ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN GÉNIE MÉCANIQUE
M.Ing.

PAR
CHERIF MAKREM

OPTIMISATION DE L'ORDONNANCEMENT PAR L'APPROCHE HYBRIDE BASÉE SUR LES RÉSEAUX DE NEURONES

MONTRÉAL, LE 21 DECEMBRE 2004

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Dao Thien My, directeur de mémoire
Département de génie mécanique à l'École de Technologie Supérieure

Mme Sylvie Nadeau, présidente de jury
Département de génie mécanique à l'École de Technologie Supérieure

M. Richard Lepage, membre de jury externe
Département de génie de la production automatisée à l'École de Technologie Supérieure

IL A FAIT L'OBJET D'UNE PRÉSENTATION DEVANT JURY
LE 23 NOVEMBRE 2004
À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

OPTIMISATION DE L'ORDONNANCEMENT PAR L'APPROCHE HYBRIDE BASÉE SUR LES RÉSEAUX DE NEURONES

Cherif Makrem

SOMMAIRE

Les problèmes d'ordonnancement se posent dans de nombreux domaines tels que la productique et l'informatique. Leur variété vient de la diversité des données, des contraintes et des critères d'optimisation qu'ils impliquent. Ce mémoire traite le problème de l'ordonnancement déterministe dans un atelier à tâches («Job shop» et cellules de production) sur la base d'une utilisation des réseaux de neurones. Ce problème est un problème d'optimisation NP-Complet lorsque le nombre de machines et de tâches est supérieur à deux. Les données sont constituées de l'ensemble des tâches à exécuter, de leurs gammes opératoires, de leurs durées ainsi que de l'ensemble des machines. Les contraintes prises en compte sont les contraintes de partage de ressources et de précédences. Les variables de décision concernent les dates de début et les dates de fin des opérations. Un critère d'optimisation est considéré, le "makespan" qui correspond à la minimisation du temps total de travail. L'utilisation des réseaux de neurones est intéressante car le parallélisme intrinsèque de ces derniers offre, a priori, une possibilité de traiter des problèmes de grandes tailles dans un temps limité. Une étude comparative des différentes approches de réseaux de neurones utilisés dans l'optimisation a été effectuée. Elle nous a permis d'apprécier les potentialités des réseaux de neurones de Hopfield dans le traitement d'une variété de problèmes d'optimisation. Notre travail a consisté ensuite à ajuster les particularités des réseaux de neurones à mettre en oeuvre pour la résolution de notre problème d'ordonnancement. Les propositions de ce mémoire sont articulées autour d'une utilisation combinée des réseaux de neurones avec un algorithme heuristique. Cette combinaison peut apporter, dans la majorités des cas, une amélioration nette de la qualité de solutions. Enfin, une des particularités fondamentales des réseaux de neurones étant la robustesse, il nous a paru intéressant de chercher dans quelle mesure il est possible d'explorer utilement cette propriété. Cette démarche nous a conduit à la proposition d'un réseau récent de Hopfield (Quantized Hopfield), qui nous permet d'obtenir les solutions optimales très fréquemment et beaucoup plus rapidement que d'autres réseaux de Hopfield.

OPTIMIZATION OF SCHEDULING BY THE HYBRID APPROACH BASED ON THE NEURAL NETWORKS

Cherif Makrem

ABSTRACT

Scheduling problems occur often in various domains such as manufacturing and computer science. Their variety is due to the diversity of data, constraints and optimization criteria which they imply. This thesis studies the problem of deterministic scheduling in a job-shop and cellular manufacturing, using an artificial neural network approach. This is an NP-complete optimization problem when the number of machines and the number of tasks exceed 2. The data is composed of the set of tasks to be executed, their operational ranges, their duration and the set of machines on which they will be executed. The constraints which have been considered, are : resource sharing and precedence constraints. Decision variables correspond to the start and end dates of operations. One optimization criteria have been considered : makespan (which corresponds to the minimization of the total scheduling duration). The use of an artificial neural network is interesting since its intrinsic parallelism facilitates handling large sized problems in a limited time. A comparative study of various approaches of neural networks used in optimization has been undertaken in this thesis. This has enabled us to appreciate the potential of Hopfield neural networks for solving a variety of optimization problems. Our next contribution consists of adjusting specifics of neural networks to be implemented for solving our scheduling problem. Proposals made in this thesis are centred around : a combined utilisation of neural nets and heuristic algorithms. This combination can bring (under certain conditions) a significant improvement of the quality of solutions. Finally, robustness being a fundamental particularity of neural nets, it appeared interesting for us to explore under which conditions this property could be usefully exploited. This final approach led us to propose a recent kind of neural networks (Quantized Hopfield), which allows us to obtain optimal design solutions very frequently and much more quickly than other Hopfield networks.

REMERCIEMENTS

Je tiens à remercier mon directeur de recherche Dao Thien My pour son soutien, sa disponibilité et ses précieux conseils dans la présente recherche. Son bon jugement et son enthousiasme ont été pour moi une source de motivation.

Je tiens à remercier Madame Sylvie Nadeau, Professeure à l'École de Technologie Supérieure, département de génie mécanique, de me faire l'honneur de s'intéresser à ce travail et de faire partie du jury. Je remercie également Monsieur Richard Lepage, Professeur à l'École de Technologie Supérieure, département de génie de la production automatisée, pour l'intérêt qu'il a bien voulu porter à ce travail et d'avoir accepté de faire partie du jury. De plus, je salue tous mes bons collègues du laboratoire qui ont su m'encourager et partager leurs précieuses informations quand le besoin urgent se présentait.

Enfin, je remercie tous les membres de ma famille qui ont su me transmettre cette force de persévérer et d'entreprendre qui m'accompagne, à chaque fois, dans tout projet que je réalise.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS

JS	De l'anglais pour Job Shop
PC	Production cellulaire
RN	Réseaux de neurone
RNH	Réseaux de neurone de Hopfield
NS	Approche de Nowicki et Smutnicki
CTQ	Contrôle total de la qualité
EPT	Entretien productif total
C_{max}	Temps de fin de la dernière opération dans l'ordonnancement
F_{total}	Le temps de passage total des tâches dans l'atelier
PODA	Par ordre d'arrivée
TOC	Dont le Temps d'opération est le plus court
TOL	Dont le Temps d'opération est le plus long
PNE	Programmation en nombres entiers
PLEM	Programmation linéaire en variables entiers ou mixtes
PNLEM	Programmation non linéaire en variables entiers ou mixtes
PNL	Programmation non linéaire
RS	Recuit simulé
AG	Algorithmes génétiques
PM	Perceptron multicouche
MB	Machine de Boltzmann
DG	Diagramme de Gantt
SF	Solution faisable
RL	Recherche locale
SMED	Technique de changement de moule en une minute ou moins

INTRODUCTION

Dans les entreprises manufacturières, l'augmentation continuelle des coûts d'équipement et de main-d'œuvre et le développement rapide des nouvelles technologies de l'information et de la communication posent des problèmes que le contexte de la compétitivité mondiale impose de résoudre. Un de ces problèmes est le développement de la performance en termes de durée des cycles de fabrication, respect des dates de livraison prévues et l'adaptation réactive face aux variations du carnet commercial.

En plus, les exigences des marchés soumis à une forte concurrence et les attentes des clients deviennent de plus en plus fortes en termes de qualité, de coût et de délais de mise à disposition. C'est pourquoi il est crucial de disposer de méthodes et d'outils de plus en plus performants pour l'organisation et la conduite de la production. Parmi ces outils est l'ordonnancement qui est un élément indispensable de la planification et s'impose comme l'outil pour la simulation d'alternatives de production. "L'ordonnancement est la planification de l'exécution de la production à très court terme" [42]. Il permet d'avoir une meilleure vision de l'activité globale de l'entreprise, de construire des planifications optimisées des ressources en tenant compte des différentes contraintes et de fournir des délais fiables aux clients.

Dans le premier chapitre nous examinons la nécessité de l'ordonnancement pour la planification de l'entreprise et le contrôle des différents systèmes de production. Par la suite nous présentons des approches classiques et leurs limites pour la résolution des problèmes d'ordonnancement.

Dans le deuxième chapitre nous présentons les différents types de réseaux de neurones utilisés dans l'optimisation, leurs applications et leurs limites. Ensuite nous comparons les différentes approches neuronales afin de choisir la meilleure que nous pouvons l'adopter pour résoudre le problème d'ordonnancement dans un environnement de "Job Shop" et de production cellulaire.

Dans le troisième chapitre, qui constitue l'élément essentiel de notre travail, nous proposons l'application d'une approche hybride qui combine les réseaux de neurones, ainsi qu'une procédure de recherche locale, pour résoudre le problème d'ordonnancement dans un environnement de "Job Shop" (JS) ou de production cellulaire (PC), qui sont des problèmes NP-complets (voir définition en annexe 2). Le Réseau de Neurone de Hopfield (RNH) et la procédure de recherche locale de Nowicki et Smutnicki (NS) sont présentés. En approches combinées, RNH est utilisé pour obtenir les solutions faisables et l'approche NS est pour améliorer la qualité des ces solutions.

Dans le dernier chapitre nous présentons la validation et le degré d'efficacité de la nouvelle approche présentée dans le chapitre précèdent appuyé par des cas typiques en reprenant des exemples de la littérature. Chaque phase de notre approche sera démontrée et les résultats seront analysés et critiqués.

En résumé, ce mémoire illustre les alternatives intéressantes d'application des systèmes de neurones, pour résoudre les problèmes complexes d'optimisation d'ordonnancement, puisque sa force est dans la convergence rapide à la solution et la réduction du nombre de calcul.

CHAPITRE 1

L'ORDONNANCEMENT DE LA PRODUCTION

1.1 Introduction

Compte tenu de l'évolution des marchés et leurs exigences, ce chapitre examine la nécessité de l'ordonnancement pour la planification de l'entreprise et le contrôle des différents systèmes de production. Un des systèmes les plus connus, le "Job Shop", y est abordé de même que la production cellulaire. Une présentation des approches classiques et leurs limites pour la résolution des problèmes d'ordonnancement vient compléter ce chapitre.

1.2 Problématique de l'ordonnancement de la production

L'environnement actuel des entreprises est caractérisé par des marchés soumis à une forte concurrence et sur lesquels les exigences et les attentes des clients deviennent de plus en plus fortes en termes de qualité, de coût et de délais de mise à disposition (Figure 1). Cette évolution se renforce par le développement rapide des nouvelles technologies de l'information et de la communication qui permettent une relation directe entre les entreprises et entre les entreprises et leurs clients. Dans un tel contexte, la performance de l'entreprise se construit selon deux dimensions :

- une dimension technologique qui vise à développer les performances intrinsèques des produits mis sur le marché afin de satisfaire aux exigences de qualité et de réduction du coût de possession des produits. L'innovation technologique joue ici un rôle important et peut constituer un élément différenciateur déterminant pour la pénétration et le développement d'un marché. Il faut relever à ce niveau que l'évolution technologique rapide des produits et les exigences de personnalisation de ces produits qu'attendent les marchés conduisent les entreprises à abandonner souvent les productions de masse pour

s'orienter vers des productions de petites ou moyennes séries, sinon à la commande. Ceci leur impose donc d'avoir des systèmes de production flexibles et évolutifs, capables de s'adapter aux exigences et aux besoins des marchés rapidement et efficacement.

- une dimension organisationnelle, qui vise à développer la performance en termes de durée des cycles de fabrication, respect des dates de livraison prévues, gestion des stocks et des en-cours, adaptation et réactivité face aux variations du carnet commercial. Cette dimension joue un rôle de plus en plus important, dans la mesure où les marchés sont de plus en plus versatiles et évolutifs et exigent des temps de réponse des entreprises de plus en plus courts. Il faut donc disposer de méthodes et d'outils de plus en plus performants pour l'organisation et la conduite de la production.

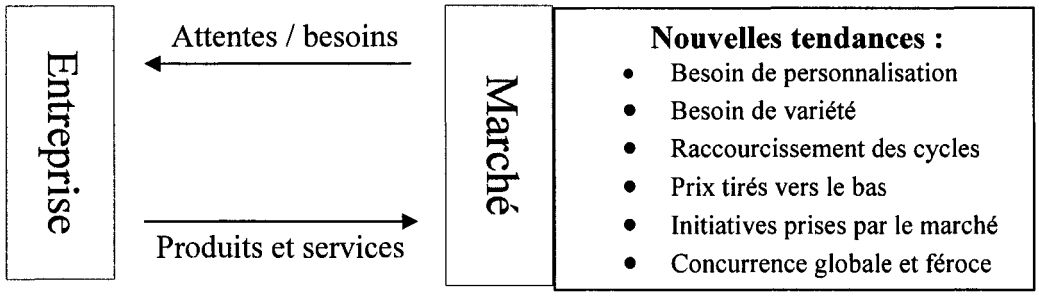


Figure 1 Nouvelles exigences et tendances des marchés [41]

Pour atteindre ces objectifs, l'organisation repose en général sur la mise en œuvre d'un certain nombre de fonctions, parmi lesquelles la fonction ordonnancement joue un rôle essentiel. Cette fonction vise en effet à contrôler les priorités, à préserver le bien-être du personnel, à prévoir les activités de maintenance préventive et prédictive, à organiser l'utilisation des ressources technologiques et humaines présentes dans les ateliers de l'entreprise pour satisfaire soit directement les demandes des clients, soit les demandes issues d'un plan de production préparé par la fonction de planification de l'entreprise.

Cette fonction doit organiser l'exécution simultanée de multiples travaux à l'aide de ressources polyvalentes disponibles en quantités limitées, ce qui constitue un problème complexe à résoudre, un problème d'allocation de ressources sujet à des contraintes. Bien sûr, au sein des entreprises, cette fonction a toujours existé, mais elle est aujourd'hui confrontée à des problèmes de plus en plus complexes à résoudre, à cause du grand nombre de travaux qu'elle doit simultanément réaliser, et pour chacun d'eux des délais de réalisation de plus en plus courts.

1.3 Le système traditionnel "Job Shop"

L'environnement de "Job Shop" (JS) est un système de production où les équipements sont regroupés par types d'opérations, l'aménagement d'usine est habituellement divisé en plusieurs ateliers, chacun se composant de machines de types semblables pour exécuter une fonction (figure 2). L'avantage d'un tel aménagement est l'utilisation maximale du parc machine. De plus, il permet l'achat d'un équipement à usages multiples, peut supporter une variété de requis de procédé, flexible dans l'allocation du personnel et des équipements et moins vulnérables aux arrêts causés par un bris mécanique ou une absence. Dans ce type d'aménagement, le cheminement des produits est généralement irrégulier, la taille de lot est habituellement grande pour réduire au minimum le prix de revient unitaire de mise en course, ce qui crée des difficultés de manutention et de circulation entre les diverses sections. En outre, la planification de la production, spécialement l'ordonnancement, y est relativement complexe puisque n'importe quelle section peut constituer un point d'entrée, de transit ou de sortie du produit dans le système. Les responsables de cette planification ont alors de plus en plus recours à l'informatique dans leur travail.

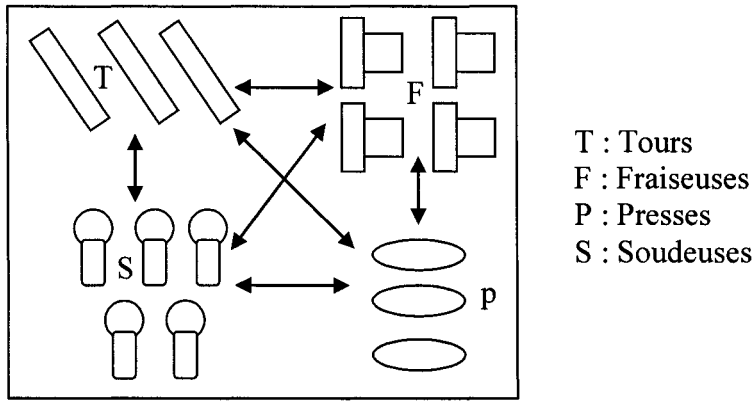


Figure 2 Disposition de système Job Shop

1.4 Du système "Job Shop" à la production cellulaire

Il semble bien que dans certains secteurs d'activité, les lignes de production de masse aient vécu et que les nouvelles contraintes du marché obligent les producteurs à évoluer vers des structures plus adaptables et réactives comme les cellules flexibles. Une flexibilité accrue et une durée de cycle plus courte sont les raisons qui poussent les entreprises à opter pour la production cellulaire. La production cellulaire permet en effet de fabriquer un grand nombre de variantes de produits en limitant au maximum les gaspillages. La variété croissante des produits et la disponibilité de machines flexibles remettent cette méthode de production à l'ordre du jour. Pour cela, les entreprises de production sont confrontées à de nouveaux défis en raison de l'âpre concurrence (prix attractifs, introduction rapide de nouveaux produits dans un appareil de production existant, fortes variations de la demande et du délai de livraison).

Le principe de base de cet aménagement est de tenir compte de la similitude entre les diverses opérations et du groupement de produits de même type. Dans un système de production cellulaire, l'usine est divisée en plusieurs cellules de travail, chacune se compose de différents types de machines pour exécuter des fonctions multiples et consécutives (Figure 3). L'avantage majeur qu'on en retire est la simplification des

activités de pilotage, d'ordonnancement et de planification du processus, et la réduction des temps de mise en route. De plus, ce type d'aménagement réduit les stocks de produits en cours, les délais de fabrication et les temps d'attente des pièces, et améliore les conditions de travail.

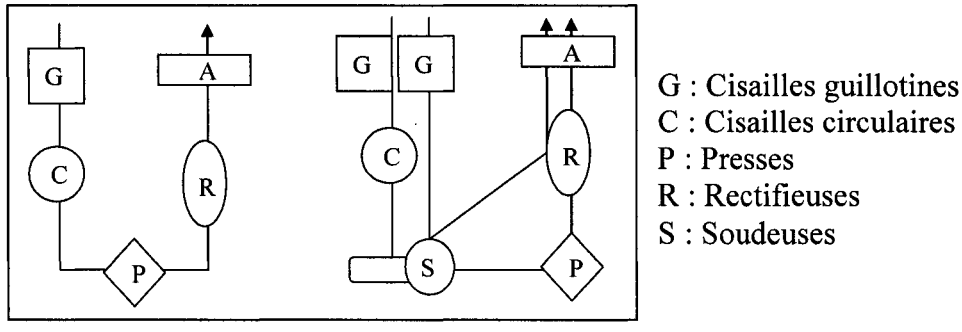


Figure 3 Disposition de système production cellulaire

En 1993 Guo R.S. et al. [2] ont montré qu'afin de réaliser des avantages concurrentiels, le système PC doit tenir compte de la conception de l'aménagement d'usine, la commande du flux opérationnel, les besoins en équipements, et la mobilisation du personnel (accroissement de la polyvalence des employés et de leur niveau de connaissance). Ceci est systématiquement montré dans la figure 4.

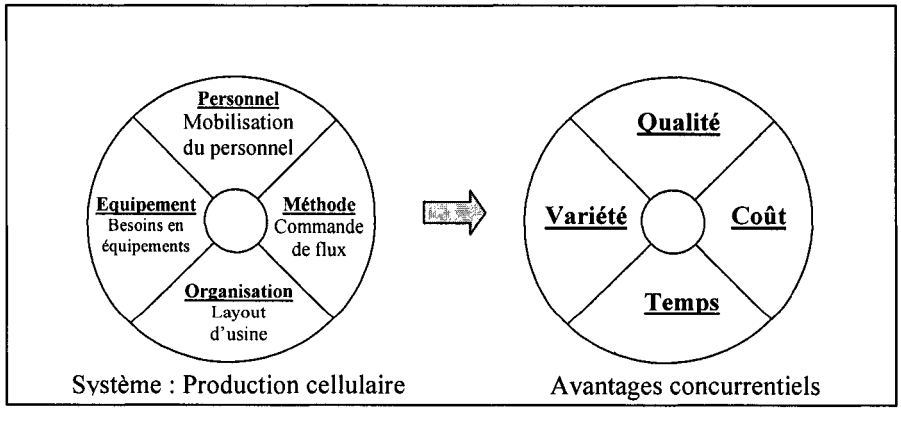


Figure 4 Utilisation de système PC pour réaliser des avantages concurrentiels [2]

Dans les sections suivantes, nous expliquerons en détails chacune de ces issues.

1.4.1 Conception d'un aménagement d'usine

La figure 2 montre la disposition d'un système JS ainsi que les flux de production. Chaque atelier se compose des machines semblables pour exécuter une fonction. Dans ce type d'aménagement la communication est difficile entre les employées. Naturellement, les personnels tendent à se concentrer sur leur propre efficacité fonctionnelle. En conséquence, des problèmes sérieux seront trouvés dans une telle disposition :

1- Mauvaise qualité
   - un bon nombre de pièces traitées avant la découverte des problèmes
   - communication faible entre les différents ateliers
2- Faible variété de produits
   - flexibilité faible
3- Coût élevé
   - Inventaire élevé
   - défauts de produits
   - surproduction
4- Longue durée de cycle
   - perte de temps de transport
   - perte de délai d'attente

Les problèmes trouvés dans la disposition JS peuvent être résolus en changeant en une disposition de PC comme montré dans la figure 3. Ici, chaque cellule de travail se compose de différents types de machines pour exécuter des fonctions multiples et consécutives. Les machines en chaque cellule sont consacrées à un ou peu de flux de production et l'arrangement de disposition suit habituellement une forme de U pour réduire la distance physique entre les machines et entre les cellules de travail, et

augmente la communication entre les personnels. En conséquence, des avantages sérieux seront trouvés dans une telle disposition :

1- réduction de la durée de cycle en éliminant :
    - les transports inutiles (les machines sont plus près)
    - les attentes inutiles (opérateurs multidisciplinaires)
2- réduction de coût en éliminant les sources de non-valeur ajoutée telles que :
    - les mouvements inutiles (en U, moins de surfaces libres)
    - le stockage inutile (moins d'espace disponible autour des postes)
    - les tâches inutiles (moins de transferts à d'autres postes)
    - les produits défectueux (l'opérateur est l'unique responsable de plusieurs opérations)
    - la surproduction limitée à une cellule et non à chaque machine

La PC permette d'aménager l'environnement de travail de façon à optimiser les activités et garantir ainsi l'efficacité du travail accompli. Aussi la PC améliore la qualité du produit puisque la distance physique se raccourcit et une meilleure communication et une rétroaction (feedback) plus rapide d'informations sur les défauts sont réalisées.

1.4.2 Commande de flux opérationnel

La commande de flux opérationnel cherche à réduire la durée de cycle de production, l'inventaire et la production au taux de la demande du marché. Donc les concepts de la fabrication à petits volumes, inventaire zéro et production juste à temps seront utilisés dans le système PC.

