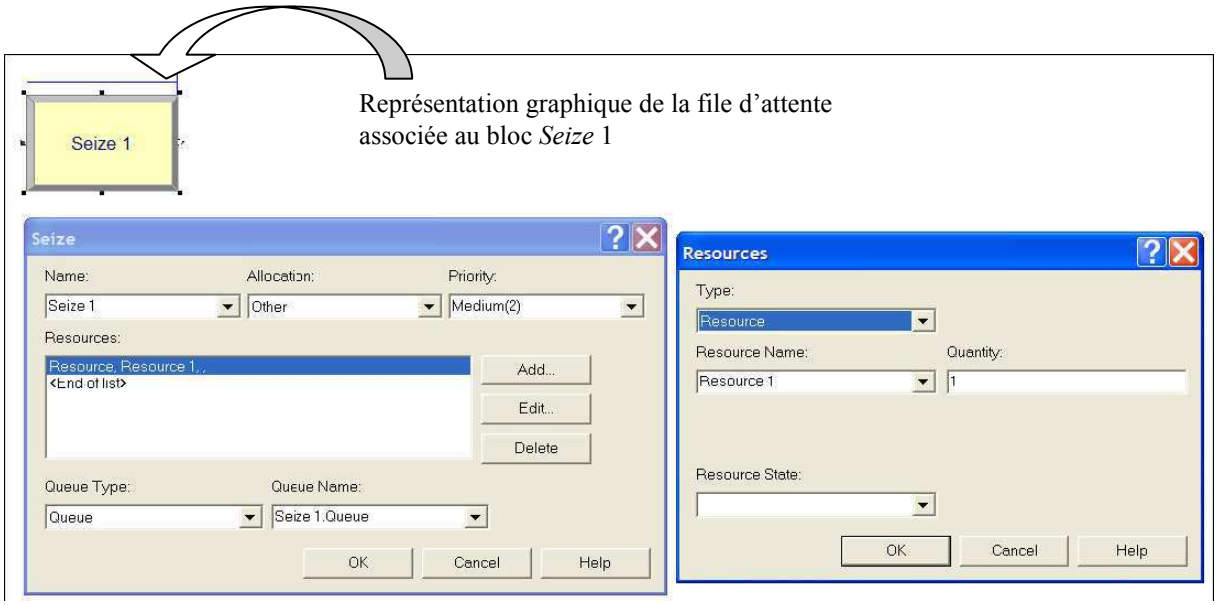
Le bloc représenté dans la figure suivante est intitulé *Seize* 1 (champ *Name* = *Seize* 1). Pour simplifier la compréhension, considérons que seulement un type de ressource est concerné (dans l'exemple, *Resource* 1), alors :

- le nom de la ressource est spécifié dans le champ *Resource Name*, soit *Resource Name* = *Resource* 1 (l'ajout d'un autre type de ressource donnerait lieu à une ligne supplémentaire

dans la liste *Resources*),

- le nombre (minimum) de ressources (de type *Resource* 1) disponibles est spécifié dans le champ *Quantity*, par exemple *Quantity* = 1.

Sachant qu'une ressource peut ne pas être disponible, les entités, en attente d'un nombre suffisant de ressources disponibles, sont stockées dans une file d'attente, intégrée (en amont) au bloc ***Seize***, et dont le nom est indiqué dans le champ *Queue Name* (soit *Queue Name* = *Seize* 1.*Queue*).

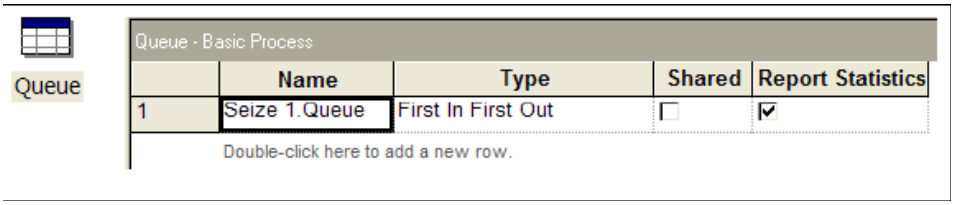


Une file d'attente est caractérisée (configurée) par le bloc ***Queue*** (issu du *template Basic Process*, appartenant au cadre expérimental et donc non traversé par une entité), voir la figure suivante :

- le champ *Name* permet de déclarer une file d'attente, par exemple *Seize* 1.*Queue*,

- le champ *Type* permet d'indiquer le mode de gestion de la file d'attente. Par défaut, le mode de gestion est de type *First In, First Out* (FIFO).