La première différence opérationnelle entre les "jobs shop" et les cellules de travail est la taille de lot. Puisque la distance physique entre les différents jobs shop est longue, les pièces sont transférées dans de grandes tailles de lot combinées avec l'utilisation de

grands stocks tampon. D'autre part, le système de cellules de travail peut déplacer les pièces d'une machine à une autre dans de petites tailles de lot parce que les processus consécutifs sont physiquement liés ensembles. Ce lien réduit non seulement la durée de cycle mais réduit également le niveau d'inventaire dans l'usine.

La deuxième différence opérationnelle entre les jobs shop et les cellules de travail est la méthode de dégagement des pièces. Le système de production job shop utilise un système en flux **poussé** dans l'usine. Tandis que le système de production cellules de travail adopte le concept «juste à temps» et utilise un système en flux **tiré** quand l'inventaire est inférieur à un niveau prédéterminé. Cette méthode permet un inventaire très réduit et un coût réduit. Elle élimine également les causes des aléas de production par l'amélioration continue, ce qui représente un gros avantage.

1.4.3 Choix d'équipement

Pour soutenir le système PC, les critères de sélection d'équipement incluent la mise en course rapide, peu coûteux, peu volumineux, et simple à maintenir.

La mise en course rapide permet le changement rapide d'un produit à l'autre. La réduction de taille de lot, le niveau d'inventaire, aussi bien que la durée de cycle de production sont les raisons principales pour lesquelles la réduction de temps de mise en course est si importante. Si la machine est trop compliquée et la mise en course prend trop de temps, les gens utilisent de grandes tailles de lot pour réduire le prix de revient unitaire. Il pourrait être justifié dans une basse variété de produit et un volume de production élevé, cependant, il n'est pas approprié à un système dédié à la flexibilité. La technique de réduction des temps de mise en course la plus connus est le système SMED (Single Minute Exchange Die), ou changement de moule en une minute ou moins. Cette technique met l'accent sur la séparation des réglages internes et externes. Les réglages internes ne peuvent être faits que quand la machine est arrêtée, comme le montage ou le

démontage des moules. Les réglages externes peuvent être faits quand la machine est en marche, comme le transport des moules de leur lieu de stockage ou vers celui-ci. L'idée est de faire tous les réglages externes quand la machine est en marche, de convertir le plus grand nombre de réglages internes en réglages externes et de réduire le temps des réglages internes et externes.

Des équipements peu coûteux et des petites machines dédiées, permettent à l'usine d'utiliser des multiples lignes parallèles et d'ajuster le volume de production beaucoup plus facilement pour répondre à la demande du marché.

1.4.4 Mobilisation du personnel

L'ensemble du personnel doit être mobilisé pour identifier les enjeux, éliminer les problèmes et tout ce qui cause des imperfections dans le flux de produits et de services à destination des clients. L'organisation du travail évoluera, en fonction de la maturité des processus, l'élimination du gaspillage et par la mise en place d'équipes autonomes de production et de travailleurs polyvalents. Deux concepts utiles qui devraient être pratiqués dans le système de cellules de travail, le contrôle de la qualité total et l'entretien productif total.

Le contrôle total de qualité (CQT) et l'entretien productif total (EPT) sont des concepts qui font participer tous les employés. Le but de CQT est de réaliser l'amélioration continue de la qualité. Le but de EPT est de réaliser l'efficacité globale du système de production par la participation des travailleurs dans l'entretien productif. Et comme la qualité doit être construite dans le processus plutôt qu'être réalisée par l'inspection, l'entretien productif est beaucoup plus souhaitable que l'entretien de panne. Par exemple, des personnes de production devraient être formées pour faire le ménage courant et l'entretien simple pour empêcher la détérioration des machines. Les opérateurs devraient être formés pour développer des qualifications multifonctionnelles.

Le tableau suivant compare les péjoratives procédures de gestion aux nouvelles procédures de gestion

Tableau I

Comparaison entre les péjoratives et les nouvelles procédures de gestion

Péjoratives procédures	**Nouvelles procédures**
les directeurs contrôlent	les ouvriers sont des penseurs
organisation verticale	organisation horizontale
le bénéfice est en premier	la qualité est en premier
commande	amélioration
une compagnie → des capitaux	une compagnie → des gens

N'importe quel changement exige une grande quantité d'énergie et l'engagement des personnes. Il n'y a malheureusement aucune solution facile, mais avec la compréhension du besoin de changement et avec une vision claire du futur, nous devrions pouvoir la faire se produire. La PC s'intègre particulièrement bien dans un programme de juste à temps qui nécessite plus de flexibilité et d'accroissement de la polyvalence des employés et de leur niveau de connaissances. Le passage vers la PC représente un changement majeur qui mérite au préalable une planification minutieuse, basée sur des résultats à atteindre.

1.5 Les résultats escomptés de la production cellulaire

L'introduction de la PC va plus loin qu'un simple repositionnement des moyens de production et exerce un impact considérable sur l'entreprise. La PC offre généralement les avantages suivants :

- Réduction des coûts et des délais de production
- Amélioration de la qualité (autocontrôle)
- Réduction des stocks de produits en cours (une meilleure utilisation de l'espace)
- Augmentation de la productivité, ce qui peut avoir une grande influence sur le résultat final d'une entreprise
- Optimisation du flux de production (fait diminuer les distances parcourues)
- Augmentation de la fiabilité de livraison

Pour exploiter pleinement le potentiel de la fabrication cellulaire, l'entreprise doit harmoniser tous les éléments clés de son organisation de façon à tirer profit de la conformité des produits, du traitement de l'information et du soutien du client.

1.6 Les exigences et les limites de la production cellulaire

Du point de vue de l'aménagement, on installe des cellules de fabrication (ou groupes de machines) où sont produites toutes les pièces qui ont assez de similitudes pour être considérées comme une «famille». Idéalement, les cellules sont indépendantes les unes des autres de façon à ce que chaque famille de produits se voit allouée des moyens spécifiques. Mais dans la pratique, il est rarement justifié sous l'angle commercial de reproduire toutes les machines nécessaires dans chacune des cellules. Quand une machine doit exécuter des opérations pour plusieurs cellules, l'on crée souvent un "service central" (de préférence lorsqu'il y a surcapacité).

Les compromis entre des exigences contradictoires ou les limitations techniques accroissent la complexité ou s'écartent du flux idéal des marchandises, mais offrent tout de même des avantages de taille par rapport à l'aménagement fonctionnel de l'atelier. À l'avenir, la politique d'investissement d'une entreprise pourra être influencée par le choix de machines plus petites pouvant être intégrées dans la cellule.

Les autres aspects à prendre en considération avant de passer à la fabrication cellulaire sont les suivants :

- la flexibilité du personnel, étant donné que les opérateurs doivent pouvoir exécuter plusieurs tâches au sein d'une cellule.
- un nombre trop élevé de cellules de travail diminue l'efficacité, à plus forte raison si l'on souhaite travailler quasiment à pleine capacité.
- un acheminement fiable des matériaux revêt d'importance.
- une faible fiabilité d'une série de processus réduit fortement la fiabilité de la cellule.
- Trouver des clients intéressés à commander régulièrement de petites quantités de produits

1.7 Différence entre JS et PC au niveau d'ordonnancement

L'environnement "Job shop" est caractérisé par la fabrication des pièces qui diffèrent en terme de besoins de transformation, de matériels, de temps de fabrication, de séquence de traitement et de mise en route. Le problème consiste ainsi à trouver la séquence des tâches, en respectant les contraintes de précédence selon les critères que l'on cherche (minimiser le makespan correspondant au temps de fin de la dernière opération dans l'ordonnancement, minimiser le temps de passage, etc.).

Par contre, l'ordonnancement de la production dans un environnement production cellulaire relève encore d'études à venir puisqu'il n'y pas encore de modèle conçu spécifiquement pour ce concept. Cependant, l'attribution des postes de travail, le cheminement des produits et la configuration des cellules dynamiques sont déjà connus pour un horizon donné. Notons qu'à ce niveau, qu'il peut être stratégique d'attendre le plus longtemps possible avant de reconfigurer les cellules. Autant que possible, un poste de travail doit être déplacé à un moment qui n'engendre pas un arrêt de production.

Donc, dans notre travail, on va supposer que les temps de transport et de mise-en-course sont minimums tout au long du processus dans un atelier JS. Par contre, pour la PC les temps de transport et de mise-en-course sont réellement minimums et on peut les considérer négligeables. Et par conséquent, le contrôle du processus de fabrication dans un environnement PC est plus simple qu'un environnement JS. Cette hypothèse est utilisée par Radharamanan [40].

Dans un environnement JS, trouver un ordre optimum des opérations pour un ensemble de tâches données et un groupe de machines n'est pas une tâche facile [40]. Pour **n** tâches et **m** machines, toutes les combinaisons possibles des ordonnancements sont $(n\ !)^m$. Par contre dans un environnement PC, le problème devient beaucoup plus simple. Quand une famille de pièces est assignée à un groupe donné de machine, la taille de problème devient n!.

Le tableau suivant récapitule les différences entre JS et PC au niveau de l'ordonnancement.

Tableau II

Différences entre JS et PC au niveau de l'ordonnancement

	JS	**PC**
Attribution des postes de travail	à long terme	à court terme
Cheminement des produits	à long terme	à court terme
Temps de transport entre les postes	important	négligeable
Temps de Mise-en-course des machines	long	rapide
Flux de production	produit	famille de produits
Complexité d'ordonnancement	compliqué	plus simple
Optimalité de l'ordonnancement	moins optimal	plus optimal
Combinaisons possibles d'ordonnancement	$(n\ !)^m$	n!

1.8 Approches classiques des problèmes d'ordonnancement

Les méthodes de résolution des problèmes d'ordonnancement sont largement présentées dans la littérature [7]. Elles puisent dans toutes les techniques d'optimisation :

- les méthodes de programmation mathématique [32] sont extrêmement fréquentes dans les applications. Plusieurs heuristiques visent à réduire le volume de l'enveloppe convexe des solutions en ajoutant des contraintes.
- Les méthodes arborescentes [4] sont des méthodes exactes, qui essaient d'énumérer les solutions réalisables de manière à trouver rapidement de bonnes solutions. Appliquées aux problèmes d'ordonnancement, ces approches nécessitent des temps de calcul qui croissent exponentiellement avec le nombre de variables.
- Les méthodes de programmation dynamique [4] sont des techniques énumératives fondées sur l'idée que des solutions à des sous problèmes du problème peuvent aider à guider la recherche de la solution optimale du problème global. Là encore, cette approche atteint ses limites pour les problèmes de grandes tailles.

D'après l'étude présenté par Berard, Azzaro-pantel, Pibouleau et Domenech en 1997 [7], ces méthodes présentent l'avantage de garantir l'optimalité de la solution fournie, si le problème est convexe. Pour des problèmes de taille réelle, ces procédures requièrent des temps d'exécution prohibitifs et sont alors généralement associées à des méthodes heuristiques ou des algorithmes de recherche locale, ce qui leur fait perdre beaucoup de leur rigueur mathématique. Les algorithmes basés sur la programmation mathématique ou sur la programmation dynamique s'avèrent très rapidement inadaptés pour des problèmes de grande taille. La plupart des méthodes proposées pour la résolution des problèmes d'ordonnancement sont du type procédures arborescentes fondées sur une modélisation par un graphe disjonctif. D'autres sont basées notamment sur la programmation linéaire en nombres entiers. Plus récemment, les méthodes de type recuit

simulé (RS) [31] et algorithmes génétiques (AG) [33] ont été utilisées. Les méthodes de type RS, que nous présenterons ultérieurement, sont des méthodes dérivées de la physique statistique et destinées à éviter les minima locaux des fonctions de coût (voir paragraphe 2.5.2.1 page 34) et pour les algorithmes génétiques des solutions potentielles sont considérées comme des individus qui évoluent dans une population.

De nombreuses tentatives de résolution [35] du cas général du job shop ont également été proposées par des algorithmes utilisant des règles de priorité. Parmi les plus classiques, citons PODA (Par ordre d'arrivée), sélection de l'opération disponible le plus tôt, TOC (Dont le temps d'opération est le plus court), sélection de l'opération ayant le plus petit temps opératoire, TOL (Dont le temps d'opération est le plus long) sélection de l'opération ayant le plus grand temps opératoire.

Par ailleurs, la plupart des études menées en génie des procédés n'intègrent pas toutes les caractéristiques majeures que l'on peut rencontrer sur le terrain et qui contribuent largement à la complexité du problème. Ainsi, le problème d'instabilité des produits intermédiaires et la gestion des politiques de stockage ne sont pas toujours considérés.

De même, il y a deux limitations principales à l'utilisation des méthodes de programmation mathématique, programmation dynamique et arborescentes. La première concerne le degré de représentation du système : ces méthodes ne peuvent représenter finement le système productif, qu'en conduisant très vite à des formalismes non linéaires en variables mixtes. La seconde est liée au problème combinatoire : plus le degré de détail et la taille du problème sont importants, plus le caractère combinatoire est marqué.

Pour illustrer cela, le graphe (figure 5) présenté par Berard, Azzaro-pantel, Pibouleau et Domenech (1997) [7] montre les résultats d'un recensement des nombres moyens d'équipements et de produits pour les exemples mentionnés dans la littérature illustrant les principales méthodes d'optimisation (la programmation en nombres entiers (PNE),

programmation linéaire en variables entières ou mixtes (PLEM), programmation non linéaire en variables entières ou mixtes (PNLEM), programmation non linéaire (PNL), méthodes heuristiques, recuit simulé, algorithmes génétiques). En effet, la combinatoire du problème dépend fortement de ces deux paramètres (les nombres moyens d'équipements et de produits).

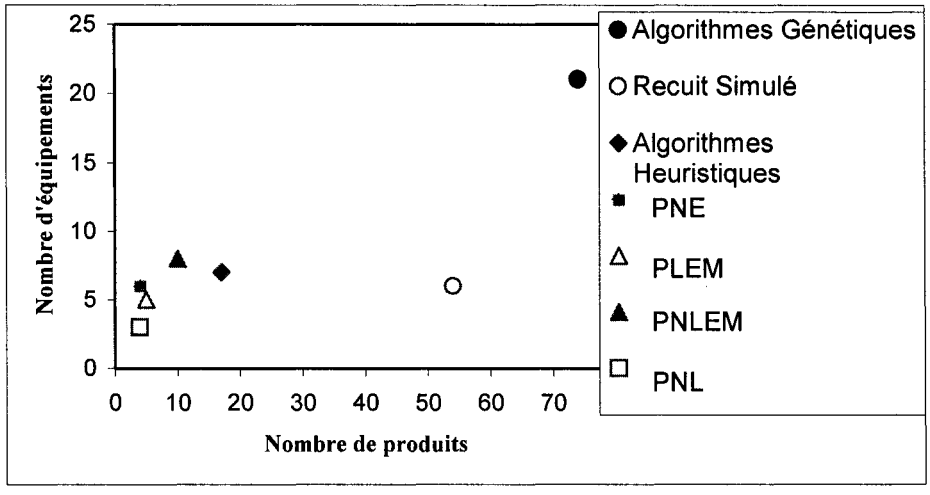


Figure 5 Taille des exemples de la littérature en fonction des méthodes d'optimisation utilisées [7]

Les problèmes limités, par la taille de leur structure et le nombre de contraintes appliquées, rendent praticables la mise en oeuvre de méthodes de résolution par des méthodes déterministes de type PLEM, PNLEM couplées ou non à des méthodes heuristiques. Par contre, quand la taille et le nombre de contraintes du problème augmentent, ces méthodes s'avèrent difficiles voire impossibles à utiliser à cause de l'explosion de la combinatoire sous-jacente [4]. Pour résoudre ce problème, les systèmes de neurones avec leur capacité d'exécuter un grand nombre de calculs dans un temps limité offrent une alternative intéressante. C'est pourquoi dans le second chapitre on va présenter en détail les performances et les limites des différents réseaux de neurones utilisés dans l'optimisation.

CHAPITRE 2

LES APPROCHES NEURONALES ET L'ORDONNANCEMENT

2.1 Introduction

Le fonctionnement des systèmes de neurones, caractérisé par le principe de parallélisme intrinsèque, permet d'exécuter un grand nombre de calculs dans un temps limité et offrent une alternative intéressante pour résoudre le problème complexe d'optimisation d'ordonnancement. L'utilisation des réseaux de neurones comme l'outil d'optimisation de l'ordonnancement est un outil puissant et plus performant que les autres outils classiques. Ce second chapitre présente les différents types de réseaux de neurones utilisés dans l'optimisation, leurs applications et leurs limites, afin de comparer les différentes approches et choisir le meilleur pour l'adopter à résoudre le problème JS et PC.

C'est quoi les réseaux neuronaux?

S'inspirant du mode de fonctionnement des systèmes nerveux et plus particulièrement de celui du cerveau, les réseaux de neurones y empruntent leur structure (ensemble de cellules autonomes), les connexions qui relient ces cellules (mode de communication) et leur capacité d'adaptation et d'apprentissage.

Fondements biologiques :

Les éléments de base du cerveau sont les neurones. Un neurone biologique est composé des parties suivantes :

- le corps cellulaire contient le noyau du neurone et effectue les transformations biochimiques nécessaires à la vie du neurone;
- les dendrites forment une sorte d'arborescence autour du corps cellulaire. Ceux sont elles qui permettent au neurone de capter les signaux qui parviennent de l'extérieur;
- l'axone est la fibre nerveuse qui permet de transporter les signaux émis par le neurone;
- les synapses jouent un rôle fondamental pour permettre aux neurones de communiquer.

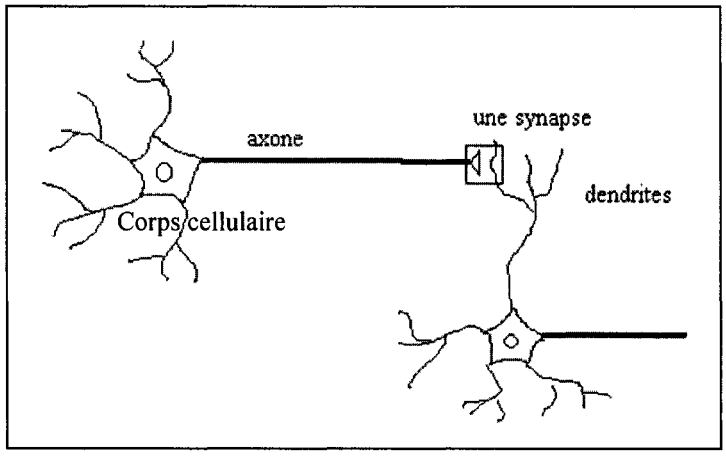


Figure 6 Modèle simplifié de neurones biologiques

Dans quels secteurs d'activité les rencontre-t-on ?

De par leur souplesse et leur capacité d'apprentissage, les réseaux de neurones sont utilisés dans des domaines aussi variés que l'intelligence artificielle (robots), les jeux vidéos, la reconnaissance de caractères, la synthèse vocale, l'analyse boursière, le traitement des images, bref, dans toute activité où l'environnement est par définition en constant changement.

2.2 Historique

Les premiers à proposer un modèle sont deux biophysiciens de Chicago, McCulloch et Pitts, qui inventent en 1943 le premier neurone formel qui portera leurs noms (neurone de McCulloch-Pitts).

Quelques années plus tard, en 1949, Hebb propose une formulation du mécanisme d'apprentissage [26], sous la forme d'une règle de modification des connexions synaptiques (règle de Hebb).

Le premier réseau de neurones artificiels apparaît en 1958, grâce aux travaux de Rosenblatt [27] qui conçoit le fameux perceptron. Le perceptron est inspiré du système visuel (en terme d'architecture neurobiologique) et possède une couche de neurones d'entrée ("perceptive") ainsi qu'une couche de neurones de sortie ("décisionnelle"). Ce réseau parvient à apprendre à identifier des formes simples et à calculer certaines fonctions logiques. Il constitue donc le premier système artificiel présentant une faculté jusque là réservée aux êtres vivants : la capacité d'apprendre par l'expérience.

Malgré tout l'enthousiasme que soulève le travail de Rosenblatt dans le début des années 60, la fin de cette décennie sera marquée en 1969, par une critique virulente dû perceptron par Minsky et Papert [28]. Ils utilisent une solide argumentation mathématique pour démontrer les limitations des réseaux de neurones à une seule couche. Ils montrent toutes les limites de ce modèle, et soulèvent particulièrement l'incapacité du perceptron à résoudre les problèmes non linéairement séparables, tel que le célèbre problème du XOR (OU exclusif). Il s'en suivra alors, face à la déception, une période noire d'une quinzaine d'années dans le domaine des réseaux de neurones artificiels.

Il faudra attendre le début des années 80 et le génie de Hopfield pour que l'intérêt pour ce domaine soit de nouveau présent. En effet, Hopfield démontre en 1982 [8] tout l'intérêt d'utiliser des réseaux récurrents (dits "feed-back"). Les réseaux récurrents constituent alors la deuxième grande classe de réseaux de neurones, avec les réseaux type perceptron (dits "feed-forward"). En parallèle des travaux de Hopfield, Werbos conçoit son algorithme de rétropropagation de l'erreur, qui offre un mécanisme d'apprentissage pour les réseaux multicouches de type perceptron (appelés PM pour Perceptron multicouche), fournissant ainsi un moyen simple d'entraîner les neurones des couches cachées. Notons que la thèse de Werbos, publiée en 1974 est restée longtemps sous silence. L'algorithme de rétropropagation ne sera pourtant popularisé qu'en 1986 par Rumelhart [29]. À partir de ce moment, la recherche sur les réseaux de neurones connaît un essor fulgurant et les applications commerciales de ce succès académique suivent au cours des années 90.

2.3 Neurone formel

Le premier neurone formel est apparu en 1943. On le doit à McCulloch et Pitts. Le neurone formel est une modélisation mathématique qui approxime les principes du fonctionnement du neurone biologique.

Interprétation mathématique

D'un point de vue mathématique, le neurone formel peut être représenté de la manière suivante :

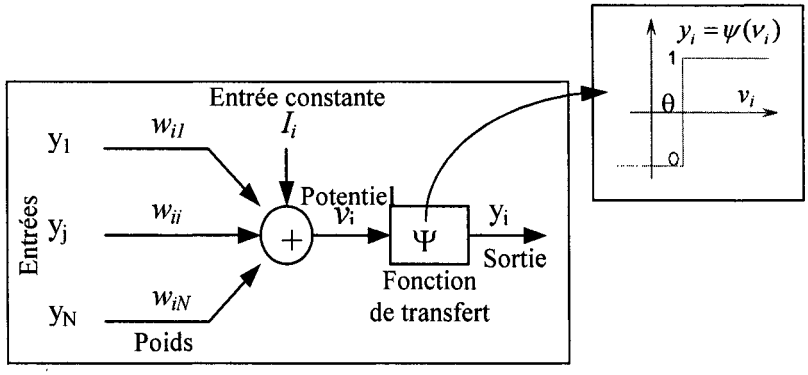


Figure 7 Modèle simplifié de neurone formel

Où v_i est le potentiel du neurone i, défini pour un réseau de N neurones mutuellement connectés comme suit :

$$v_i = \sum_{j=1}^{N} w_{ij} y_j + I_i \quad (2.1)$$

I_i : l'entrée directe du neurone i et

w_{ij} : le poids de connexion entre les neurones i et j

Le neurone formel donc va calculer la somme de ses entrées (y_1, ..., y_j,..., y_N, I_i), pondérées par les poids synaptiques (w_{i1}, ..., w_{iN}), et la comparer à son seuil teta (θ). Si le résultat est supérieur au seuil, alors la valeur renvoyée est 1, sinon la valeur renvoyée est 0.

D'où la sortie y_i du neurone i est :

$$y_i = \psi\left(\sum w_{ij} y_j + I_i - \theta\right) \quad (2.2)$$

avec ψ = fonction d'activation (ou fonction de seuillage) sert à introduire une non linéarité dans le fonctionnement du neurone.

2.4 Les approches neuronales et l'optimisation

L'ordonnancement est un problème d'allocation de ressources sujet à des contraintes. Autrement dit, c'est un problème d'optimisation dont l'objectif est d'assigner un nombre limité de ressources pour une période de temps définie de telle sorte que le coût de production soit minimal.

Depuis le début des années 1980, les technologies neuronales ont montré des potentialités intéressantes pour la résolution des problèmes d'optimisation. Ils présentent, dans ce domaine, deux atouts majeurs : le premier réside dans le fait que certains algorithmes neuronaux résolvent souvent très bien des problèmes d'optimisation; le second vient de ce que ces algorithmes sont particulièrement adaptés aux problèmes qui requièrent des temps de réponse extrêmement courts.

2.4.1 Les neurones formels utilisés pour l'optimisation

Dans les réseaux de neurones utilisés pour l'optimisation, les neurones sont soit binaires (leur fonction d'activation (Ψ) est un échelon entre –1 et +1), soit analogiques à fonction d'activation sigmoïde [4]. Dans ce dernier cas, la sortie y_i du neurone i est donnée par :

$y_i = \text{th}(\gamma v_i)$ (2.3)

où γ est la pente à l'origine de la sigmoïde.

La figure 8 montre des exemples de fonctions de transfert d'un neurone analogique pour différentes valeurs de γ (γ =50,10,5,3,2,1,0.5,0.2) [4].

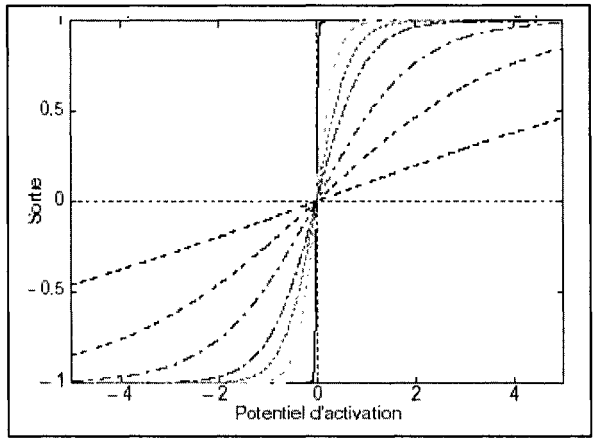


Figure 8 Fonction sigmoïde [4]

On note que la sigmoïde se rapproche de l'échelon lorsque γ augmente (figure 8)

2.4.2 Architectures de réseaux de neurones pour l'optimisation

L'architecture d'un réseau de neurones est définie par l'ensemble des neurones et des connexions entre eux.

Les **réseaux de neurones bouclés** (ou récurrents) constituent les techniques neuronales les plus fréquemment utilisées pour résoudre des problèmes d'optimisation combinatoire [4]. Ils se caractérisent par un graphe de connexions cyclique : lorsqu'on se déplace dans le réseau en suivant le sens des connexions, il est possible de trouver au moins un chemin qui revient à son point de départ (un tel chemin est désigné sous le terme de « cycle »). La sortie d'un neurone du réseau peut donc être fonction d'elle-même; cela n'est évidemment concevable que si la notion de *temps* est explicitement prise en considération [4].

2.5 Étude des différentes approches de RN utilisés dans l'optimisation

En 1994, Dagli [6] a présenté plusieurs types de réseaux neuronaux utilisés pour l'optimisation :

- réseau de recherche
- réseau probabiliste
- réseau de concurrence
- réseau correcteur d'erreurs

Dans un **réseau de recherche,** le réseau de neurone est installé de telle manière que la dynamique du réseau conduit fréquemment vers un minimum local qui représente une solution possible, qui n'est pas nécessairement proche de l'optimum ou bien qui ne correspond pas à une solution acceptable. Un exemple de ce type est l'approche de réseau de Hopfield.

Les **réseaux probabilistes** sont utilisés pour éviter d'être piégé trop facilement dans des minima locaux, ils sont basés sur les résultats probabilistes de la sortie de réseau. Un exemple de ce type est l'approche des machines de Boltzmann.

Dans le type de **réseaux de compétition**, la solution vient du gagnant de la compétition d'un groupe de neurones de sorties par des inhibitions latérales pour s'assurer que seulement un choix simple est fait. Un exemple de ce type est la carte auto-organisatrice de Kohonen.

Dans quelques cas complexes où beaucoup de contraintes sont présentes, les **réseaux correcteurs d'erreurs** sont utilisés pour apprendre les modèles de sortie des ordonnancements. Un exemple de ce type est l'approche populaire perceptron multicouche (PM).

Dans ce que suit on va s'intéresser seulement aux deux premiers types qui sont les plus utilisé dans l'ordonnancement des opérations manufacturières.

2.5.1 Les réseaux de neurones récurrents de Hopfield

Le réseau de Hopfield est sans doute le plus simple et le mieux connu théoriquement des réseaux de neurones. Il est constitué d'une couche de neurones complètement connectés (figure 9). Le modèle de Hopfield est un réseau non supervisé (auto-associatif) récurrent (voir définitions en annexe 2).

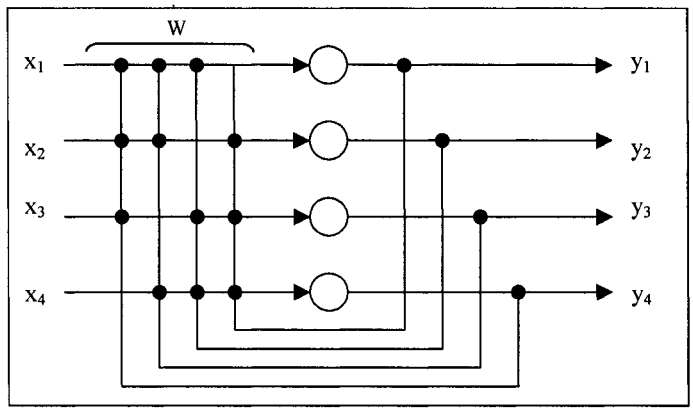


Figure 9 Architecture d'un réseau de Hopfield de 4 neurones

Avec :

W : matrice de poids,

X : vecteur d'entrée initiale et

Y : vecteur des sorties des neurones du réseau.

2.5.1.1 Principes de fonctionnement du réseau de Hopfield pour l'optimisation

Pour l'optimisation, on utilise le réseau de la manière suivante : à partir d'un état initial, on laisse le réseau évoluer librement jusqu'à un attracteur, qui est généralement, pour les problèmes d'optimisation, un état stable indépendant du temps (un point fixe de la

dynamique). On dit alors que le réseau a convergé : la convergence est atteinte lorsque les sorties des neurones n'évoluent plus.

La dynamique du réseau est généralement asynchrone : entre deux instants de temps, un seul neurone, choisi aléatoirement, est mis à jour ; autrement dit, son potentiel est calculé, et sa sortie réévaluée en conséquence.

Quand ces réseaux sont utilisés pour résoudre des problèmes d'optimisation, les poids des connexions sont déterminés analytiquement à partir de la formulation du problème ; en général, cela est fait directement à partir de la fonction énergie associée au problème. De plus, les sorties des neurones, dans l'attracteur vers lequel converge le réseau, codent une solution au problème d'optimisation [4].

2.5.1.2 Réseaux de Hopfield binaires

Le réseau de neurones proposé initialement par Hopfield en 1982 [8] est un réseau de neurones récurrent à temps discret, dont la matrice des connexions w_{ij} est symétrique et à diagonale nulle ($w_{ij}=w_{ji}$ et $w_{ii}=0$).
Le potentiel du neurone i au temps k est la somme pondérée de l'activité des autres neurones du réseau au temps k-1 :

$$v_i(k) = \sum_{j=1}^{N} w_{ij}\, y_j(k-1) + I_i \quad (2.4)$$

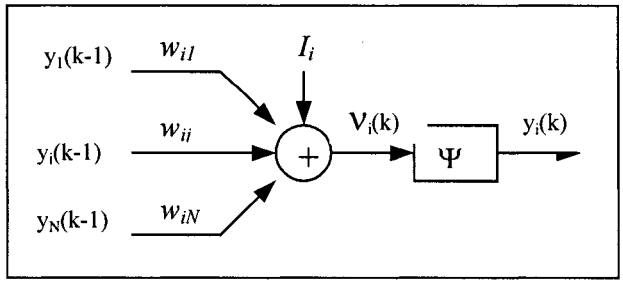


Figure 10 Neurone formel et notion de *temps*

avec $y_i(k)$ est la sortie du neurone i à l'instant k

$$y_i(k) = \psi(\sum w_{ij} y_j(k-1) + I_i - \theta)$$

I_i est l'entrée directe du neurone i,

w_{ij} est le poids de la connexion entre les neurones i et j.

Hopfield a utilisé une fonction d'énergie associée au réseau comme outil pour définir des réseaux récurrents et pour comprendre leur dynamique [8]. Les attracteurs vers lesquels converge un tel réseau correspondent aux minima de cette fonction énergie, définie par Hopfield comme suit :

$$E(y) = \underbrace{-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} y_i y_j}_{E1} \underbrace{-\sum_{i=1}^{N} I_i y_i}_{E2} \qquad (2.5)$$

Le premier terme de cette fonction (E1) est un indice de l'attraction relative qu'exerce une configuration possible des neurones. Elle est fonction de l'état de tous les neurones à un moment donné, et des coefficients synaptiques de leurs connexions.

Le deuxième terme (E2) permet d'estimer l'énergie qu'exercent les entrées constantes des différents neurones. Elle est fonction de l'état de tous les neurones à un moment donné et leurs biais.

Cette énergie E(y) est une fonction de N variables qui comporte généralement un très grand nombre de minima locaux.

2.5.1.3 Réseaux de Hopfield analogiques

En 1984, Hopfield [9] a proposé une version continue (dite analogique) des réseaux de neurones récursifs binaires, présentés dans le paragraphe précédent et en 1985 Hopfield et Tank [10] ont appliqué ce type de réseaux de neurones à l'optimisation combinatoire. Dans ce cas, la fonction d'énergie correspondante est de la forme :

$$E(y) = \underbrace{-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} y_i y_j}_{E1} \underbrace{-\sum_{i=1}^{N} I_i y_i}_{E2} + \underbrace{\frac{\alpha}{\gamma}\sum_{i=1}^{n}\int_0^{y_i} \Psi^{-1}(y)dy}_{E3} \qquad (2.6)$$

avec α est un réel positif,

γ est la pente à l'origine du sigmoïde,

Ψ est la fonction d'activation des neurones.

Les deux premiers termes de cette fonction (E1 et E2) représentent l'énergie des réseaux discrets. Le troisième terme (E3) tend à conduire les activations vers leur valeur de repos, Ψ (0). Elle est fonction de la fonction d'activation et l'état de tous les neurones à un moment donné.

En général, le dernier terme de cette énergie est négligeable comparé aux précédents dès lors que la pente γ à l'origine des neurones est grande ou encore que α est petit. Le système d'équations d'évolution contraint la fonction d'énergie E à décroître vers un minimum local (voir annexe 1 qui montre les propriétés de convergence des réseaux de Hopfield).

En pratique, avec une réalisation électronique d'un tel réseau, les temps de convergence sont de l'ordre de quelques nanosecondes ou microsecondes. Cela autorise en général plusieurs résolutions du problème en partant de différents points initiaux. Cette stratégie peut parfois se montrer efficace, mais en général les minima attracteurs ne sont pas de qualité suffisante.

2.5.1.4 Application des réseaux de Hopfield à l'optimisation

L'application de ce type de réseaux à la résolution de problèmes d'optimisation comprend les étapes suivantes :

Étape 1 : **Déterminer un codage du problème**

Il s'agit de trouver une représentation du problème d'optimisation telle qu'une solution, c'est-à-dire une instanciation des variables du problème, soit représentée par l'état des sorties des neurones à la convergence.

Cette première étape consiste à reformuler le problème d'un point de vue mathématique, afin qu'il soit possible de le résoudre par un réseau de neurones.

Par exemple, pour le problème d'ordonnancement, une solution au problème, c'est-à-dire un ordonnancement faisable, peut être codé comme une matrice de permutation, c'est-à-dire une matrice carrée à éléments binaires (Voir l'exemple dans le paragraphe 3.7.1.1 page 45).

Étape 2 : **Déterminer l'énergie du réseau de Hopfield**

Il existe une fonction de l'état, dite fonction d'énergie, ou fonction de Liapounov, qui décroît toujours pendant l'évolution libre du réseau ; les états stables de celui-ci sont donc des minima de cette fonction d'énergie.

Pour cela on cherche à exprimer la fonction de coût et les contraintes sous la forme d'une énergie d'un réseau de Hopfield. (Voir paragraphe 3.7.1.3 page 47)

Étape 3 : **Déterminer les équations d'évolution de chaque neurone**

Les interconnexions entre le neurone (i) et les autres neurones sont déterminées par l'équation de mouvement.

Le changement de l'état d'entrée du neurone (i) est donné par les dérivés partielles de la fonction d'énergie E. L'équation d'évolution [14] est donnée par :

$$\frac{du_i}{dt} = -\frac{\partial E(v_1, v_2, ..., v_N)}{\partial v_i} \qquad (2.7)$$

avec u_i : l'entrée du neurone i

v_i : la sortie du neurone i

Le but des réseaux de neurone pour résoudre les problèmes d'optimisation est de minimiser la fonction d'énergie dans l'équation (2.7). Il est habituellement plus facile d'établir l'équation d'évolution que la fonction d'énergie.

À partir de l'équation (2.7) la fonction d'énergie peut être obtenue :

$$E = \int dE = -\int \frac{du_i}{dt} dv_i \qquad (2.8)$$

Étape 4 : **Démarrer l'exécution du réseau de neurones**

Généralement, quand on ne dispose d'aucune connaissance à priori sur la localisation de la solution recherchée, les états des neurones sont initialisés aléatoirement. La dynamique du réseau de neurone, fondée sur les équations d'évolution des neurones, les fera converger vers un minimum local de l'énergie.

Ce type de méthodologie a été appliquée pour résoudre une grande variété de problèmes combinatoires. Et pour illustrer les propos qui précèdent, nous allons détailler la résolution d'un exemple concret (problème Job Shop) dans le chapitre 3 (paragraphe 3.7.2 page 54).

2.5.1.5 Limitations des réseaux de Hopfield

L'utilisation des réseaux de Hopfield pour résoudre des problèmes d'optimisation pose un certain nombre de problèmes. La principale difficulté, d'après Dreyfus et al. (2002) [4] , vient du fait que la dynamique du réseau conduit fréquemment vers un minimum local de l'énergie, qui n'est pas nécessairement proche de l'optimum, ou bien qui ne correspond pas à une solution acceptable. Car la fonction d'énergie exprimée en fonction des contraintes est combinée à la fonction de coût. De plus, le réseau de Hopfield ne fait aucune distinction entre les contraintes strictes (celles qui sont définies par le problème d'optimisation) et les contraintes de préférence (celles qui résultent du codage du problème), autrement que par des pondérations différentes dans la fonction d'énergie. Ces difficultés ont été à l'origine de nombreux travaux qui ont cherché à limiter les effets.

2.5.2 La machine de Boltzmann (MB)

La machine de Boltzmann a été introduite en 1984 par Ackley, Seijnowski et Hinton [22]. Ce réseau est une extension originale de Hopfield. C'est un réseau récurrent mais qui a la particularité d'inclure une couche cachée (figure 11). L'idée des créateurs du réseau est d'essayer de résoudre les problèmes du réseau de Hopfield qui est la sensibilité aux minima locaux ou l'évolution vers des états poubelles (solutions non acceptables).

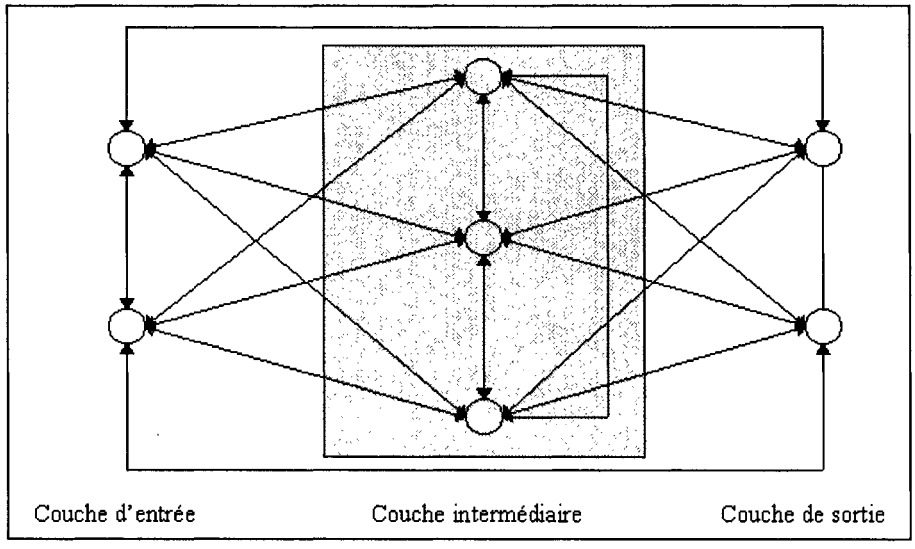


Figure 11 Architecture d'un réseau de Boltzmann

La machine de Boltzmann peut être considérée comme une combinaison des principes du **recuit simulé** avec des **réseaux de Hopfield binaires**.

2.5.2.1 Le principe de recuit simulé [4]

Le but du recuit dans cet algorithme est d'éviter les minimums locaux au profit d'un minimum plus global.

Cette technique est similaire à celle utilisée en sidérurgie pour refroidir le métal. En effet lors du refroidissement de l'acier, les atomes s'organisent en structures cristallines. Si la vitesse de refroidissement est rapide, le métal sera fragile. Pour éviter de fragiliser le métal, on effectue un refroidissement par palier qui permet d'éviter les contraintes imposées à un refroidissement rapide. On descend donc la température, puis on attend un certain temps avant de refroidir encore. Le nombre de paliers et leurs paramètres (température, durée) correspond à un programme de recuit.

Donc l'idée de l'algorithme de recuit simulé est la suivante : à des paliers de températures décroissantes, l'algorithme utilise une procédure itérative, développée par Metropolis en 1953 [23], pour atteindre un état de quasi-équilibre thermodynamique. Cette procédure permet de sortir des minima locaux avec une probabilité d'autant plus grande que la température est élevée. Quand l'algorithme atteint les très basses températures, les états les plus probables constituent d'excellentes solutions du problème d'optimisation.

2.5.2.2 Application des réseaux de Boltzmann à l'ordonnancement

Pour appliquer la machine de Boltzmann à l'ordonnancement, on peut utiliser le même codage que celui utilisé pour les réseaux de Hopfield. Les sorties binaires du réseau codent une solution au problème. Les énergies associées à la fonction de coût à minimiser et aux contraintes sont exactement les mêmes que celles utilisées dans un réseau de Hopfield binaire. Les sorties des neurones sont initialisées aléatoirement avec les valeurs 0 ou 1. L'état initial ne code donc pas nécessairement une solution acceptable au problème.

Dans tout ce qui précède, on a travaillé exactement de la même manière qu'avec un réseau de Hopfield. La mise à jour des neurones est cependant différente. Elle consiste tout d'abord à calculer la variation d'énergie induite par son changement d'état. Ainsi, la variation d'énergie associée au passage de 0 à 1 de la sortie d'un neurone est donnée par :

$$\Delta E_i = E_{y_i=1} - E_{y_i=0} = -\sum_{j=1 \& j \neq i}^{N} w_{ij} y_j \qquad (2.9)$$

Ces valeurs dépendent de l'état courant des autres neurones du réseau. La probabilité (P_i) de mettre la sortie du neurone i à 1 est calculée à partir de ΔE_i et de la température selon la formule utilisée en recuit simulé :

$$P_i = \begin{cases} 1 & \text{si } \Delta E_i \leq 0 \\ \exp\left(-\frac{\Delta E_i}{T}\right) & \text{sinon} \end{cases}$$

Si la probabilité vaut 1, la sortie du neurone *i* est mise à 1. Sinon, on tire un nombre aléatoire entre 0 et 1. Si celui-ci est inférieur à la probabilité calculée, alors la sortie du neurone *i* est mise à 1 ; dans le cas contraire, elle est mise à 0. Cette procédure de mise à jour des neurones est itérée un grand nombre de fois, de manière à s'approcher de l'équilibre thermodynamique à la température T.

La procédure que l'on vient de spécifier est appliquée sur des paliers de températures décroissantes, comme en recuit simulé. Le réseau a convergé quand le nombre de changements d'état de neurones est réduit au minimum.

Ce qui vient d'être énoncé établit donc un lien étroit entre le recuit simulé, les réseaux de neurones récurrents de Hopfield et les machines de Boltzmann.

2.5.2.3 Limitations des réseaux de Boltzmann

Becker [24] et Downs [25] ont montré que dans la pratique, les réseaux de Boltzmann ont le gros problème d'être très lents et peuvent nécessiter des capacités de mémoire importantes. L'application des réseaux de Boltzmann dans les problèmes d'ordonnancement est limitée seulement pour le problème du voyageur de commerce, les problèmes d'ordonnancement des opérations manufacturières ne sont pas encore entamés.