Le bloc ***Queue*** permet de définir plusieurs files d'attente dans un même modèle.

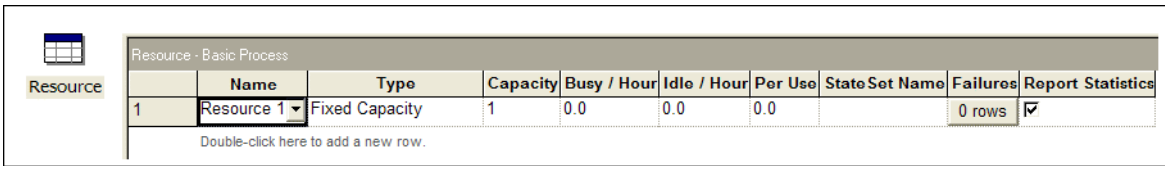


Les types de ressource, ainsi que le nombre pour chaque type de ressources, sont indiqués dans le bloc ***Resource*** (issu du *template Basic Process*, appartenant au cadre expérimental et donc non traversé par une entité), voir la figure suivante :

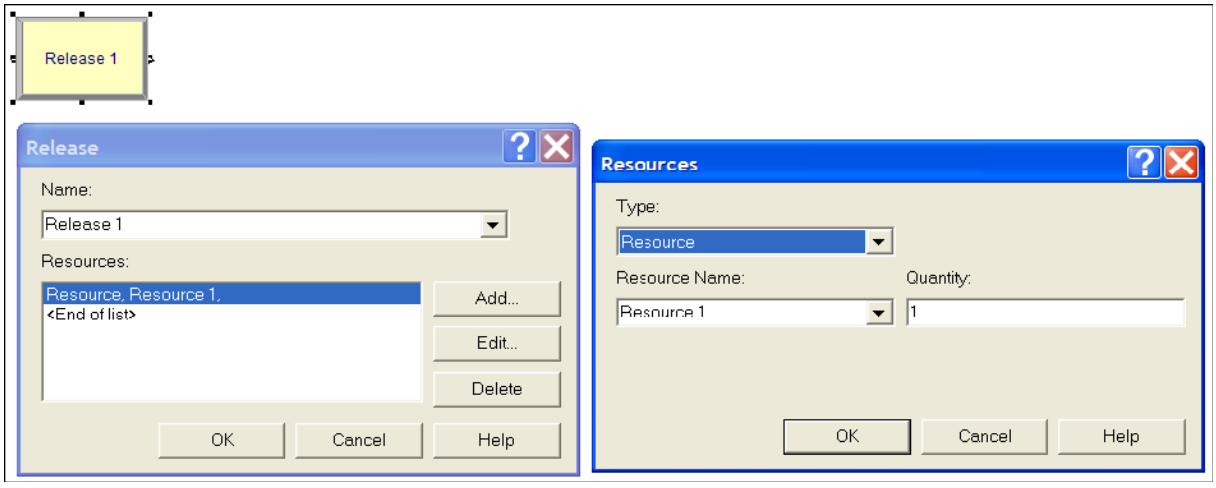
- le champ *Name* permet de déclarer une ressource, par exemple *Resource* 1,

- le champ *Capacity* permet de définir le nombre d'unité de la ressource, par exemple 1.