2.6 Le choix d'une technique neuronale pour l'ordonnancement

Les limitations et les considérations développées dans les sections précédentes permettent d'effectuer des choix en fonction :

- de la complexité du problème : nombre de variables, nombre de contraintes, type de coûts à minimiser ;
- des exigences sur la qualité de la solution : si par exemple, les données du problème à résoudre sont entachées de bruit, il n'est pas nécessaire de chercher à s'approcher d'aussi près que possible de l'optimum ;
- du temps de développement disponible : temps de calcul pour évaluer une solution potentielle.

Face à un problème d'optimisation de l'ordonnancement, la comparaison des performances des différentes approches entre elles est délicate. Si l'on est exigeant sur la qualité de la solution, les réseaux de Boltzmann sont plus performants, mais souffrent de limitations dues à leur mise au point délicate. De plus, ces algorithmes sont lents et peuvent nécessiter des capacités de mémoire importantes [24 et 25]. Si on est moins exigeant sur la qualité de la solution et l'on souhaite un temps de développement moins long, les réseaux de Hopfield analogiques sont plus adaptés.

2.7 Vers une approche hybride

Dans la recherche des meilleurs algorithmes capables de résoudre les problèmes d'ordonnancement JS et PC, une tendance actuelle veut que l'on bâtisse une méthode simple et rapide adaptée aux problèmes de toutes tailles.

Pour réduire les temps de résolution tout en produisant d'excellentes solutions, une approche hybride est souvent le meilleur choix. Et puisque les Réseaux de Neurones de Hopfield (RNH) sont davantage adaptés aux problèmes où le temps de résolution l'emporte sur les exigences en termes de qualité de solution, on a décidé de le combiner avec une heuristique permettant de raffiner la qualité de solution pour résoudre nos problèmes d'ordonnancement JS et PC. Cette nouvelle approche est présentée dans le chapitre qui suit.

CHAPITRE 3

L'ORDONNANCEMENT AVEC LE RESEAU DE HOPFIELD HYBRIDE

3.1 Introduction

Ce chapitre présente l'application d'une approche hybride qui combine les réseaux de neurones, ainsi qu'une heuristique, pour résoudre le problème d'ordonnancement JS et PC, des problèmes NP-complets. Le réseau de neurone de Hopfield proposé et l'heuristique qui peut être combinée avec le réseau neuronal sont présentés. En approches combinées, RNH est utilisé pour obtenir les solutions faisables, l'algorithme heuristique est utilisé comme un outil de recherche local pour améliorer la qualité des solutions obtenues.

3.2 Problématique

Soit un ensemble M de ***m*** machines et un ensemble N de ***n*** tâches à effectuer. Chaque tâche peut être décomposée en plusieurs opérations.

Dans un problème d'ordonnancement classique, il existe une relation d'ordre entre les différentes opérations et/ou entre les différentes tâches. Si l'on s'intéresse par exemple à une chaîne de production, l'emballage des produits finis ne peut pas avoir lieu avant que tous les traitements aient été effectués sur le produit en question. On a donc une relation de précédence entre les opérations.

Le problème d'ordonnancement dans un environnement Job Shop ou production cellulaire est complexe, non seulement par le nombre de possibilités de combinaisons de séquences de tâches, mais aussi par le fait que l'efficacité des machines est différente en fonction des opérations effectuer. Le temps des opérations des tâches sur chaque

machine est différent et la priorité varie d'une tâche à l'autre. Des exemples d'ordonnancement job shop et production cellulaire sont détaillés dans les paragraphes 3.5 et 3.6.

3.3 Objectif

La résolution d'un problème d'ordonnancement consiste donc d'une part à affecter toutes les opérations à des machines pouvant les exécuter et d'autre part à leur donner une date de début qui permet de respecter l'ensemble des contraintes. Bien évidemment ces deux parties ne peuvent pas être dissociées l'une de l'autre.

On cherche ensuite à optimiser un critère qui peut être de plusieurs natures, comme nous le verrons dans le paragraphe qui suit. De manière générale, on souhaite rentabiliser au mieux son affaire.

3.4 Critères à optimiser

Il n'est pas facile de statuer sur nos objectifs de l'ordonnancement. Il faut mentionner que du point de vue de l'optimisation, de nombreux critères peuvent être optimisés. Alors nous indiquerons dans des termes généraux quelques critères [34].

3.4.1 Critères basés sur le temps d'achèvement de la tâche

Les principaux critères rattachés à cette catégorie sont :

- C_{max} : la date de fin du projet "*makespan*"
- F_{total} : le temps de passage total des tâches dans l'atelier
- C_{moyen} : le temps moyen nécessaire pour l'achèvement des toutes les tâches
- F_{moyen} : le temps de passage moyen d'une tâche dans l'atelier

La minimisation de C_{max} implique essentiellement la minimisation des coûts de l'ordonnancement lequel dépend de combien de temps le système a consacré pour traiter la tâche la plus longue. La minimisation de F_{total} implique la minimisation du coût de l'ordonnancement apparenté directement au temps nécessaire pour le traitement de toutes les tâches. La minimisation du temps moyen de passage F_{moyen} implique que le coût de l'ordonnancement est directement lié au temps moyen nécessaire pour le traitement de la tâche. La minimisation du temps moyen d'achèvement du traitement d'une tâche donnée, C_{moyen}, est équivalent à F_{moyen}.

Parfois nous considérons les mesures pondérées. Nous reconnaissons que certaines tâches sont plus importantes que d'autres. Alors nous suggérons de minimiser la somme pondérée telle que :

- $CO_{pondéré}$: Coût total pondérés de l'ordonnancement
- $CR_{pondéré}$: Coût total pondérés des retards

3.4.2 Critères basés sur les délais de livraison :

Les principaux critères rattachés à cette catégorie sont :

- la somme des retards par rapport à des dates de fin désirées
- retard moyen des tâches
- Durée moyenne des retards
- la somme des avances

Il y a des pénalités encourues quand la tâche est en retard ou en avance. L'objectif raisonnable sera de minimiser :

- le coût moyen des retards
- le coût moyen des avances

3.4.3 Critères basés sur les coûts d'inventaire

Ici, nous désirons minimiser :

- le nombre moyen de tâches en attente devant les machines
- le nombre moyen des tâches inachevées

3.4.4 Critères basés sur les coûts d'utilisation

Ici nous choisissons de minimiser :

- le nombre moyen des tâches actuellement prêtes à être traitées à n'importe quel instant
- la somme des temps morts par machine

Dans notre travail, nous intéressons au critère le plus courant, qui est la date de fin du projet "*makespan*". Afin de comparer nos résultats avec des exemples pris de la littérature. La majorité des articles et travaux étudiés ont utilisé le makespan [11], [12], [15]. La méthode la plus célèbre qui tient compte de ce critère est celle de Johnson. C'est un heuristique développé pour deux et trois processeurs seulement. L'hypothèse le plus restrictive imposé par cette méthode est que l'ordre de passage des commandes doivent être traitées par deux postes de travail successifs. Chaque commande doit d'abord être traitée par le premier poste, puis par le second, et non inversement.

3.5 Définition du problème d'ordonnancement job shop

L'environnement "Job shop" est caractérisé par la fabrication des pièces qui différent en terme de besoins de transformation, de matériels, de temps de fabrication, de séquence de traitement et de mise en route. Le problème consiste ainsi à trouver la séquence des tâches qui minimise le makespan correspondant au temps de fin de la dernière opération dans l'ordonnancement.

Le tableau 3 présente un exemple d'ordonnancement job shop à quatre tâches (J1, J2, J3, J4) et 3 machines (M1, M2, M3). Les opérations de chaque tâche devraient être faites dans l'ordre donné par le tableau. La tâche J1 se compose de trois opérations (O1, O2 et O3), l'opération O1 devrait être faite par la machine M1 et nécessite un temps d'exécution de k1 unités puis l'opération O2 doit être faite par M2 et prend k2 unités de temps, etc.

Tableau III

Exemple d'ordonnancement Job shop

Tâche	**(Opération) (Machine) (Temps)**		
J1	(O1) (M1) (k1)	(O2) (M2) (k2)	(O3) (M3) (k3)
J2	(O4) (M3) (k4)	(O5) (M1) (k5)	(O6) (M2) (k6)
J3	(O7) (M1) (k7)	(O8) (M3) (k8)	(O9) (M2) (k9)
J4	(O10) (M2) (k10)	(O11) (M3) (k11)	(O12) (M1) (k12)

3.6 Définition du problème d'ordonnancement production cellulaire

Dans notre problème d'ordonnancement, on suppose que la cellule pourrait seulement produire certains types de produits et un flux harmonieux de matériaux et de composants est obtenu tout au long du processus avec des temps de transport et d'encours minimums. Le problème consiste ainsi à trouver la séquence des différentes familles de produits qui minimise le Makespan.

Le tableau 4 présente un exemple d'ordonnancement de production cellulaire à quatre familles de produits et 9 machines. Les opérations de chaque famille de produits devraient être faites dans l'ordre donné par le tableau. La famille 1 se compose de quatre opérations (O1, O2, O3 et O4), l'opération O1 devrait être faite par la machine M1 et nécessite un temps d'exécution de k1 unités puis l'opération O2 doit être faite par M2 et prend k2 unités de temps, etc.

Tableau IV

Exemple d'ordonnancement de la production cellulaire

Tâche	(Opération) (Machine) (Temps)			
Famille 1	(O1) (M1) (k1)	(O2) (M2) (k2)	(O3) (M3) (k3)	(O4) (M8) (k4)
Famille 2	(O5) (M4) (k5)	(O6) (M5) (k6)	(O7) (M6) (k7)	(O8) (M8) (k8)
Famille 3	(O9) (M1) (k9)	(O10) (M3) (k10)	(O11) (M8) (k11)	
Famille 4	(O26) (M7) (k12)	(O27) (M8) (k13)	(O28) (M9) (k14)	

3.7 Nouvelle approche hybride pour l'ordonnancement

En 1988 Foo et Takefuji [12, 13] ont présenté une approche très influente en utilisant le réseau de neurones de Hopfield pour résoudre le problème Job Shop. Mais d'après Xin-li et Wan-Liang [15], leur méthode ne peut pas garder les sorties régulières de réseau de neurones en tant que solution faisable et ne peut pas garantir l'optimalité.

Le travail qui suit présente une approche hybride pour le problème d'ordonnancement Job Shop et Production Cellulaire (Figure 12). Cette nouvelle approche combine le réseau de neurones de Hopfield et une procédure de recherche locale de Nowicki et de Smutnicki (1996) [21].

L'approche consiste en deux phases principales :

- ***Assignation des priorités et des temps de début des opérations*** : cette phase se sert de réseau de Hopfield pour construire des solutions faisables de l'ordonnancement et définir le diagramme de Gantt.

- ***Procédure de recherche locale*** : cette phase se sert de la procédure de recherche taboue de Nowicki et de Smutnicki (1996) pour raffiner la solution obtenue par le réseau de Hopfield.

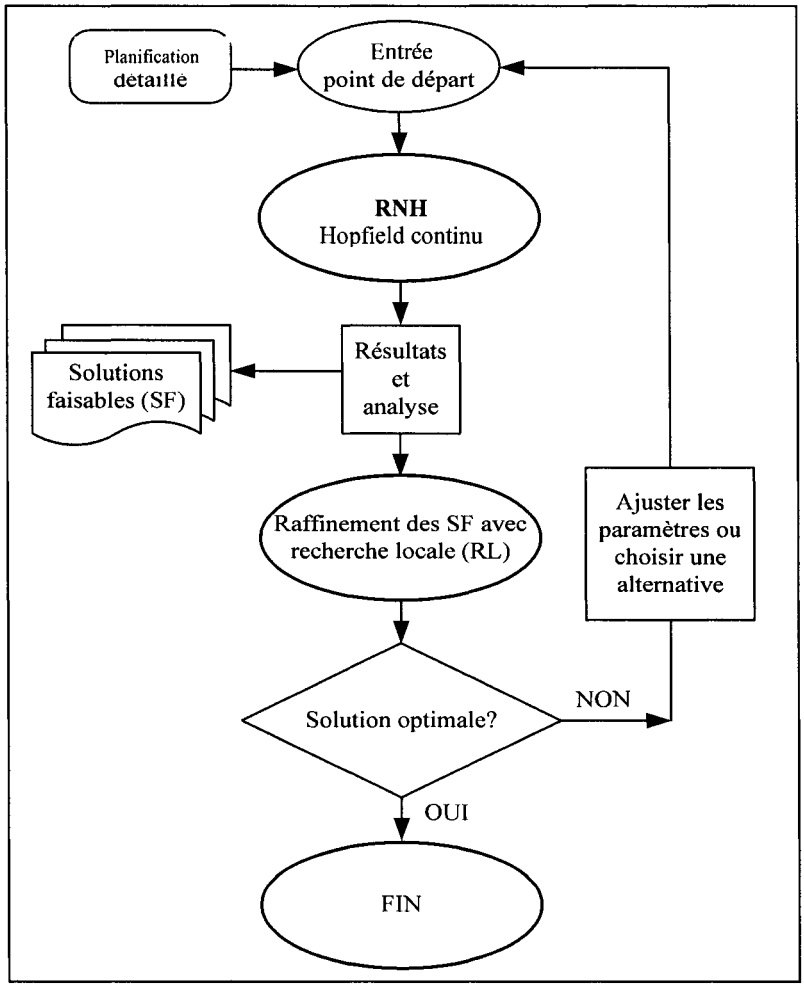


Figure 12 Organigramme de la nouvelle approche pour l'ordonnancement

3.7.1 Caractéristiques du RNH pour l'ordonnancement

Ayer et al. (1990) [16] ont montré qu'un réseau continu de Hopfield est bien plus rapide et fiable qu'un réseau de neurones binaires (voir paragraphe 2.5.1.2 et 2.5.1.3), car les vallées de l'espace des solutions dans le cas continu sont plus larges que dans le cas binaire. Pour cela, on va utiliser Hopfield continu dans notre travail.

3.7.1.1 Codage du problème

Il s'agit, dans un premier temps, de trouver la représentation du problème d'ordonnancement telle qu'une solution soit représentée par l'état des sorties des neurones à la convergence.

En utilisant le réseau de neurones de Hopfield pour résoudre le problème Job Shop ou production cellulaire, une matrice de permutation avec (N) lignes et (N+1) colonnes est généralement appliquée pour le représenter (où N est le nombre total des opérations) (Figure 13). La valeur des éléments g_{ij} dans la matrice est seulement 1 ou 0. Si g_{ij} est marqué 1 alors l'opération O_i dépend de l'opération O_j, sinon l'opération O_i ne dépend pas de l'opération O_j. La première colonne dans la matrice fournit l'hypothèse qu'une opération O_i ne dépend d'aucune autre opération (elle peut commencer au temps 0).

	0	O_1	O_2	...	O_j	...	O_N
O_1	g_{10}	g_{11}	g_{12}	...	g_{1j}	...	g_{1N}
O_2	g_{20}	g_{21}	...	...	...	...	g_{2N}
:	:	:	:	:	:	:	:
O_i	g_{j0}	:	:	:	g_{ij}	:	:
:	:	:	:	:	:	:	:
O_N	g_{N0}	g_{N1}	...	...	g_{Nj}	...	g_{NN}

Figure 13 Organisation des neurones basés sur une représentation matricielle

Ainsi, en se basant sur cette représentation, le nombre total des neurones nécessaires pour représenter un problème job shop ou production cellulaire est T=N(N+1).

3.7.1.2 Contraintes à satisfaire

Quant aux contraintes à satisfaire, elles sont de deux types, celles qui sont définies par le problème d'optimisation, et celles qui résultent du codage du problème.
Il y a quatre contraintes de base à coder dans le réseau de neurones qui sont définies par le problème d'optimisation :

(1) Le plus petit entre les nombres n et m des tâches devrait commencer au temps t=0, afin d'éviter les temps morts (*idle time*) des machines au temps t=0, (avec ***m*** le nombre de machines et ***n*** le nombre de tâches à effectuer)
(2) L'autodépendance sur chaque opération n'est pas permise
(3) La dépendance mutuelle de l'un sur l'autre n'est pas également permise
(4) Les relations de précédence entre les opérations doivent être respectées

Il y a 2 autres contraintes supplémentaires qui résultent du codage du problème :

(5) On permet à chaque opération de dépendre seulement d'une autre opération. Donc on s'attend à ce que seulement un neurone s'allume à n'importe quelle ligne de la matrice,
(6) On permet qu'exactement N neurones doivent être allumés. En effet, sans cette contrainte supplémentaire, les contraintes définies ci-dessus ne sont pas encore suffisantes pour garantir la validité d'une solution, car le processus de minimisation ferait naturellement converger toutes les sorties des neurones vers 0, ce qui n'aurait aucun sens.

3.7.1.3 Détermination de l'énergie du réseau

Il existe une fonction d'état, dite fonction d'énergie, ou fonction de Liapounov, qui décroît toujours pendant l'évolution libre du réseau de neurones de Hopfield; les états stables de celui-ci sont donc des minima de cette fonction d'énergie.
On peut inverser le problème : supposons que l'on se pose un problème d'optimisation combinatoire d'ordonnancement, alors les points fixes de la dynamique de ce réseau constituent des solutions au problème d'optimisation combinatoire que l'on cherche à résoudre.

Si l'on peut construire un tel réseau, il peut alors trouver de lui-même, à partir d'un état initial quelconque, une solution au problème d'optimisation. La mise en oeuvre de cette idée nécessite donc de trouver la fonction d'énergie. Les démarches qu'il faut suivre pour déterminer cette énergie sont les suivantes :

Les contraintes (1), (2) et (4) sont imposées par l'entrée constante des neurones. Dans la contrainte du type (1), si n>m, il est nécessaire que le nombre m des tâches devrait commencer au temps t=0, sinon le nombre n des tâches devrait commencer à t=0. Autrement dit, il faut mettre autant de machines que possible en exécution à t=0 afin de chercher un optimal. Cette condition est imposée en plaçant aléatoirement de la polarisation positive dans la matrice aux neurones appropriés de telle sorte que ces neurones déclencheront facilement. Prenant l'exemple job shop à 3 machines (m=3) et 4 tâches (n=4) présentés dans le tableau III, on peut placer des inhibitions faibles pour $I_{1,0}$, $I_{4,0}$ et $I_{10,0}$ (voir figure 14).

La contrainte du type (2) est nécessaire pour éviter l'auto dépendance sur chaque opération, cette condition est imposée en plaçant des inhibitions (pondération négative) aux neurones appropriés de telle sorte que ces neurones ne déclencheront pas.
Pour le même exemple présenté dans le tableau III, on peut voir des inhibitions fortes sur le diagonale de la matrice des entrées constantes (I_{11}, I_{12} ... I_{NN}) (voir figure 14).

La contrainte (4) est importante pour que les relations de précédence entre les opérations ne soient pas violées. Encore, des poids inhibiteurs forts sont appliqués aux neurones appropriés pour imposer les relations de priorité d'opération.

I_{ij} =

10	-10	-10	-10	0	0	0	0	0	0	0	0	0
-10	0	-10	-10	0	0	0	0	0	0	0	0	0
-10	0	0	-10	0	0	0	0	0	0	0	0	0
10	0	0	0	-10	-10	-10	0	0	0	0	0	0
-10	0	0	0	0	-10	-10	0	0	0	0	0	0
-10	0	0	0	0	0	-10	0	0	0	0	0	0
-10	0	0	0	0	0	0	-10	-10	-10	0	0	0
-10	0	0	0	0	0	0	0	-10	-10	0	0	0
-10	0	0	0	0	0	0	0	0	-10	0	0	0
10	0	0	0	0	0	0	0	0	0	-10	-10	-10
-10	0	0	0	0	0	0	0	0	0	0	-10	-10
-10	0	0	0	0	0	0	0	0	0	0	0	-10

Figure 14 Représentation matricielle des entrées constantes des neurones

Les contraintes (3), (5) et (6) sont imposées par une fonction d'énergie E qui décrit le réseau de neurones dont le minimum correspond à une solution optimale de l'ordonnancement. Dans le paragraphe suivant on va définir cette fonction d'énergie.

En se basant sur le travail présenté par Foo et Takefuji en 1988 [12], l'énergie qui permet d'imposer la contrainte (5) est :

$$E1=\frac{1}{2}\sum_{x=1}^{N}\sum_{i=1}^{N+1}\sum_{j=1\&j\neq i}^{N+1}v_{xi}v_{xj} \qquad (3.1)$$

v_{xi} est la sortie du neurone en position (x, i) de la matrice.

Cette énergie (E1) est zéro si et seulement si chaque ligne dans la matrice des neurones ne contiennent pas plus d'un neurone allumé.

L'énergie qui permet d'imposer la contrainte (6) qu'exactement N neurones soient dans l'état allumé est :

$$E2=\frac{1}{2}\left(\sum_{x=1}^{N}\sum_{i=1}^{N+1}v_{xi}-N\right)^2 \qquad (3.2)$$

Cette énergie (E2) est zéro si et seulement s'il y a N neurones allumés dans la matrice entière.

La contrainte du type (3) est nécessaire pour éviter la dépendance mutuelle de l'un sur l'autre. Cette contrainte est considérée comme une inhibition asymétrique de la matrice. En 2002 Xu Xin-li et Wang Wan-liang [15] ont introduit une énergie qui permet d'imposer la connexion asymétrique :

$$E3=\frac{1}{2}\sum_{x=2}^{N}\sum_{i=2\&i\neq x+1}^{N}v_{xi}v_{(i-1,x+1)} \qquad (3.3)$$

Cette énergie (E3) est zéro si et seulement si les neurones (i, j) et (j-1, i+1) (i=2, 3,..., N, j=2,3,..., N) ne peuvent pas simultanément être allumés.

Donc l'énergie totale du réseau de Hopfield est la somme pondérée des fonctions définies ci-dessus :

E=A * E1 + B * E2 + C * E3

$$E=\frac{A}{2}\sum_{x=1}^{N}\sum_{i=1}^{N+1}\sum_{j=1\&j\neq i}^{N+1}v_{xi}v_{xj}+\frac{B}{2}\left(\sum_{x=1}^{N}\sum_{i=1}^{N+1}v_{xi}-N\right)^2+\frac{C}{2}\sum_{x=2}^{N}\sum_{i=2\&i\neq x+1}^{N}v_{xi}v_{(i-1)(x+1)} \qquad (3.4)$$

Tel que la constante :

A est le coefficient de connexions inhibitrices dans chaque ligne

B est le coefficient de l'inhibition globale

C est le coefficient de la connexion asymétrique

Les trois coefficients A, B et C sont des constantes positives arbitraires qui doivent être ajustées en fonction des poids relatifs des différentes contraintes. En outre, ces paramètres, suggérés aléatoirement dans la première simulation, ne sont pas nécessairement optimaux et souvent donnent des ordonnancements invalides. En géneral, des ordonnancements non acceptables peuvent être classifiés en trois types :

1- il y a plusieurs ou aucun neurone activé dans une rangée ou des rangées données de la matrice de sortie,
2- il y a plus ou moins que N neurones activés dans la matrice de sortie,
3- Il y a des neurones activés qui sont symétriques par rapport à la diagonale de la matrice.

Après la convergence, si on obtient une solution non acceptable (elle viole une ou plusieurs contraintes), on peut redémarrer le réseau en ajustant les coefficients (A, B et C), par exemple si la première contrainte est violée(il y a plusieurs ou aucun neurone activé dans une rangée ou des rangées données de la matrice), alors on augmente la valeur du coefficient (A) pour que la fonction d'énergie tente d'imposer seulement un 1 dans chaque rangée de la matrice de permutation, et de même pour les deux autres types de contraintes.

3.7.1.4 Détermination des équations d'évolution des neurones

Le but des réseaux de neurone pour résoudre les problèmes d'optimisation est de minimiser la fonction d'énergie. Puisqu'il est habituellement plus facile d'établir l'équation d'évolution que la fonction d'énergie, on va déterminer une équation linéaire

de l'évolution qui décrit le comportement du réseau de Hopfield en suivant le développement suivant :

on peut réécrire la fonction d'énergie quadratique du problème sous la forme d'une fonction d'énergie d'un réseau de Hopfield définie en 1985 [10] par :

$$E=-\frac{1}{2}\sum_{x=1}^{N}\sum_{i=1}^{N+1}\sum_{y=1}^{N}\sum_{j=1}^{N+1}w_{xi,yj}v_{xi}v_{yj}-\sum_{x=1}^{N}\sum_{i=1}^{N+1}v_{xi}I_{xi} \qquad (3.5)$$

Dans cette équation, on détermine les poids $w_{ij,kl}$ à partir de la forme analytique des contraintes, en égalisant les termes entre (3.4) et (3.5).
Ainsi, si l'on considère la contrainte E1, sa contribution aux coefficients synaptiques s'écrit :

$$-\delta_{xy}(1-\delta_{ij})$$

où δ est l'opérateur de Kronecker :

$$\delta_{ij}=\begin{cases}1 \text{ si } i=j\\ 0 \text{ si } i\neq j\end{cases}$$

De même, la contribution de la contrainte E2 est –1.

Enfin, la contribution de la contrainte E3 est :

$$-\delta_{y(i-1)}\delta_{j(x+1)}(1-\delta_{i1})(1-\delta_{x1})(1-\delta_{xy})(1-\delta_{ij})$$

La forme finale des poids de connexion entre les neurones (x,i) et (y,j) est donc :

$$w_{xi,yj}=-A\delta_{xy}(1-\delta_{ij})-B-C\delta_{y(i-1)}\delta_{j(x+1)}(1-\delta_{i1})(1-\delta_{x1})(1-\delta_{xy})(1-\delta_{ij}) \qquad (3.6)$$

La valeur d'entrée externe du neurone en position (x, i) de la matrice vaut :

$I_{xi} = C * N$

Dans le cas de neurones analogiques (continus) Hopfield et Tank (1985) [10] ont défini l'équation linéaire d'évolution, décrivant le comportement du réseau, comme suit :

$$\frac{du_{xi}}{dt} = -u_{xi} + \sum_{y=1 \& y \neq x}^{N} \sum_{j=1 \& j \neq i}^{N+1} w_{ij,yj} v_{yj} + I_{xi} \qquad (3.7)$$

Alors d'après (3.6) et (3.7) l'équation linéaire d'évolution, décrivant le comportement du réseau de Hopfield continu, peut être obtenu comme suit :

$$\frac{du_{xi}}{dt} = -u_{xi} - A \sum_{j=1 \& j \neq i}^{N} v_{xj} - B \sum_{x=1}^{N} \sum_{j=1}^{N+1} (v_{xj} - N) - C v_{(i-1)(x+1)} (1-\delta_{i1})(1-\delta_{x1})(1-\delta_{x(i-1)})(1-\delta_{i(x+1)}) + I_{xi} \qquad (3.8)$$

Cette équation d'évolution va nous servir pour utiliser la méthode d'Euler de premier ordre dans l'algorithme présenté dans la section 3.7.1.6 page 53.

3.7.1.5 La fonction sigmoïde d'entrée-sortie

Dans le paragraphe 2.4.1, on a défini la fonction d'activation sigmoïde du neurone *i* par : $v_i = g(u_i) = \tanh(\lambda * u_i)$ qui a une sortie entre –1 et 1. Mais dans ce qui suit on a préféré utiliser des neurones à sortie continue entre 0 et 1. donc on obtient :

$$v_{ij} = g(u_{ij}) = \frac{1}{2}(1 + \tanh(\lambda * u_{ij})) \qquad (3.9)$$

3.7.1.6 Description de l'algorithme

La procédure suivante décrit l'algorithme proposé en se basant sur la méthode d'Euler de premier ordre. C'est la méthode utilisée pour approximer les solutions $Uij(t+1)$. Il s'agit certainement de la méthode la plus simple d'intégration numérique [19].

1. Choisir aléatoirement les valeurs initiales de $u_{ij}(t)$, avec i=1,...,N et j=1,...,N+1
2. Mettre t =0
3. Évaluer les valeurs de $v_{ij}(t)$ en se basant sur la fonction sigmoïde définie par :

$$v_{ij}(t)=g(u_{ij}(t))=\frac{1}{2}(1+\tanh(\lambda * u_{ij}(t)))$$

4. Utiliser l'équation de mouvement (3.8) pour calculer $\Delta u_{ij}(t)$:

$$\Delta u_{ij}(t)=-u_{xi}-A\sum_{j\neq i} v_{xj}(t)-B\sum_{x}\sum_{j}(v_{xj}(t)-N)-Cv_{(i-1)(x+1)}(t)(1-\delta_{i1})(1-\delta_{x1})(1-\delta_{x(i-1)})(1-\delta_{i(x+1)})+Ixi$$

5. Calculer $Uij(t+1)$ en se basant sur la méthode d'Euler de premier ordre :

$$Uij(t+1)=Uij(t)+\Delta Uij(t)$$

6. Utiliser la fonction d'énergie globale (3.4) pour calculer l'énergie E(t) :

$$E(t)=\frac{A}{2}\sum_{x=1}^{N}\sum_{i=1}^{N+1}\sum_{j=1\&j\neq i}^{N+1}v_{xi}(t)v_{xj}(t)+\frac{B}{2}\left(\sum_{x=1}^{N}\sum_{i=1}^{N+1}v_{xi}(t)-N\right)^2+\frac{C}{2}\sum_{x=2}^{N}\sum_{i=2\&i\neq x+1}^{N}v_{xi}(t)v_{(i-1)(x+1)}(t)$$

7. Incrémenter t par le pas Δt
8. Si t=T terminer la procédure, sinon retourner à l'étape 2.

3.7.2 Mise en œuvre de la première phase de l'approche proposée

Pour illustrer les propos qui précèdent à l'aide d'un petit exemple concret, nous allons détailler la résolution d'un problème job shop, proposée par Xin-li , Wan-liang (2001) [15]

- **Étape 1** : Données du problème

Nous allons examinés un exemple d'ordonnancement JS à quatre tâches (J1, J2, J3, J4) et 3 machines (M1, M2, M3). Les opérations de chaque tâche devraient être faites dans l'ordre donné par le tableau IV.

Tableau V

Exemple d'ordonnancement JS (12 opérations)

Tâche	(Opération) (Machine) (Temps)		
J1	(O1) (M1) (5)	(O2) (M2) (8)	(O3) (M3) (2)
J2	(O4) (M3) (7)	(O5) (M1) (3)	(O6) (M2) (9)
J3	(O7) (M1) (1)	(O8) (M3) (7)	(O9) (M2) (10)
J4	(O10) (M2) (4)	(O11) (M3) (11)	(O12) (M1) (7)

- **Étape 2** : Codage du problème

Une solution au problème, peut être codé comme une matrice de permutation à 12 lignes et 13 colonnes, à éléments binaires. A chaque élément de la matrice est associé un neurone. Ce codage nécessite donc (12x13) neurones et $(12x13)^2$ connexions entre les neurones. Un ordonnancement acceptable est représenté par une matrice contenant exactement un neurone allumé (élément=1) par ligne et ayant exactement 12 neurones allumés en totalité.

- **Étape 3** : Détermination de l'énergie du réseau

D'après l'équation (3.4), l'énergie du réseau est comme suit :

$$E=\frac{A}{2}\sum_{x=1}^{12}\sum_{i=1}^{13}\sum_{j=1\&j\neq i}^{13}v_{xi}v_{xj}+\frac{B}{2}\left(\sum_{x=1}^{12}\sum_{i=1}^{13}v_{xi}-12\right)^2+\frac{C}{2}\sum_{x=2}^{12}\sum_{i=2\&i\neq x+1}^{13}v_{xi}v_{(i-1)(x+1)}$$

La convergence des simulations à un ordonnancement valide est très sensible aux paramètres A, B et C dans les termes de la fonction d'énergie. Les trois constantes de pondération des contraintes sont déterminées après plusieurs ajustements (A=400, B=400 et C=200) (Voir paragraphe 3.7.1.3 page 45).

- **Étape 4** : Détermination des équations d'évolution des neurones

D'après l'équation (3.8), l'équation d'évolution des neurones est comme suit :

$$\frac{du_{xi}}{dt}=-u_{xi}-A\sum_{j=1\&j\neq i}^{12}v_{xj}-B\sum_{x=1}^{12}\sum_{j=1}^{13}(v_{xj}-12)-Cv_{(i-1)(x+1)}(1-\delta_{i1})(1-\delta_{x1})(1-\delta_{x(i-1)})(1-\delta_{i(x+1)})+I_{xi}$$

- **Étape 5** : Lancement de la dynamique

A partir d'un état initial aléatoire des entrées, une dynamique asynchrone, fondée sur les équations d'évolution des neurones, fait converger le réseau vers un minimum local de l'énergie, dans lequel les entrées et les sorties des neurones ne varient plus. A la convergence, en lisant les valeurs de sorties des neurones, on obtient un codage d'une solution potentielle au problème. La validité de cette solution doit cependant être vérifiée. Si elle n'est pas acceptable on doit ajuster les constantes de pondération (A, B et C).

- **Étape 6** : Simulation

L'algorithme présenté dans le paragraphe 3.7.1.6 est mis en application dans MATLAB. Pour plus de détails le lecteur doit se rapporter à la documentation du code de programme en annexe 3.

- **Étape 7** : Résultats de la simulation

Après plusieurs vérifications de la validité de la solution et plusieurs ajustements des constantes de pondération (A, B et C) (Voir paragraphe 3.7.1.3 page 50), en lisant les valeurs de sorties des neurones, on obtient un codage d'une solution faisable.
La figure suivante montre une représentation matricielle d'une solution faisable de l'ordonnancement.

	0	1	2	3	4	5	6	7	8	9	10	11	12
1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	1	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	1
9	0	0	0	0	0	1	0	0	0	0	0	0	0
10	1	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	1	0	0	0	0	0	0	0	0	0
12	0	0	1	0	0	0	0	0	0	0	0	0	0

Figure 15 Représentation matricielle de la solution (12 opérations)

La figure 16 représente l'évolution de l'énergie en fonction du temps. On remarque bien qu'elle converge rapidement vers l'état stable de réseau.

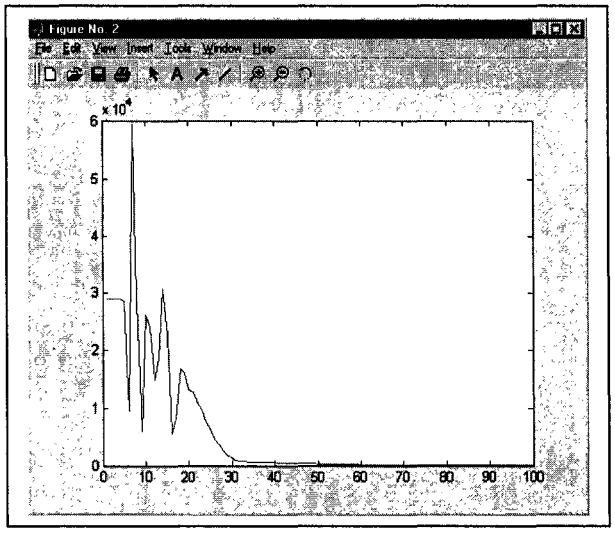


Figure 16 Évolution de la fonction d'énergie (12 opérations)

Les sorties des neurones qui sont 1 dans la matrice peuvent servir à construire des arbres de coût (Figure 17).

Les nœuds dans les arbres de coût représentent les opérations, chaque opération est codée par un triplet (k,l,m), k est le numéro de la tâche, l est l'ordre de l'opération et m est le numéro de la machine. Les chaînes avec la direction dénotent les rapports de dépendance sur chaque opération, et les poids des chaînes représentent la durée d'exécution des opérations. Le noeud sur le dessus de l'arbre représente l'opération nulle dont la durée de la transformation est 0.

Donc pour construire les arbres de coût au complet, on doit passer par tous les neurones qui sont 1 dans la matrice. Chaque neurone activé (sortie 1 de la matrice) représente la dépendance de l'opération correspondant au numéro de la ligne du neurone activé à l'opération correspondant au numéro de la colonne de celle-ci. Par exemple, dans la matrice (fig15), le neurone activé dans la ligne 2 et la colonne 1, représente la dépendance de l'opération 2 à l'opération 1 et le poids de la chaîne entre les deux nœuds représente la durée d'exécution de l'opération 2.

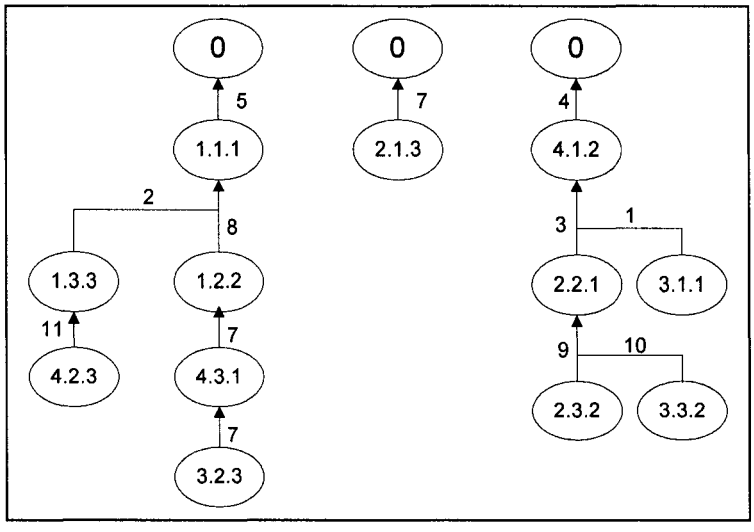


Figure 17 Les arbres de coût (12 opérations)

Basé sur les arbres de coût, nous allons construire le diagramme de Gantt (DG) qui est une planification représentant graphiquement les arbres de coût. Il permet le suivi des différentes opérations mises en œuvre. Il renseigne sur la durée d'une tâche, le moment où elle débute et celui où elle s'achève (Figure 18). Les lignes vont correspondre aux différentes machines et les colonnes vont correspondre aux unités de temps. Dans cette étape, nous allons rechercher les opérations antérieures à partir des arbres de coût. Il s'agit de savoir si la réalisation d'une tâche dépend de l'achèvement d'une autre.

Les arbres de coût nous indiquent que les opérations (1.1.1), (2.1.3) et (4.1.2) n'ont pas de tâches antérieures. Cela veut donc dire qu'on peut démarrer la réalisation de ces opérations dès l'instant zéro sur le diagramme de gantt. Avant de commencer les opérations (1.3.3) et (1.2.2), nous devons finir l'opération (1.1.1); avant de commencer les opérations (2.2.1) et (3.1.1), nous devons finir l'opération (4.1.2) et ainsi de suite jusqu'à la dernière opération.

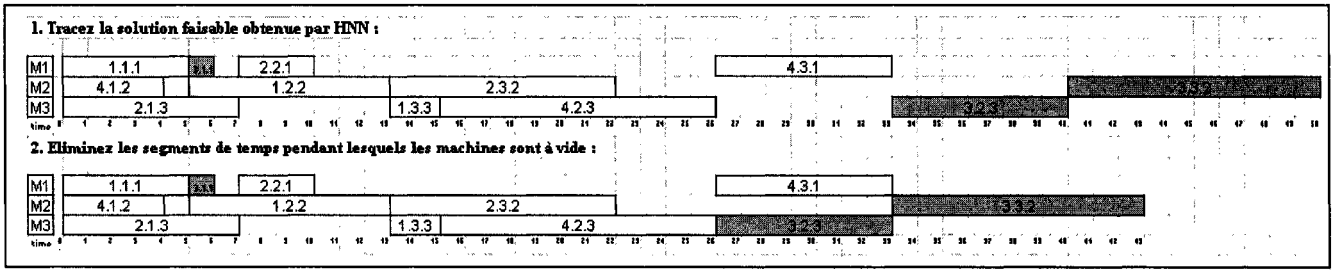


Figure 18 Diagramme de Gantt obtenu par RNH (Makespan=43)

En comparant notre résultat trouvé (Makespan=43) à l'optimum global présenté par Foo et Takefuji [12] et Xin-li, Wan-liang (2001) [15] (Makespan=32) on remarque bien les limitations de l'utilisation des réseaux de Hopfield déjà présentés dans le paragraphe 2.5.1.5.

La principale difficulté vient du fait que la dynamique du réseau conduit fréquemment vers un minimum local.

- **Étape 8** : Conclusion

Avec cet exemple on montre l'importance de limiter les défauts du réseau de Hopfield, en combinant ce dernier avec un algorithme de recherche locale.

3.7.3 Caractéristiques de la procédure de recherche locale

Puisqu'il n'y a aucune garantie que l'ordonnancement obtenu dans la première phase est un optimal global, la recherche locale peut être appliquée pour essayer de diminuer la durée totale de l'ordonnancement (makespan).

L'examen de la littérature de l'ordonnancement job shop [30] montre que les méthodes utilisées pour résoudre ce problème se sont graduellement améliorées au cours des dernières années. Elles sont basées sur le choix des permutations des opérations adjacentes qui nécessitent la même machine pour être traitées. La première contribution principale est fournie par Laarhooven (1988, 1992) [36 et 37]. D'autres améliorations ont été fournies par Grabowski (1988) [38] et Matsuo (1988) [39]. La recherche locale la plus récente est formulée par Nowicki et Smutnicki (NS) (1996).

Alors nous allons nous concentrer sur l'approche de Nowicki et de Smutnicki, pour proposer et mettre en application la procédure de recherche locale la plus restrictive dans la littérature. Le Nowicki et Smutnicki est une méthode qui exploite une stratégie de la recherche tabou, elle est reconnue largement comme la procédure la plus efficace pour obtenir des bonnes solutions aux problèmes d'ordonnancement dans un temps relativement court [30].

Quoique la méthode de NS applique une recherche locale fortement contrainte qui évite plusieurs itérations inutiles, définies par Laarhooven (1992) et Matsuo (1988). L'évaluation des permutations proposée par NS représente des stratégies puissantes et efficaces.

- **Étape 1** : Identification du chemin critique

La procédure de recherche locale commence en identifiant le chemin critique dans la solution obtenue par le réseau de Hopfield. Un chemin critique est la succession des opérations enclenchées entre elles sans marge de temps depuis le démarrage jusqu'à la fin des tâches. Tout retard sur une opération située sur le chemin critique compromet le délai final du projet et n'importe quelle opération sur le chemin critique s'appelle une opération critique. Les deux figures suivantes montrent l'exemple d'identification du

chemin critique. La figure 19 montre un exemple d'une solution initiale d'un problème job shop [17] et la figure 20 présente le chemin critique identifié (en couleur bleu).

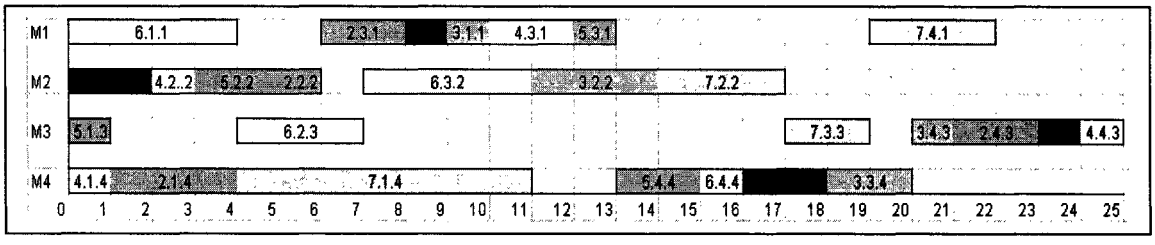


Figure 19 Diagramme de Gantt d'une solution initiale [17]

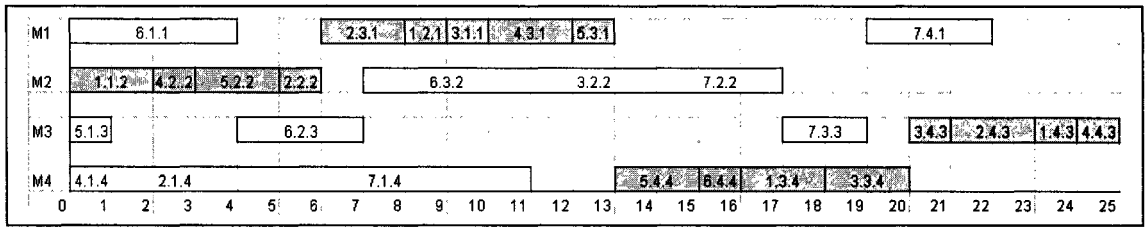


Figure 20 Chemin critique

- **Étape 2** : Identification des blocs critiques

La deuxième étape consiste à décomposer le chemin critique en un certain nombre de blocs où un bloc est un ordre maximal des opérations critiques adjacentes qui exigent la même machine et appartiennent au chemin critique. La figure suivante identifie les différents blocs de l'exemple précédent.