Le bloc ***Resource*** permet de définir plusieurs types de ressources dans un même modèle.



e) *Release* (issu du *template Advanced Process*) : Un bloc ***Release*** permet de « relâcher » des ressources. Celui représenté dans la figure suivante est intitulé *Release* 1 (champ *Name* = *Release*1). Quand une entité entre dans ce bloc, elle libère (relâche) la, ou les ressources dont le nom est spécifié dans le champ *Resource Name*, par exemple *Resource* 1, le nombre de ressources libérées est spécifié dans le champ *Quantity*, par exemple 1. On peut noter que l'exécution de cette tâche est instantanée, autrement dit le temps de passage d'une entité dans un bloc ***Release*** est nul. Pour simplifier, seul un type de ressource est concerné (dans l'exemple, *Resource* 1), l'ajout d'un autre type de ressource donnerait lieu à une ligne supplémentaire dans la liste *Resources*.



f) *Assign* (issu du *template Basic Process*) : Un bloc ***Assign*** permet d'assigner une valeur, notamment, à un attribut, une variable (éventuellement propre à SIMAN, par exemple relative à l'état d'une ressource), durant l'exécution d'une simulation. Quand une entité entre dans un bloc *Assign*, l'expression - logique ou mathématique - spécifiée dans le champ *New Value* est évaluée et assignée, selon le contenu du champ *Type* (*Attribute, Variable, ...*), à un attribut (rattaché à

l'entité « activant » le bloc) ou une variable. Dans la figure suivante, le bloc intitulé *Assign* 1 (champ *Name* = *Assign* 1) permet de déclarer :

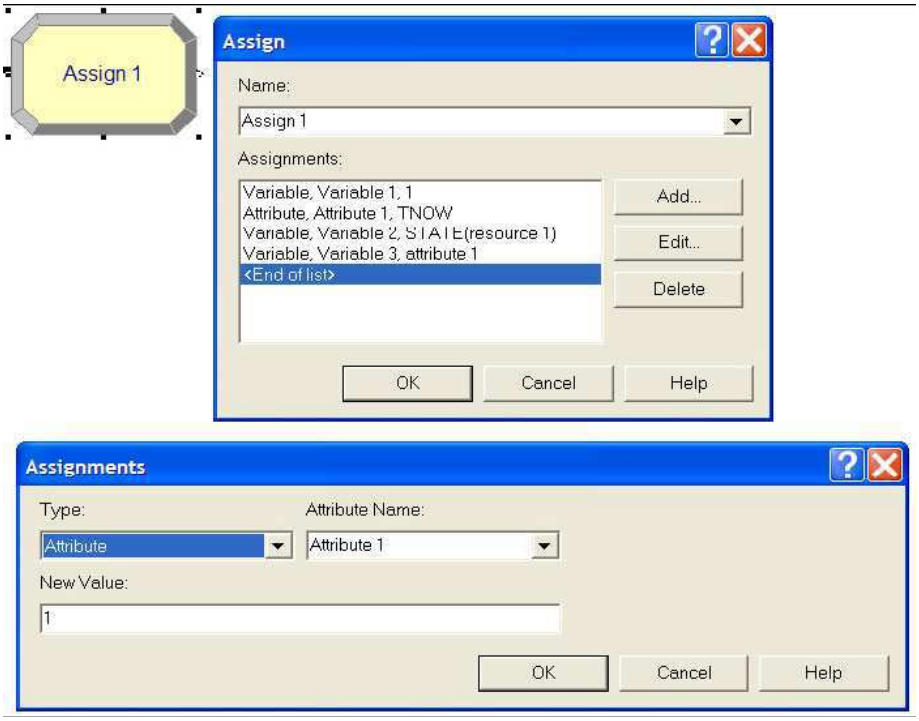
- une variable *Variable* 1 à 1 ;

- un attribut *Attribute* 1 à TNOW ;

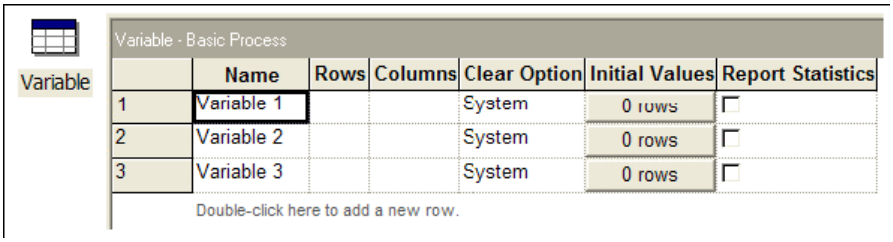
- une variable *Variable* 2 à *STATE*(*resource* 1). La variable *STATE*(*resource* 1) restitue l'état courant de la ressource *resource* 1 (les valeurs possibles sont : -1=*Idle* ; -2=*Busy* ;