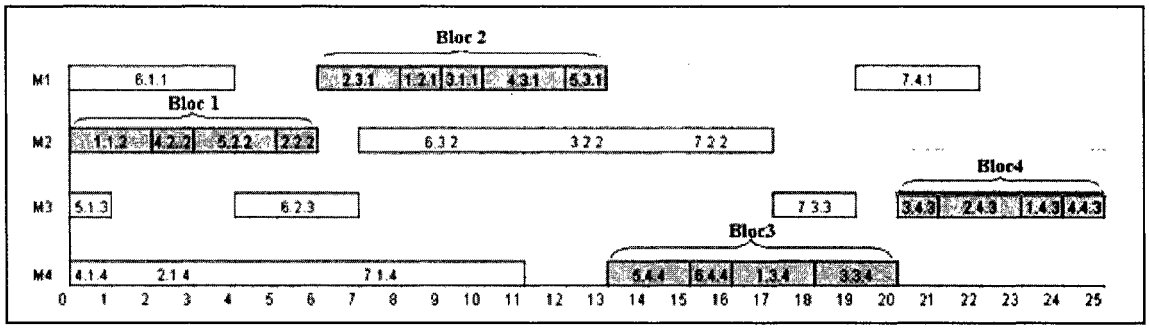


Figure 21 Blocs critiques

- **Étape 3** : Permutation des opérations dans les blocs critiques

Les permutations possibles entre les opérations diffèrent d'une approche à un autre. Le tableau suivant présenté par Jain, Rangaswamy et Meeran (2000) [30], présente le nombre de permutations possibles en fonction des approches présentées dans la littérature, pour un problème job shop à 6 tâches et 6 machines.

Tableau VI

Le nombre de permutations définies par chacune des diverses approches [30]

Approche	Nombre de permutations possibles
Van Laarhooven et al. (1988, 1992)	15
Grabowski et al. (1988)	11
Matsuo et al. (1988)	10
Nowicki and Smutnicki (1996)	2

On remarque bien que la procédure NS est la plus efficace pour obtenir des solutions dans un temps relativement court. Quoique la méthode de NS applique une recherche locale fortement contrainte qui **évite plusieurs permutations** des opérations inutiles définies par Laarhooven (1992) Grabowski et Matsuo (1988). Dans cette étape nous utilisons la recherche locale à deux échanges de NS, définie comme suit :

- Soient (b) blocs critiques donnés, les permutations possibles sont les suivantes (voir Figure 22) :

  - Pour le premier bloc, permuter seulement les deux dernières opérations de bloc
  - Pour le dernier bloc, permuter seulement les deux premières opérations de bloc
  - Pour les autres blocs, permuter seulement les deux dernières et deux premières opérations de bloc.

Dans le cas où le premier et/ou dernier bloc contiennent seulement deux opérations, ces dernières sont permutées. Si un bloc contient seulement une opération, alors aucun échange n'est fait. La figure suivante montre les différentes permutations possibles pour le même exemple présenté à l'étape 1 page 61.

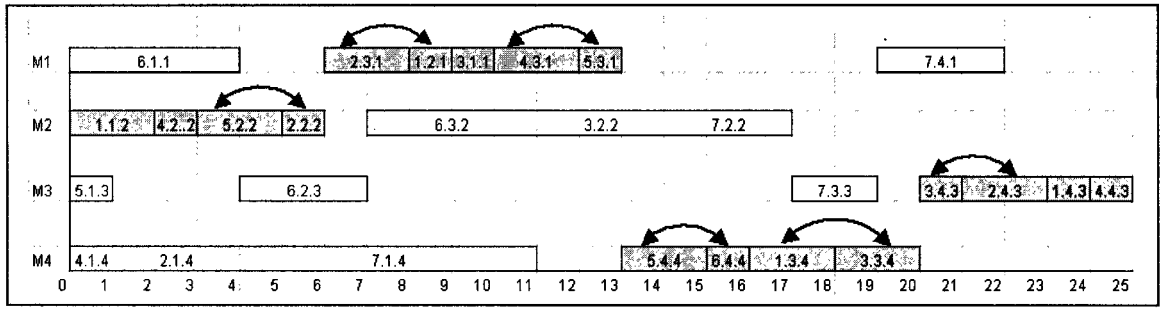


Figure 22 Les permutations possibles des opérations

Si la permutation améliore le Makespan alors on l'accepte. Autrement, la permutation est défaite. Une fois l'échange accepté, le chemin critique peut être changé et un nouveau chemin critique doit être identifié.

- **Étape 4** : Critères d'arrêt de la recherche tabou

Si la permutation des opérations dans n'importe quel bloc de chemin critique n'améliore pas le makespan, alors la recherche locale termine.

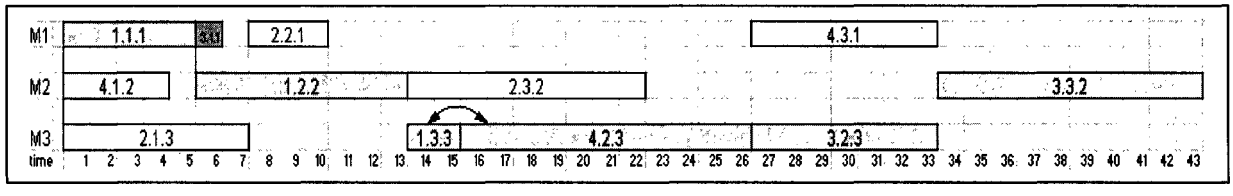
3.7.4 Mise en œuvre de la deuxième phase de l'approche proposée

Pour illustrer les propos qui précèdent à l'aide d'un petit exemple concret, nous allons appliquer la procédure de recherche tabou au même problème job shop présenté dans le paragraphe 3.7.2 (page 54) afin d'améliorer le résultat trouvé par le réseau de hopfield.

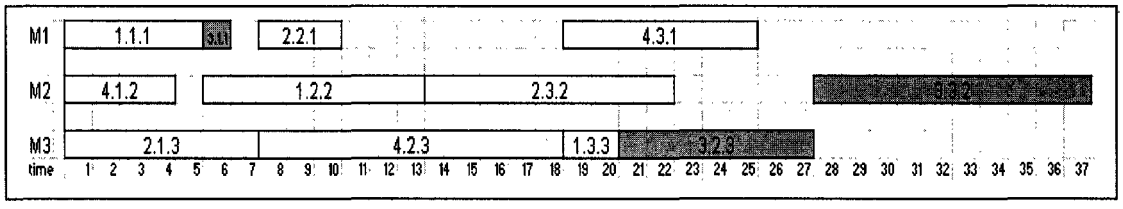
L'ordonnancement obtenu par RNH est un optimal local. Le diagramme de Gantt de la solution est présenté à la figure 18. Cet ordonnancement est considéré comme une solution initiale pour la procédure de recherche locale NS.

Itération 1 :

Identification de chemin critique, blocs critiques et les permutations possibles des opérations, à partir de l'optimal local trouvé (figure 18),

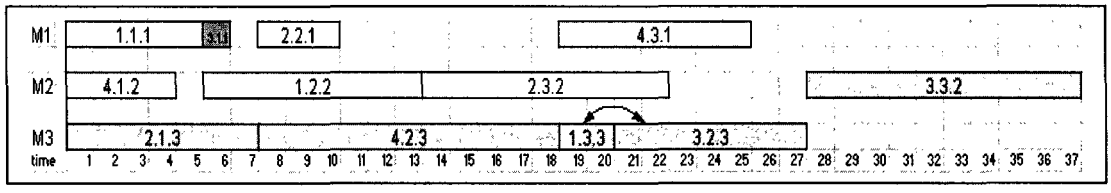


Après la permutation de l'opération (1.3.3) et (4.2.3) et l'élimination des segments de temps pendant lesquels les machines sont à vide (idle time), la solution améliorée (1) est trouvée :

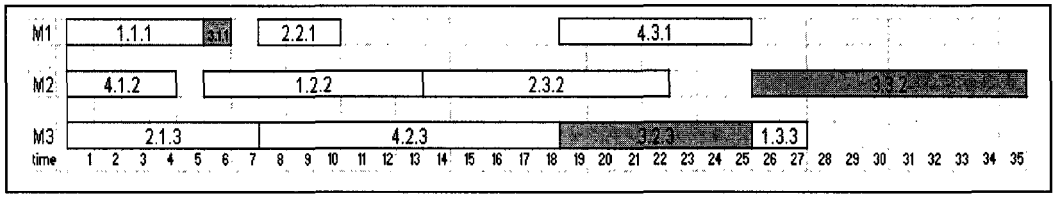


Itération 2 :

Identification du nouveau chemin critique, blocs critiques et les permutations possibles des opérations :

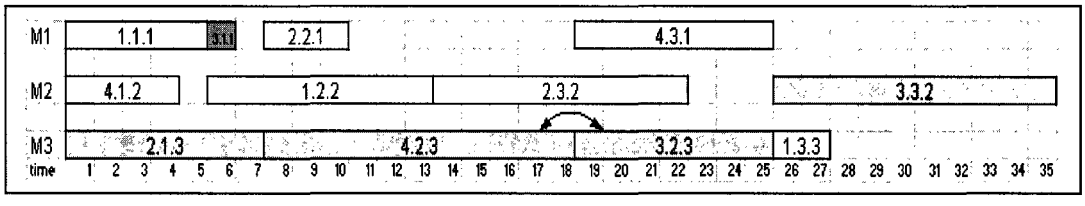


Traçage de la solution améliorée (2) :

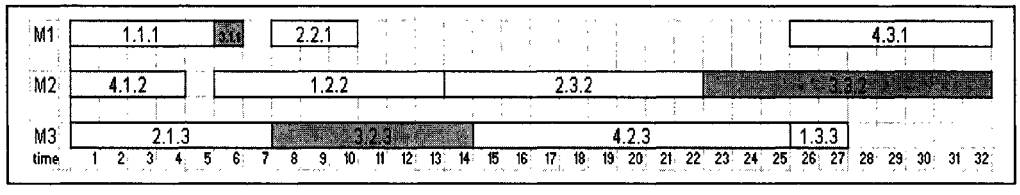


Itération 3 :

Identification du nouveau chemin critique, blocs critiques et les permutations possibles des opérations :

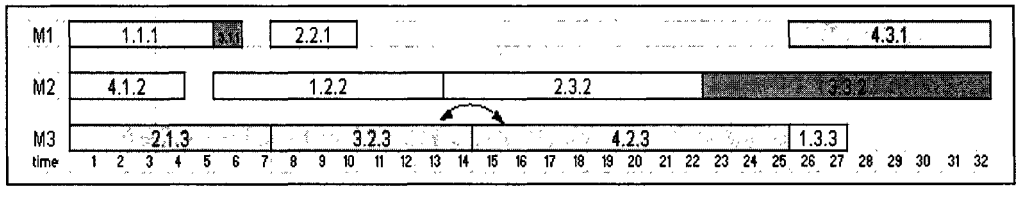


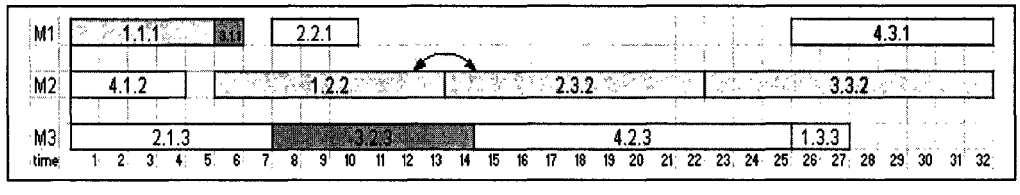
Traçage de la solution améliorée (3) :



Itération 4 :

On a identifié deux nouveaux chemins critiques :





La permutation des opérations dans n'importe quel bloc des chemins critiques n'améliore pas le makespan, alors la recherche locale est finie et la solution finale (Makespan=32) a été trouvée.

En comparant notre résultat trouvé à l'optimum global présenté par Foo et Takefuji [12] et Xin-li et Wan-liang [15] (Makespan=32), on remarque bien qu'après trois itérations l'optimal global est obtenu.

La figure suivante montre l'évolution du Makespan en fonction des itérations.

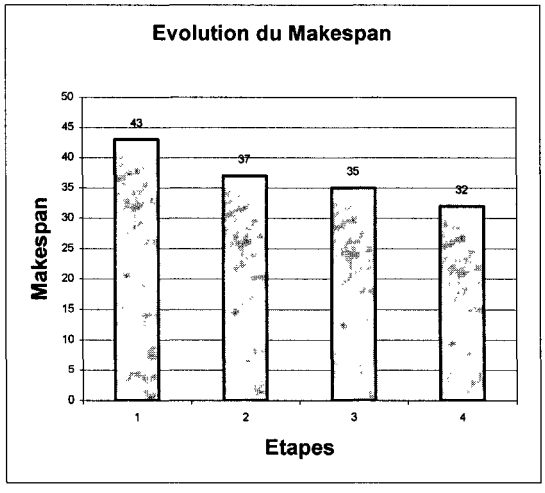


Figure 23 Évolution du Makespan (12 opérations)

3.8 Performance de l'approche présentée

Afin de voir le comportement du système et le degré de la robustesse, nous avons examiné l'approche avec le même exemple présenté précédemment (page 54), en changeant les entrées aléatoires des neurones de Hopfield. Tous les autres paramètres ont été jugés constants.

Pour chaque entrée aléatoire, une simulation a été effectuée. Les résultats de l'étude de simulation peuvent être visualisés dans le tableau suivant :

Tableau VII

Résultats de simulations avec différentes entrées de neurones

	Première phase de l'approche (RNH)		**Deuxième phase de l'approche (RL)**	
	Temps d'exécution (seconde)	**Makespan**	**Nombre d'itérations**	**Makespan**
1ère simulation	5	43	3	32
2ème simulation	5	34	1	33
3ème simulation	5	40	3	32
4ème simulation	5	38	3	32

D'après ces résultats on remarque bien l'efficacité de notre approche au niveau temps de résolution ainsi que la qualité de la solution, quelque soit l'état initiale des neurones d'entrée.

3.9 Conclusion

Ce travail présente le réseau de Hopfield hybride pour le problème d'ordonnancement job shop et la production cellulaire. Le réseau de neurone de Hopfield est utilisé pour obtenir des solutions faisables de l'ordonnancement et l'algorithme heuristique est utilisé pour améliorer la qualité des solutions obtenues.

Les résultats du problème (12opérations) obtenus par simulation valident l'approche hybride et montrent leur efficacité au niveau de temps de résolution ainsi que de la qualité de la solution finale. Puisque il y a beaucoup de contraintes à satisfaire dans les problèmes d'ordonnancement job shop et production cellulaire, la première partie de

notre approche (le réseau de neurone de Hopfield) a montré plus d'efficacité de satisfaction de contrainte que la minimisation du coût, puisque elle nous donne une solution acceptable de l'ordonnancement par une simulation sur matlab dans 5 secondes. L'efficacité de la deuxième partie de l'approche (recherche taboue de Nowicki et de Smutnicki 1996) est constatée, puisque on a obtenu des bons résultats au problème d'ordonnancement en quelques itérations.

Est-ce que notre approche hybride est efficace aussi pour les problèmes de moyenne et grande taille ? Dans le chapitre 4, on va essayer de répondre à cette question.

CHAPITRE 4

VALIDATION ET SYNTHÈSE

4.1 Introduction

Nous présentons ici la validation et le degré d'efficacité de la nouvelle approche présentée dans le chapitre précédent, pour les problèmes de moyenne et grande taille, appuyé par des cas typiques en reprenant des exemples de la littérature. Pour chaque application, nous présentons le problème et nous décrivons brièvement les différentes phases de notre approche mise en œuvre pour le résoudre. Les résultats seront détaillés et la robustesse de l'approche sera évaluée.

4.2 Exemple d'ordonnancement Job Shop de 28 opérations

Nous allons détailler la résolution d'un problème job shop de taille moyenne, proposée par Hanada et Ohnishi en 1993 [11].

- **Étape 1** : données du problème

L'exemple d'ordonnancement Job Shop est composé de huit tâches, 6 machines et en totalité 28 opérations. Les opérations de chaque tâche devraient être faites dans l'ordre donné par le tableau suivant.

Tableau VIII

Exemple d'ordonnancement JS (28 opérations)

Tâche	(Opération) (Machine) (Temps)			
J1	(O1) (M5) (9)	(O2) (M6) (12)	(O3) (M3) (11)	(O4) (M2) (6)
J2	(O5) (M4) (5)	(O6) (M2) (4)	(O7) (M6) (3)	(O8) (M6) (3)
J3	(O9) (M1) (5)	(O10) (M3) (4)	(O11) (M3) (3)	
J4	(O12) (M4) (5)	(O13) (M6) (4)	(O14) (M1) (3)	(O15) (M4) (3)
J5	(O16) (M6) (5)	(O17) (M1) (4)	(O18) (M3) (3)	
J6	(O19) (M4) (5)	(O20) (M5) (4)	(O21) (M6) (3)	(O22) (M4) (3)
J7	(O23) (M6) (5)	(O24) (M3) (4)	(O25) (M6) (3)	
J8	(O26) (M6) (2)	(O27) (M5) (2)	(O28) (M2) (2)	

- **Étape 2** : Codage du problème

Une solution au problème peut être codée comme une matrice de permutation à 28 lignes et 29 colonnes, à éléments binaires. A chaque élément de la matrice est associé un neurone. Ce codage nécessite donc (28x29) neurones et $(28x29)^2$ connexions entre les neurones. Un ordonnancement acceptable est représenté par une matrice contenant exactement un neurone allumé (élément=1) par ligne et ayant exactement 28 neurones allumés en totalité.

- **Étape 3** : Détermination de l'énergie du réseau

D'après l'équation (3.4), l'énergie du réseau est comme suit :

$$E=\frac{A}{2}\sum_{x=1}^{28}\sum_{i=1}^{29}\sum_{j=1\&j\neq i}^{29}v_{xi}v_{xj}+\frac{B}{2}\left(\sum_{x=1}^{28}\sum_{i=1}^{29}v_{xi}-28\right)^2+\frac{C}{2}\sum_{x=2}^{28}\sum_{i=2\&i\neq x+1}^{29}v_{xi}v_{(i-1)(x+1)}$$

Les trois constantes de pondération des contraintes A, B et C sont déterminées après plusieurs ajustements (A=200, B=200 et C=100). C'est à dire après la convergence, si l'on obtient une solution non acceptable (elle viole une ou plusieurs contrainte), on peut redémarrer le réseau en ajustant les coefficients (A, B et C), par exemple en augmentant celui qui est associé à une contrainte violée.

- **Étape 4** : Détermination des équations d'évolution des neurones

D'après l'équation (3.8), l'équation d'évolution des neurones est comme suit :

$$\frac{du_{xi}}{dt}=-u_{xi}-A\sum_{j=1\&j\neq i}^{28}v_{xj}-B\sum_{x=1}^{28}\sum_{j=1}^{29}(v_{xj}-28)-Cv_{(i-1)(x+1)}(1-\delta_{i1})(1-\delta_{x1})(1-\delta_{x(i-1)})(1-\delta_{i(x+1)})+I_{xi}$$

- **Étape 5** : Simulation

L'algorithme présenté dans le paragraphe 3.7.1.6 est mis en application dans MATLAB. Pour plus de détails le lecteur doit se rapporter à la documentation du code de programme en annexe 4.

- **Étape 6** : Résultats de la simulation

Après plusieurs vérifications de la validité de la solution et plusieurs ajustements des constantes de pondération (A, B et C), en lisant les valeurs de sorties des neurones, on obtient un codage d'une solution faisable.

La figure suivante montre une représentation matricielle d'une solution faisable de l'ordonnancement.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
15	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
24	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
28	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24 Représentation matricielle de la solution (28 opérations)

La figure 25 représente l'évolution de l'énergie en fonction du temps. On remarque bien qu'elle converge rapidement vers l'état stable de réseau.

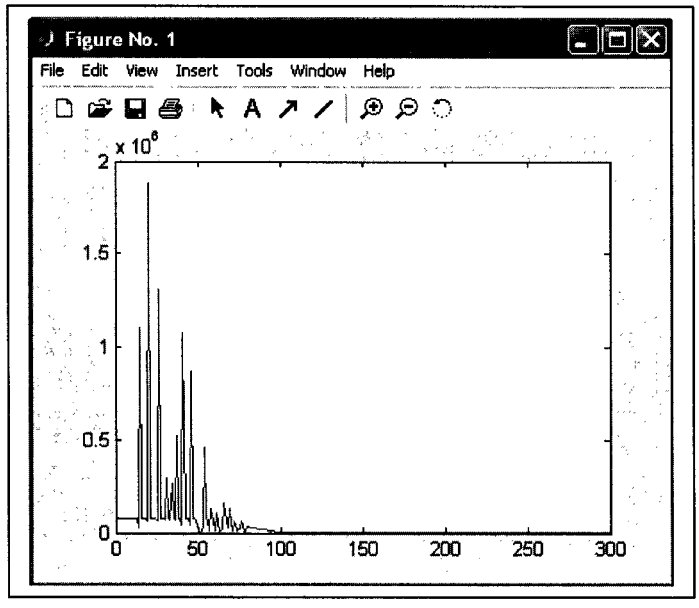


Figure 25 Évolution de la fonction d'énergie (28 opérations)

A partir de la représentation matricielle de la solution (figure 24), les arbres de coût sont construits (Figure 26).

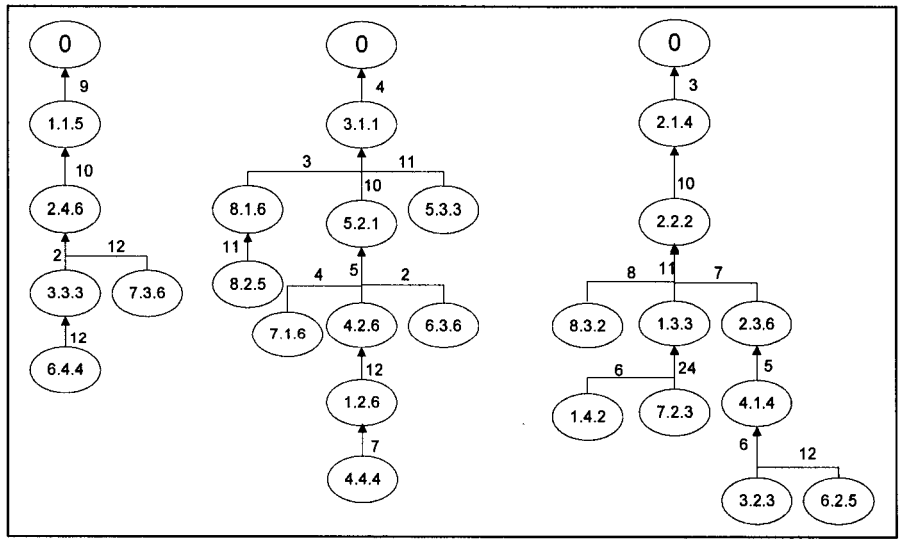


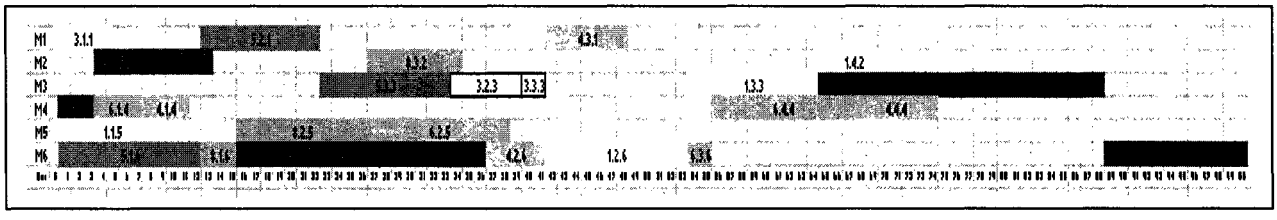
Figure 26 Les arbres de coût (28 opérations)

Basé sur les arbres de coût, nous avons construit le diagramme de Gantt. L'ordonnancement obtenu par RNH est un optimal local (Makespan=102).