-3=*Inactive* ; -4=*Failed*) ;

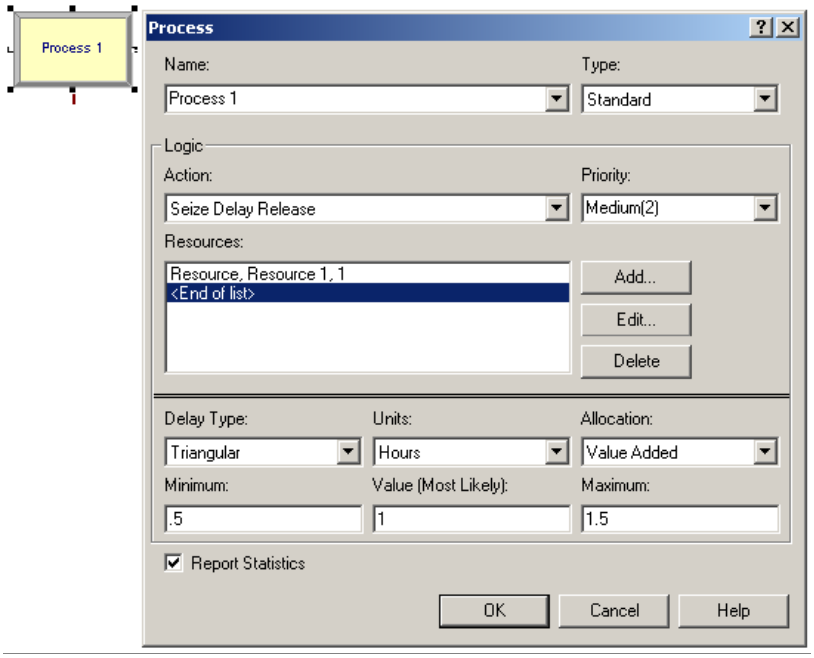
- une *Variable* 3 à *Attribute* 1.



Le bloc **Variable** (issu du *template Basic Process*, appartenant au cadre expérimental et donc non traversé par une entité) permet de déclarer des variables.



j) ***Process*** (issu du *template Basic Process*) : Un bloc ***Process*** permet de simuler le comportement d'une machine, sachant que différents modes de fonctionnement sont autorisés selon le contenu du champ *Action* (situé dans le cadre *Logic* lorsque le champ *Type* = *Standard*). Un bloc ***Process***, intitulé *Process* 1 (champ *Name* = *Process* 1), est décrit dans la figure suivante.



Lorsque le champ *Action* contient la valeur :

i) *Delay*, la machine se ramène à un simple bloc *Delay*, ce qui permet de simuler un temps de traitement (voir le cadre *Delay Type* pour assigner un temps de traitement) et le fait qu'il n'y a pas de contrainte vis-à-vis de la ressource de la machine.

2*i*) *Seize Delay*, la machine nécessite une, voire plusieurs ressources (voir le cadre *Resources* pour assigner le type, ainsi que le nombre, de ressources concernées) durant un temps (relatif au temps de traitement) *minimum* indiqué dans le cadre *Delay* (le relâchement de la ressource est supposé réalisé en aval).

3*i*) *Seize Delay Release*, *idem*. au cas 2*i*) avec une gestion au niveau du relâchement de la ressource « saisie ».

4*i*) *Delay Release*, *idem*. au cas 3*i*) sans la gestion de l'allocation de la, voire des ressources

nécessaires au traitement d'une pièce (cette gestion est supposée réalisée en amont du

bloc).

DESCRIPTION DE QUELQUES BLOCS PERMETTANT L'ANALYSE D'UN MODÈLE

Les blocs décrits au VIII.2 permettent de modéliser un système physique, sans pour autant fournir d'informations (exceptées celles données par défaut dans le rapport final). La collecte d'informations spécifiques se fait en utilisant des blocs supplémentaires tels que le bloc Record et le bloc Static. Dans ce qui suit nous décrivant le bloc Static à titre d'exemple.