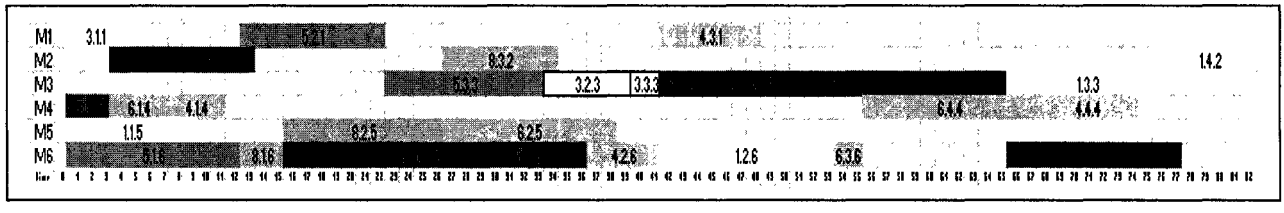
- **Étape 7** : Application de la recherche tabou

Une recherche tabou est donc appliquée pour essayer d'améliorer cette solution. Après quatre itérations, la solution finale est obtenue (Makespan=74).

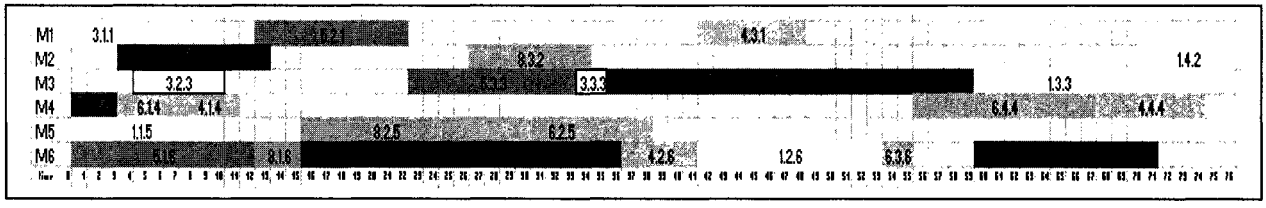
- Solution de la 1er itération : (Makespan=100)



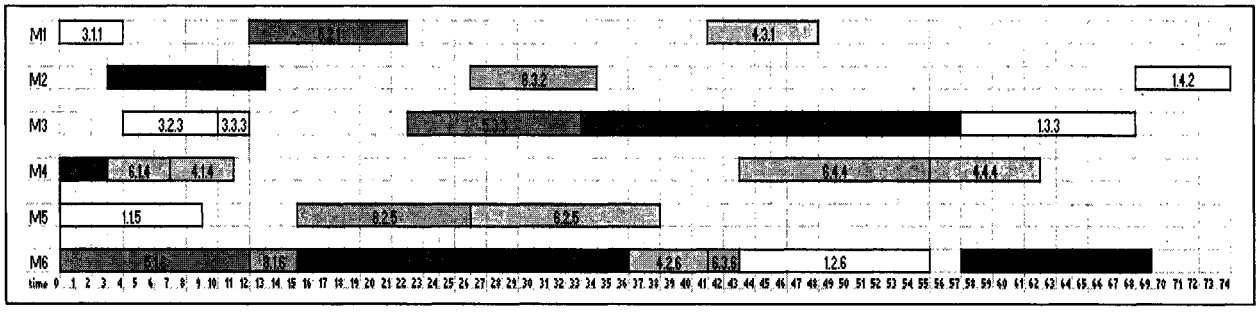
- Solution de la 2ème itération : (Makespan=82)



- Solution de la 3ème itération : (Makespan=76)



- Solution de la 4ème itération : c'est la solution améliorée finale (Makespan=74)



La figure 27 montre l'évolution du Makespan en fonction des étapes.

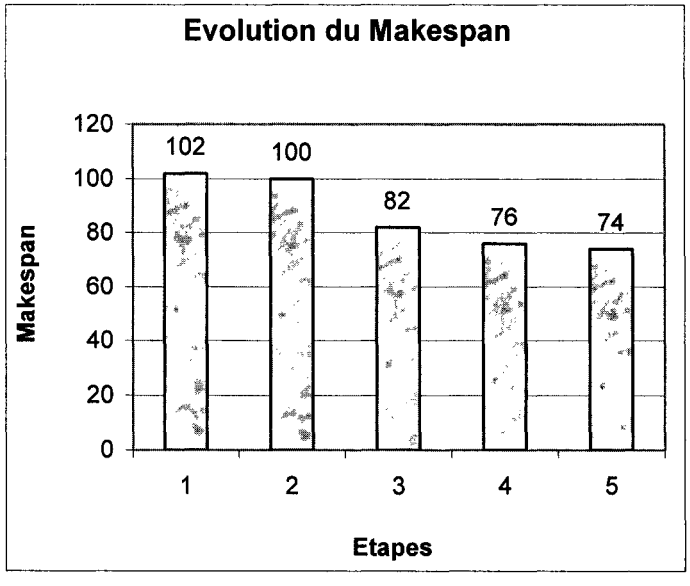


Figure 27 Évolution du Makespan (28 opérations)

- **Étape 8** : Conclusion

Les résultats obtenus par simulation valident encore l'approche hybride et montrent leur efficacité au niveau de temps de résolution ainsi que la qualité de la solution finale.

4.3 Exemple d'ordonnancement Job Shop de 50 opérations

Nous allons détailler la résolution d'un problème job shop de grande taille, proposée par Hanada et Ohnishi en 1993 [11].

- **Étape 1** : données de problème

L'exemple d'ordonnancement Job Shop est composé de 10 tâches, 8 machines et en totalité 50 opérations. Les opérations de chaque tâche devraient être faites dans l'ordre donné par le tableau suivant.

Tableau IX

Exemple d'ordonnancement JS (50 opérations)

Tâche	(Opération) (Machine) (Temps)				
J1	(O1) (M5) (9)	(O2) (M2) (6)	(O3) (M7) (12)	(O4) (M1) (3)	(O5) (M4) (7)
J2	(O6) (M2) (4)	(O7) (M8) (6)	(O8) (M1) (10)	(O9) (M4) (7)	(O10) (M3) (10)
J3	(O11) (M6) (3)	(O12) (M5) (7)	(O13) (M3) (12)	(O14) (M8) (6)	(O15) (M1) (4)
J4	(O16) (M1) (5)	(O17) (M8) (10)	(O18) (M4) (3)	(O19) (M3) (11)	(O20) (M4) (6)
J5	(O21) (M6) (12)	(O22) (M7) (6)	(O23) (M1) (5)	(O24) (M7) (12)	(O25) (M2) (4)
J6	(O26) (M4) (5)	(O27) (M6) (7)	(O28) (M5) (3)	(O29) (M3) (2)	(O30) (M5) (11)
J7	(O31) (M1) (4)	(O32) (M7) (7)	(O33) (M8) (4)	(O34) (M3) (10)	(O35) (M2) (12)
J8	(O36) (M6) (4)	(O37) (M5) (5)	(O38) (M3) (9)	(O39) (M2) (12)	(O40) (M7) (3)
J9	(O41) (M7) (6)	(O42) (M4) (5)	(O43) (M8) (12)	(O44) (M3) (7)	(O45) (M1) (5)
J10	(O46) (M5) (12)	(O47) (M4) (7)	(O48) (M7) (11)	(O49) (M6) (8)	(O50) (M7) (8)

- **Étape 2** : Codage du problème

Une solution au problème peut être codée comme une matrice de permutation à 50 lignes et 51 colonnes, à éléments binaires. A chaque élément de la matrice est associé un neurone. Ce codage nécessite donc (50x51) neurones et $(50x51)^2$ connexions entre les neurones. Un ordonnancement acceptable est représenté par une matrice contenant exactement un neurone allumé (élément=1) par ligne et ayant exactement 50 neurones allumés en totalité.

- **Étape 3** : Détermination de l'énergie du réseau

D'après l'équation (3.4), l'énergie du réseau est comme suit :

$$E=\frac{A}{2}\sum_{x=1}^{50}\sum_{i=1}^{51}\sum_{j=1\&j\neq i}^{51}v_{xi}v_{xj}+\frac{B}{2}\left(\sum_{x=1}^{50}\sum_{i=1}^{51}v_{xi}-50\right)^2+\frac{C}{2}\sum_{x=2}^{50}\sum_{i=2\&i\neq x+1}^{51}v_{xi}v_{(i-1)(x+1)}$$

Les trois constantes de pondération des contraintes A, B et C sont determinées après plusieurs ajustements (A=110, B=110 et C=50).

- **Étape 4** : Détermination des équations d'évolution des neurones

D'après l'équation (3.8), l'équation d'évolution des neurones est comme suit :

$$\frac{du_{xi}}{dt}=-u_{xi}-A\sum_{j=1\&j\neq i}^{50}v_{xj}-B\sum_{x=1}^{50}\sum_{j=1}^{51}(v_{xj}-50)-Cv_{(i-1)(x+1)}(1-\delta_{i1})(1-\delta_{x1})(1-\delta_{x(i-1)})(1-\delta_{i(x+1)})+I_{xi}$$

- **Étape 5** : Simulation

L'algorithme présenté dans le paragraphe 3.7.1.6 est mis en application dans MATLAB. Pour plus de détails le lecteur doit se rapporter à la documentation du code de programme en annexe 5.

- **Étape 6** : Résultats de la simulation

Après plusieurs vérifications de la validité de la solution et plusieurs ajustements des constantes de pondération (A, B et C), en lisant les valeurs de sorties des neurones, on obtient un codage d'une solution faisable.

La figure 28 représente l'évolution de l'énergie en fonction du temps. On remarque bien qu'elle converge rapidement vers l'état stable de réseau.

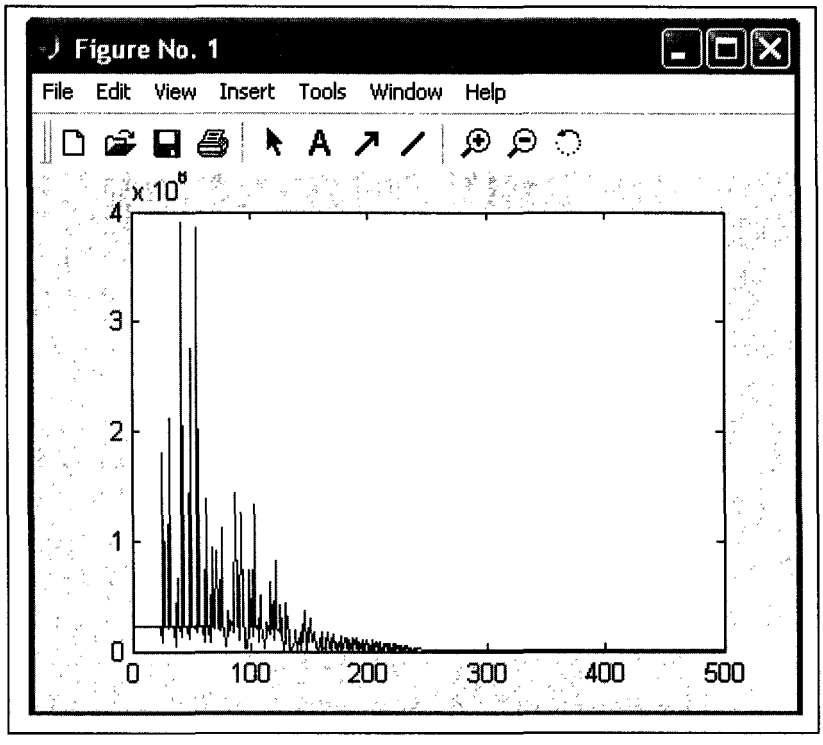


Figure 28 Évolution de la fonction d'énergie (50 opérations)

L'ordonnancement obtenu par RNH est un optimal local (Makespan=151). Ainsi, une recherche taboue est nécessaire.

- **Étape 7** : Application de la recherche taboue

La recherche locale est appliquée pour essayer de diminuer le makespan. Après douze itérations la solution améliorée finale est obtenue (Figure 29). La figure 30 montre l'évolution du Makespan en fonction des itérations.

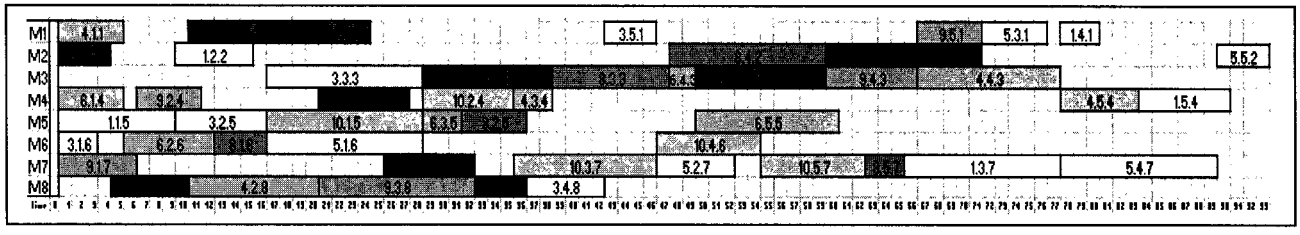


Figure 29 Diagramme de Gantt de la solution améliorée finale (Makespan=93)

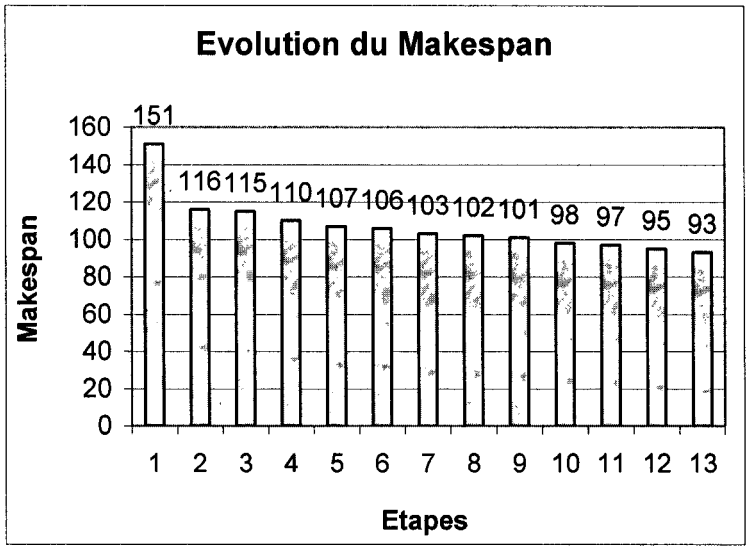


Figure 30 Évolution du Makespan (50 opérations)

- **Étape 8** : Conclusion

Les résultats obtenus par simulation valident encore l'approche hybride et montrent leur efficacité au niveau de temps de résolution pour le problème job shop et production cellulaire de grande taille. Mais la solution finale est un minimum local (Makespan=93).

4.4 Conclusion

En observant le comportement du simulateur dans le problème de petite taille, l'état du système converge à la solution optimale globale en quelques itérations. En augmentant la taille de problème, la qualité des solutions obtenues diminue et le nombre d'itérations augmente proportionnellement.

Le réseau de neurone de Hopfield a montré plus d'efficacité de satisfaction de contrainte que la minimisation du coût, puisque il est capable d'exécuter un grand nombre de calculs dans un temps limité même pour les problème de grande taille. L'efficacité de la deuxième partie de l'approche (recherche tabou de Nowicki et de Smutnicki 1996) est constatée aussi puisque on a obtenu de bonnes solutions en quelques itérations.

D'un point de vue pratique, quand le temps de résolution d'un problème est borné par l'application, notre approche devient plus avantageuse que les autres algorithmes de recuit simulé et les algorithmes génétiques, car notre méthode ne nécessite pas une mise au point très délicate et ne nécessite pas des capacités de mémoire importantes. Aussi elle est avantageuse par rapport aux autres approches classiques, vu qu'elle peut résoudre des problèmes de grandes tailles en donnant des bons résultats.

CONCLUSION

Ce mémoire résume les activités de recherche pour l'élaboration d'un outil d'aide à la décision pour l'ordonnancement. Cet outil est basé sur une approche d'optimisation composée d'un processus de deux étapes : un réseau de neurone de Hopfield et un heuristique. Le choix de la méthodologie fut présenté et justifié. D'abord, le réseau de neurone de Hopfield avec leur capacité d'exécuter un grand nombre de calculs dans un temps limité a offert une alternative intéressante pour résoudre les problèmes d'optimisation d'ordonnancement de grande taille. Ce système de neurones est utilisé pour obtenir des solutions faisables non optimales et l'algorithme heuristique est utilisé comme une recherche tabou (local) pour raffiner la qualité des solutions obtenues.

Les principes exposés dans ce mémoire pour l'ordonnancement de la production ne prétendent pas résoudre le problème dans sa globalité, mais nous espérons avoir développé quelques résultats susceptibles d'être utilisés dans un cadre opérationnel.

Nous avons formulé le problème d'ordonnancement et nous nous sommes attachés à détailler la façon de modéliser les équations générales de la dynamique, qui est l'étape préalable pour toute utilisation d'algorithme de résolution.

Le test de l'approche sur trois exemples présentés dans le chapitre 3 et 4, permet de tirer les conclusions suivantes :

- La nouvelle approche peut fournir de bons résultats même pour des problèmes de production de grande taille.
- L'approche permet de faire un ordonnancement judicieux dans un temps limité.
- L'approche offre une flexibilité acceptable, car elle peut s'adapter aux environnements manufacturiers les plus complexe JS et PC.

- Notre approche nécessite des capacités de mémoire moins importantes et une mise au point moins délicate en la comparant aux algorithmes génétiques et aux algorithmes de recuit simulé.
- La qualité des solutions diminue en augmentant la taille de problème et le nombre d'itérations augmente proportionnellement.
- L'approche proposée, n'est pas assez flexible pour résoudre un problème d'ordonnancement avec différents objectifs (critères), car la deuxième phase de l'approche (recherche locale) est utilisée pour optimiser seulement un seul critère, qui est le "makespan". Mais la première phase (RNH) reste valable pour tous les critères d'ordonnancement. Les développements peuvent être poussés pour rendre l'approche plus flexible.

Cette approche d'ordonnancement constitue un départ pour l'utilisation des réseaux de neurones hybride dans l'optimisation de l'ordonnancement. Pour développer d'avantage cette approche, plusieurs travaux de recherche ou de projets d'études pourraient être mis de l'avant. Il serait intéressant de valider notre modèle de simulation avec une analyse de cas réel en milieu industriel.

Enfin, une dernière piste de recherche ciblerait le type de réseau de Hopfield. L'idée d'étudier un réseau récent de Hopfield (Quantized Hopfield) [20] peut représenter un défi intéressant, puisqu'il permet d'obtenir les solutions optimales très fréquemment et beaucoup plus rapidement que d'autres réseaux de Hopfield ordinaires.

ANNEXE 1

LES PROPRIÉTÉS DE CONVERGENCE DES RÉSEAUX DE HOPFIELD

La fonction d'énergie correspondant à la version continue des réseaux de neurones de Hopfield est de la forme :

$$E(y)=-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}w_{ij}y_i y_j-\sum_{i=1}^{N}I_i x_i+\frac{\alpha}{\gamma}\sum_{i=1}^{n}\int_0^{y_i}\Psi^{-1}(y)dy$$

L'équation d'évolution d'un neurone continu *i* est donnée par l'équation différentielle suivante :

$$\frac{dv_i}{dt}=\mu_i\left[-\alpha_i v_i-\frac{\partial E(y)}{\partial y_i}\right]$$

et la sortie est :

$$y_i=th(\frac{\mu_i}{T})$$

Dans ces équations, $\mu_i = 1/\tau_i$ est un réel positif paramétrant la vitesse de convergence, ***T*** est la température commandant la pente à l'origine (T est l'inverse de la pente à l'origine de la fonction d'activation du neurone).

La dérivée de la fonction énergie *E* par rapport au temps peut s'écrire, d'après les équations d'évolution des neurones que l'on vient de présenter :

$$\frac{dE}{dt}=\sum_{i=1}^{N}\frac{\partial E}{\partial y_i}\frac{dy_i}{dt}=-\sum_{i=1}^{N}\tau_i\frac{dy_i}{dv_i}\left(\frac{du_i}{dt}\right)^2-\sum_{i=1}^{N}\alpha_i v_i\frac{dy_i}{dt}$$

On voit clairement que le premier terme de cette équation est toujours négatif ou nul car $\tau_i > 0$ et $dy_i/dv_i > 0$. En revanche, le second terme peut être positif, négatif ou nul ; en conséquence, la dérivée temporelle de la fonction d'énergie peut être positive (Takefuji 1992). Pour éviter cela, on utilise fréquemment une version différente des équations de mouvement, qui est la suivante :

$$\frac{dy_i}{dt}=-\mu_i\frac{\partial E(y)}{\partial y_i} \qquad i = 1,\ldots,N$$

Du point de vue des sorties des neurones, on obtient, à partir de ces équations de mouvement, les équations suivantes :

$$\frac{dy_i}{dt} \frac{\mu_i}{T}(1-y_i^2)\frac{\partial E(y)}{\partial y_i} \quad i = 1,\ldots,N$$

Les propriétés dynamiques d'un réseau de Hopfield analogique sont donc régies par un système d'équations différentielles non linéaires.
Avec cette modification, on peut montrer facilement que tout changement d'état (de sortie) d'un neurone se traduit sur l'énergie du réseau par une diminution ou un maintien à sa valeur. En effet :

$$\frac{dE}{dt}=\sum_{i=1}^{N}\frac{\partial E}{\partial y_i}\frac{dy_i}{dt}=\sum_{i=1}^{N}\left[-\tau_i\frac{dv_i}{dt}\right]\frac{dy_i}{dt}=-\sum_{i=1}^{N}\tau_i\frac{dy_i}{dv_i}\left(\frac{dv_i}{dt}\right)^2$$

$$\frac{dE}{dt}\leq 0$$

En d'autres termes, le système d'équations d'évolution contraint la fonction d'énergie E à décroître d'une manière monotone vers un minimum local. En pratique, avec une réalisation électronique d'un tel réseau, les temps de convergence sont de l'ordre de quelques nanosecondes ou microsecondes. Cela autorise en général plusieurs résolutions du problème en partant de différents points initiaux. Cette stratégie peut parfois se montrer efficace, mais en général les minima attracteurs ne sont pas de qualité suffisante.

ANNEXE 2

QUELQUES DÉFINITIONS UTILES

Définition du problème NP-Complet :

En général, un problème est formalisé de la manière suivante : un ensemble de données en entrée, et une question sur ces données (et éventuellement un calcul à effectuer). La théorie de la complexité ne traite que des problèmes de décision, c.-à-d. des problèmes qui posent une question et dont la réponse est soit *oui*, soit *non*. L'ensemble des instances d'un problème est l'ensemble des données que peut prendre ce problème en entrée. Une solution pour un problème est une structure représentable sur ordinateur qui est totalement indépendante du problème, par exemple l'ensemble des permutations sur n entiers.

On distingue essentiellement ces trois classes de complexité :

- Classe P : Un problème de décision est dans P s'il peut être résolu par un algorithme déterministe en un temps polynomial par rapport à la taille de l'instance. On dit alors que le problème est polynomial.
- Classe NP : C'est la classe des problèmes de décision pour lesquels la réponse *oui* peut être décidée par un algorithme non-déterministe en un temps polynomial par rapport à la taille de l'instance.
- Classe Co-NP : C'est l'équivalent de la classe NP pour la réponse *non*.

Un problème est **NP-Complet** s'il est dans NP, et si n'importe quel problème NP-Complet peut se réécrire à l'aide d'un algorithme polynomial comme un sous ensemble d'instance de ce problème.

De manière plus intuitive, dire qu'un problème peut être décidé à l'aide d'un algorithme non-déterministe polynomial, signifie qu'il est facile pour une solution donnée à vérifier en un temps polynomial si celle-ci répond au problème pour une instance donnée; mais que le nombre de solutions à tester pour résoudre le problème est exponentiel par rapport à la taille de l'instance. Le non-déterminisme permet de masquer la taille exponentielle des solutions à tester tout en permettant à l'algorithme de rester polynomial.

Définition de réseaux supervisés

Un réseau est dit supervisé lorsqu'on le force à converger vers un état final précis, en même temps qu'on lui présente une forme spécifique à l'entrée. Dans ce type d'apprentissage, le réseau s'adapte par comparaison entre le résultat qu'il a calculé, en fonction des entrées fournies, et la réponse attendue en sortie. Ainsi, le réseau va se modifier jusqu'à ce qu'il trouve la bonne sortie, c'est-à-dire celle attendue, correspondant à une entrée donnée.

Définition de réseaux non supervisés

Il existe également des réseaux de neurones qui s'entraînent sans besoin de supervision, c'est-à-dire, sans que l'on ait besoin de signifier au réseau comment il doit (devrait) se comporter. Le réseau est laissé libre de converger vers n'importe quel état final lorsqu'on lui présente une entrée.

Définition de réseaux récurrents

Appelés aussi réseaux rétroaction, ce sont des réseaux dans lesquels il y a retour en arrière de l'information.

ANNEXE 3

CODE MATLAB POUR L'EXEMPLE 1 (12 OPERATIONS)

```
%% Initialiser les paramètres :
N=12;
A=400;
B=400;
C=200;
to=1;
b=-100;
c=10;
deltaT=0.0005;
%% Représentation matricielle des bias :
I=[300,b,b,b,0,0,0,0,0,0,0,0,0;
   b,c,b,b,0,0,0,0,0,0,0,0,0;
   b,0,c,b,0,0,0,0,0,0,0,0,0;
   300,0,0,0,b,b,b,0,0,0,0,0,0;
   b,0,0,0,c,b,b,0,0,0,0,0,0;
   b,0,0,0,0,c,b,0,0,0,0,0,0;
   0,0,0,0,0,0,0,b,b,b,0,0,0;
   b,0,0,0,0,0,0,c,b,b,0,0,0;
   b,0,0,0,0,0,0,0,c,b,0,0,0;
   300,0,0,0,0,0,0,0,0,0,b,b,b;
   b,0,0,0,0,0,0,0,0,0,0,c,b,b;
   b,0,0,0,0,0,0,0,0,0,0,0,c,b];
%% Initialiser aléatoirement l'entrée Uij(t) :
u=-2+4*rand(N,N+1);
%% Calculer la sortie Vij(t) :
for x=1:N
   for i=1:N+1
      v(x,i)=0.5*(1+tanh(1*u(x,i)));
   end
end
%% Boucle principal de calcul :
for t=1:100
%% Calculer ΔUij(t) :
D=eye(N+1,1); %% C'est delta(i.1)
DD=eye(N,1); %% C'est delta(x,1)
deltaP=zeros(N,N); %% C'est delta(x,i-1)
deltaM=zeros(N+1,N+1); %% C'est delta(i,x+1)
for i=2:N
   deltaP(i,i-1)=1;
end
deltaP(1,N)=1;
for i=1:N
   deltaM(i,i+1)=1;
```

```
end
deltaM(N+1,1)=1;

for x=1:N
   for i=1:N+1
      if i==1
      deltau(x,i)=(-(u(x,i)/to)-A*(sum(v(x,:))-v(x,i))-B*(sum(sum(v))-N)+I(x,i))*deltaT;
      else
      deltau(x,i)=(-(u(x,i)/to)-A*(sum(v(x,:))-v(x,i))-B*(sum(sum(v))-N)-C*v(i-
1,x+1)*(1-D(i,1))*(1-DD(x,1))*(1-deltaP(x,i-1))*(1-deltaM(i,x+1))+I(x,i))*deltaT;
      end
   end
end
%% Calculer Uij(t+1) :
for x=1:N
   for i=1:N+1
      u(x,i)=u(x,i)+deltau(x,i);
   end
end
%% Mettre a jour la sortie :
for x=1:N
   for i=1:N+1
      v(x,i)=0.5*(1+tanh(1*u(x,i)));
   end
end
%% Calculer l'énergie E(t) :
v2=zeros(N,N);
for i=1:N
   for j=1:N
      v2(i,j)=v(i,j+1);
   end
end
helper=0;
for x=1:N
   for i=1:N+1
      for j=1:N+1
         if i~=j
            helper=helper+(v(x,i)*v(x,j));
         end
      end
   end
end
helper1=0;
for x=1:N
```

```
   for i=1:N
      helper1=helper1+(v2(x,i)*v2(i,x))-(v2(x,x))^2;
   end
end

E(t)=(A/2)*helper+(B/2)*((sum(sum(v)))-N)^2+(C/2)*helper1;
end %% fin t (si t=T terminer la procédure)

%% Afficher la matrice de sortie sous forme binaire (0 ou 1)
for x=1:N
   for i=1:N+1
      if v(x,i)>0.5
         v(x,i)=1;
      else
         v(x,i)=0;
      end
   end
end
%% Tracez l'énergie E(t) :
figure(1);
plot(E)
```

ANNEXE 4

CODE MATLAB POUR L'EXEMPLE 2 (28 OPERATIONS)

```
%% Initialiser les paramètres :
N=28;
A=200;
B=200;
C=100;
to=1;
b=-100;
c=50;
deltaT=0.0005;
%% Représentation matricielle des bias :
I=[200,b,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
   b,c,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
   b,0,c,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
   b,0,0,c,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
   200,0,0,0,0,b,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
   b,0,0,0,0,c,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
   b,0,0,0,0,0,c,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
   b,0,0,0,0,0,0,c,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
   200,0,0,0,0,0,0,0,0,b,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
   b,0,0,0,0,0,0,0,0,c,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
   b,0,0,0,0,0,0,0,0,0,c,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
   0,0,0,0,0,0,0,0,0,0,0,b,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
   b,0,0,0,0,0,0,0,0,0,0,0,c,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
   b,0,0,0,0,0,0,0,0,0,0,0,0,c,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
   b,0,0,0,0,0,0,0,0,0,0,0,0,0,c,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
   200,0,0,0,0,0,0,0,0,0,0,0,0,0,0,b,b,b,b,0,0,0,0,0,0,0,0,0,0,0;
   b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,c,b,b,b,0,0,0,0,0,0,0,0,0,0;
   b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,c,b,b,0,0,0,0,0,0,0,0,0,0;
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,b,b,b,b,0,0,0,0,0,0,0,0;
   b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,c,b,b,b,0,0,0,0,0,0,0;
   b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,c,b,b,b,0,0,0,0,0,0;
   b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,c,b,b,0,0,0,0,0,0;
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,b,b,b,0,0,0,0,0;
   b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,c,b,b,b,0,0,0,0;
   b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,c,b,b,0,0,0,0;
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,b,b,b,0,0,0;
   b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,c,b,b,0,0,0;
   b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,c,b,b,0,0;
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,b,b,b,0;
   b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,c,b,b;
   b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,c,b];

%% Initialiser aléatoirement l'entrée Uij(t):
u=-2+4*rand(N,N+1);
%% Calculer Vij
for x=1:N
   for i=1:N+1
```

```
      v(x,i)=0.5*(1+tanh(1*u(x,i)));
   end
end
%% boucle principal de calcul :
for t=1:300
%% Mettre l'équation de mouvement pour calculer deltaUij(t) :
D=eye(N+1,1); %% C'est delta(i,1)
DD=eye(N,1); %% C'est delta(x,1)
deltaP=zeros(N,N); %% C'est delta(x,i-1)
deltaM=zeros(N+1,N+1); %% C'est delta(i,x+1)
for i=2:N
   deltaP(i,i-1)=1;
end
deltaP(1,N)=1;
for i=1:N
   deltaM(i,i+1)=1;
end
deltaM(N+1,1)=1;

for x=1:N
   for i=1:N+1
      if i==1
      deltau(x,i)=(-(u(x,i)/to)-A*(sum(v(x,:))-v(x,i))-B*(sum(sum(v))-N)+I(x,i))*deltaT;
      else
      deltau(x,i)=(-(u(x,i)/to)-A*(sum(v(x,:))-v(x,i))-B*(sum(sum(v))-N)-C*v(i-
1,x+1)*(1-D(i,1))*(1-DD(x,1))*(1-deltaP(x,i-1))*(1-deltaM(i,x+1))+I(x,i))*deltaT;
      end
   end
end
%% Calculer Uij(t+1) :
for x=1:N
   for i=1:N+1
      u(x,i)=u(x,i)+deltau(x,i);
   end
end
%% Mettre a jour la sortie :
for x=1:N
   for i=1:N+1
      v(x,i)=0.5*(1+tanh(1*u(x,i)));
   end
end
%% Calculer l'energie
v2=zeros(N,N);
for i=1:N
```

```
   for j=1:N
      v2(i,j)=v(i,j+1);
   end
end
helper=0;
for x=1:N
   for i=1:N+1
      for j=1:N+1
         if i~=j
            helper=helper+(v(x,i)*v(x,j));
         end
      end
   end
end
helper1=0;
for x=1:N
   for i=1:N
      helper1=helper1+(v2(x,i)*v2(i,x))-(v2(x,x))^2;
   end
end

E(t)=(A/2)*helper+(B/2)*((sum(sum(v)))-N)^2+(C/2)*helper1;
end %% fin t (si t=T terminer la procédure)

%% Afficher la matrice de sortie sous forme binaire (0 ou 1)
for x=1:N
   for i=1:N+1
      if v(x,i)>0.5
         v(x,i)=1;
      else
         v(x,i)=0;
      end
   end
end
%% Tracez l'énergie E(t) :
plot(E)
```

ANNEXE 5

CODE MATLAB POUR L'EXEMPLE 3 (50 OPERATIONS)

```
%% Initialiser les paramètres :
N=50;
A=110;
B=110;
C=50
to=1;
b=-100;
a=20;
c=100;
deltaT=0.0005;
%% Représentation matricielle des bias :
I=[c,b,b,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0;
b,a,b,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,a,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,a,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,a,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
c,0,0,0,0,0,b,b,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,a,b,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,a,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,a,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,a,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
c,0,0,0,0,0,0,0,0,0,0,b,b,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,a,b,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,a,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,a,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
c,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,b,b,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
```

b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,b,b,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
c,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,b,b,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,b,b,b,b,b,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,b,b,b,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,b,b,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,b,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,b,b,b,b,b,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,b,b,b,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,b,b,0,0,0,0,0,0,
0,0,0,0;

```
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,b,0,0,0,0,0,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,0,0,0,0,0,0,
0,0,0,0;
c,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,b,b,b,b,b,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,b,b,b,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,b,b,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,b,0,
0,0,0,0;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,b,0,
0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,b,
b,b,b,b;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a,
b,b,b,b;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
a,b,b,b;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,a,b,b;
b,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,a,b];
%% Initialiser aléatoirement l'entrée Uij(t):
u=-2+4*rand(N,N+1);
%% Calculer Vij
for x=1:N
   for i=1:N+1
      v(x,i)=0.5*(1+tanh(1*u(x,i)));
   end
end
%% boucle principal de calcul :
for t=1:800
%% Mettre l'équation de mouvement pour calculer deltaUij(t) :
D=eye(N+1,1); %% C'est delta(i,1)
DD=eye(N,1); %% C'est delta(x,1)
deltaP=zeros(N,N); %% C'est delta(x,i-1)
deltaM=zeros(N+1,N+1); %% C'est delta(i,x+1)
for i=2:N
   deltaP(i,i-1)=1;
end
deltaP(1,N)=1;
for i=1:N
```

```
   deltaM(i,i+1)=1;
end
deltaM(N+1,1)=1;

for x=1:N
   for i=1:N+1
      if i==1
      deltau(x,i)=(-(u(x,i)/to)-A*(sum(v(x,:))-v(x,i))-B*(sum(sum(v))-N)+I(x,i))*deltaT;
      else
      deltau(x,i)=(-(u(x,i)/to)-A*(sum(v(x,:))-v(x,i))-B*(sum(sum(v))-N)-C*v(i-
1,x+1)*(1-D(i,1))*(1-DD(x,1))*(1-deltaP(x,i-1))*(1-deltaM(i,x+1))+I(x,i))*deltaT;
      end
   end
end
%% Calculer Uij(t+1) :
for x=1:N
   for i=1:N+1
      u(x,i)=u(x,i)+deltau(x,i);
   end
end
%% Mettre à jour la sortie :
for x=1:N
   for i=1:N+1
      v(x,i)=0.5*(1+tanh(1*u(x,i)));
   end
end
%% Calculer l'énergie
v2=zeros(N,N);
for i=1:N
   for j=1:N
      v2(i,j)=v(i,j+1);
   end
end
helper=0;
for x=1:N
   for i=1:N+1
      for j=1:N+1
         if i~=j
            helper=helper+(v(x,i)*v(x,j));
         end
      end
   end
end
helper1=0;
```

```
for x=1:N
   for i=1:N
      helper1=helper1+(v2(x,i)*v2(i,x))-(v2(x,x))^2;
   end
end

E(t)=(A/2)*helper+(B/2)*((sum(sum(v)))-N)^2+(C/2)*helper1;
end %% fin t (si t=T terminer la procédure)
%% Afficher la matrice de sortie sous forme binaire (0 ou 1)
for x=1:N
   for i=1:N+1
      if v(x,i)>0.5
         v(x,i)=1;
      else
         v(x,i)=0;
      end
   end
end
plot(E)
```

BIBLIOGRAPHIE

[1] Schonberger R.J.(1986) ; *World class manufacturing*, New York : The Free Press.

[2] Guo R.S., Slama M., Griffin R., Holman K. (1993) ; *A Work Cell Manufacturing System for VLSI Fabrication,* IEEE/CHMT Int'l Electronics Manufacturing Technology Symposium.

[3] Lopez P., Roubellat F. (2001) ; *Ordonnancement de la production* (Traité IC2, série productique), Hermes, Paris, ISBN 2-7462-0184-4.

[4] Dreyfus G., Martinez J.M., Samuelides M., Gordon M.B., Badran F., Thiria S., Hérault L. (2004) ; *Réseaux de neurones. Méthodologie et applications* ; 2ème édition.

[5] Teghem J.,Pirlot M. (2002) ; *Optimisation approchée en recherche opérationnelle*, Recherches locales, réseaux neuronaux et satisfaction de contraintes.

[6] Dagli C.H. (1994) ; *Artificial Neural Networks* for *Intelligent Manufacturing,* London, Angleterre, ISBN 0-412-48050-6, p. 159-193.

[7] Berard F., Azzaro-pantel C., Pibouleau L., Domenech S. (1997) ; *Résolution de Problèmes d'Ordonnancement en Génie des Procédés,* Deuxième Congrès International Franco-Québécois de Génie Industriel.

[8] Hopfield J. (1982) ; *Neural Networks and Physical Systems with emergent collective computational abilities*, Proceedings of National Academy of Sciences of USA, vol. 79, p. 2554-2558, 1982.

[9] Hopfield J. (1984) ; *Neurons with graded response have collective computational properties like those of two-state neurons*, Proceedings of National Academy of Sciences of USA, vol. 81, p.3088-3092.

[10] Hopfield J., Tank D. (1985) ; *Neural computation of decisions in optimization problems*, Biological Cybernetics, vol. 52, p.141-152.

[11] Hanada A., Ohnishi K., IEEE (1993) ; *Near Optimal Job shop Scheduling Using Neural Network Parallel Computing*

[12] Foo S. Y-P. , Takefuji Y. (1988) ; *Stochastic neural networks for solving Job-Shop scheduling* : Part 1. Problem Representation, International Joint Conference on Neural Networks, IEEE Vol. 2, p.275-282.

[13] Foo S. Y-P. , Takefuji Y. (1988) ; *Stochastic neural networks for solving Job-Shop scheduling* : Part 2. Architecture and simulation, International Conference on Neural Networks, IEEE TAB : II283-II290.

[14] Takefuji Y. (1992) ; *Neural Network Parallel Computing*, Kluwer Academic publishers

[15] Xin-li X. , Wan-liang W. IEEE(2001) ; *An improved neural networks with transient Chaos Method for Job-Shop scheduling problems,* Proceedings of the 4th World Congress On Intelligent Control and Automation.

[16] Ayer S.V.B. et al. (1990) ; *A theoretical investigation into the performance of the Hopfield model,* IEEE Transactions on Neural Networks, vol.1, p.204-215.

[17] Gonçalves J-F. , Magalhaes Mendes J-J. , Resende M-G-C. (2002) ; *A Hybrid Genetic Algorithm for the Job Sop Scheduling Problem,* AT & T Labs Research Technical Report TD-5EAL6J.

[18] Gagné R. (2002) ; *Outil d'aide à la décision pour la conception de systèmes manufacturiers cellulaires*, Mémoire de Maîtrise en Génie de la Production Automatisée, École de Technologie Supérieure, QC.

[19] Erwin Kreyszig (1999) ; *Advanced engineering mathematics*, 8ème edition p. 942-945

[20] Nourelfath M., Nahas N. (2003) ; *Quantized Hopfield Networks for Reliability Optimization*, Elsevier Science Ltd.

[21] Nowicki E. and Smutnicki C. (1996) ; *A fast taboo search algorithm for the job shop problem*, Management Science, p. 797-813.

[22] Ackley D.H., Hinton G.E., Sejnowski T.J. (1985) ; *A learning algorithm for Boltzmann machines*, Cognitive Science,9, p. 147-169.

[23] Metropolis N., Rosenbluth A., Rosenbluth M., Teller A., Teller E. (1953), *Equation of state calculations by fast computing machines*, Journal of Chemical Physics, vol. 21, p. 1087-1092.

[24] Sue Becker (2003) ; *Neural Computation*, Notes de cours NEURCOMP3W03, McMaster university.

[25] Oliver B. Downs (2000) ; *The Boltzmann Machine*, Hopfield Group, Princeton University,

[26] Hebb D.(1949); *The Organization of Behavior*, Wiley, New York.

[27] Rosenblatt F. (1958);*The perceptron : a probabilistic model for information storage and organization in the brain,* Psychological Review, vol.65, p.386-408.

[28] Minsky M. et Papert S. (1969); *Perceptrons*, MIT Press, Cambridge.

[29] Rumelhart D., Hinton G. et Williams R. (1986); *Learning representations by back-propagating errors*; Nature, 323 p. 533-539.

[30] Jain A. S., Rangaswamy B. and Meeran S. (2000); *New and "Stronger" Job-Shop Neighbourhoods : A Focus on the Method of Nowicki and Smutnicki (1996)*; Journal of Heuristics, Vol. 6, p.457–480

[31] Das H., Cummings P.T., LeVan M.D. (1990); *Scheduling of serial multiproduct batch processes via simulated annealing*, Comput. Chem. Engng, Vol. 14, p.1351-1362

[32] Gotha (1993) ; *Les problèmes d'ordonnancement*, RAIRO Vol. 27, p.77-150

[33] Caux C., Pierreval H., Portmann M.C. (1994) ; *Les algorithmes génétiques et leur application aux problèmes d'ordonnancement*, Proceedings Journées d'études « ordonnancement et entreprise », Toulouse .

[34] Ben saleh Z. (2000) ; *Conception et mise en œuvre d'un systeme de base de connaissance pour l'ordonnancement dynamique de gestion de priorités basé sur l'approche algorithmique*, Mémoire de Maîtrise en Génie mécanique, École de Technologie Supérieure, QC.

[35] Blackstone J., Phillips D., and Hogg G. (1982); *A state-of-the-art survey of dispatching rules for manufacturing job shop operations.* International Journal of Production Research, p.27-45

[36] Van Laarhoven, P.J.M., E.H.L. Aarts, and J.K. Lenstra. (1988); *Job Shop Scheduling by Simulated Annealing.* Report OS-R8809, Centrum voor Wiskunde en Informatica, Amsterdam.

[37] Van Laarhooven, P.J.M., E.H.L. Aarts, and J.K. Lenstra. (1992); *Job Shop Scheduling by Simulated Annealing.* Operations Research 40(1), 113-125.

[38] Grabowski J., Nowicki E., and Smutnicki C. (1988); *Block Algorithm for Scheduling Operations in a Job-Shop System.* Przeglad Statystyczny XXXV, Vol.1, p.67-80

[39] Matsuo H., Suh C.J. et Sullivan R.S. (1988). *A Controlled Search Simulated Annealing Method for the General Job-Shop Scheduling Problem.* Graduate School of Business, The University of Texas at Austin, Austin, Texas, USA.

[40] Radharamanan R. (1986). *A Heuristic Algorithm for Group Scheduling.* Proceeding of the 8th Annual Conference on Computers and Industrial Engineering, p.204-208

[41] Hohmann C. (2004). *La flexibilité des entreprises* http://membres.lycos.fr/hconline/flexibilite3.htm (Page consultée le 28 novembre 2004)

[42] Nollet J., Kélada J. et Diorio M.O. (1994). *La gestion des opérations et de la production* ; 2ème édition.