2) Le bloc ***Statistic*** (issu du *template Advanced Process*, appartenant au cadre expérimental et donc non traversé par une entité) permet, selon le contenu du champ *Type,* de :

- collecter durant la simulation des statistiques issues de variables *SIMAN*, telles que le nombre d'entités contenues dans une file d'attente (variable *NQ*, *cf.* 2.*a*), le *taux d'occupation* d'une ressource (variable *NR*, *cf.* 2.*b*) (*Type = Frequency*) ;

- spécifier les fichiers dans lesquels seront sauvegardées les données d'observations individuelles, par exemple :

- celles issues d'un bloc ***Record*** **(**relatif à un *compteur* ou un *tally*) (*Type = Count* ou *Tally*) (*cf.* 2.*c*)) ;

- celles issues d'une variable *SIMAN* (le nombre d'entités contenues à chaque instant dans une file d'attente, le *taux d'occupation* à chaque instant d'une ressource (*Type = Frequency*) (*cf.* 2.*c*)).

2.a) Lorsque le champ *Type = Frequency* et le champ *Frequency Type = Value*, le bloc ***Statistic*** permet, à travers la variable ***NQ*** (abréviation de *Number in Queue*) - mise à jour automatiquement par SIMAN, de disposer du nombre d'entités contenues dans une file d'attente.

Par exemple, en supposant défini une file d'attente intitulée *Process* 1.*Queue*, la variable *Statistic* 3 (champ *Name* = *Statistic* 3) permet de connaître le nombre d'entités présentes à chaque instant dans la file d'attente *Process* 1.*Queue* lorsque le champ *Expression* = *NQ*(*Process* 1.*Queue*) (voir figure ci-dessous).

2.b) Lorsque le champ *Type* = *Frequency* et le champ *Frequency Type* = *Value*, le bloc ***Statistic*** permet, à travers la variable ***NR*** (abréviation de *Number of busy Resource units*) - mise à jour automatiquement par SIMAN, de disposer du *taux d'occupation* d'une ressource.

Par exemple, une machine constituée de *n* ressources permet le traitement en parallèle de *n* pièces. Une ressource peut être *occupée* (*busy*), soit *disponible* (*idle*). Par exemple, considérons une machine, représentée par un bloc *Process* 1, utilisant une ressource *Resource* 1 de capacité égale à 3 (donnée déclarée dans un bloc ***Resource***), la variable *Statistic* 4 (champ *Name* = *Statistic* 4) permet de connaître le nombre de ressources *Resource* 1 *occupées* au cours du temps (ce nombre pouvant être égal à 0, 1, 2 ou 3) lorsque le champ *Expression* =*NR*(*Resource* 1) (voir figure ci-dessous).

2.c) Le nom du fichier - ainsi que son répertoire si celui-ci est différent du répertoire contenant l'application - dans lequel sont enregistrées les données issues d'un bloc ***Record*** ou d'une variable *SIMAN* (*NQ* ou *NR*) est indiqué dans le champ *Counter Output File* ou *Tally Output File* (selon que le bloc *Record* est relatif à un *compteur* ou un *tally*) ou *Output File* (pour une variable *SIMAN NQ* ou *NR*). Le contenu du champ *Type* est respectivement égal à *Counter, Tally* ou *Frequency* selon que les données sont issues d'un *compteur,* d'un *tally* ou d'une variable *SIMAN* (*NQ* ou *NR*). Afin de disposer des données au format *csv* (abréviation de *comma-separated-value*, « *comma* » signifiant virgule), il suffit de mettre l'extension *.csv* au fichier de sauvegarde. Ce format est reconnu notamment par *MatLab* (*via* la commande *CSVREAD*) et *Excel*. Au préalable, il faut indiquer que le fichier de sauvegarde est au format texte ; pour cela cocher la case *Write Statistics Output Files as Text* accessible *via* le menu *Run/Setup/Run Control/Advanced.*