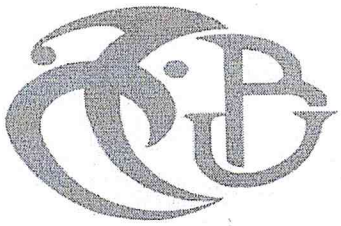
République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Saad Dahlab Blida

N° D'ordre :.....



Faculté des sciences

Département d'informatique

Mémoire Présenté par :

BOUSSAID Imane CHADI Zineb

En vue d'obtenir le diplôme de master

Domaine : Mathématique et informatique
Filière : Informatique
Spécialité : Informatique
Option : Génie des systèmes informatique

Sujet :

Application neuro-Evolutionnaire pour la simulation dans les systèmes de production automatisés

Soutenu le :

M. Abed		Président
M. Guessoum.		Examinateur
M Aroussi		Examinateur
Mme.	A. Miloud Aouidate	Promotrice
M.	Gaham Mehdi	Encadreur

Promotion
2014 / 2015

Remerciement

*Tout d'abord, louange à « **Allah** » qui nous a guidés sur le droit chemin tout au long du travail et nous a inspirés les bons pas et les justes reflexes. Sans sa miséricorde, ce travail n'aura pas abouti.*

Nous remercions nos parents respectifs qui, par leur amour et leur affection nous ont permis d'arriver là où nous somme aujourd'hui.

Nous remercions notre promotrice Mme. Miloud Aouidate pour son aide précieuse, ces conseils avisés et ses idées riches.

Nos vifs remerciements vont à notre encadreur GAHAM Mahdi pour son soutien et son encouragement et son précieux temps qu'il nous a accordé

Nous tenons un grand et un spécial remerciement Nabil pour son sympathie, son aide et ses encouragements.

Nous remercions le membre de jury pour nous avoir fait l'honneur de juger notre travail.

Nous tenons à remercier également et énormément nos amis ainsi toute personne qui nous a aidés de près ou de loin.

Merci.

Imane & Zineb

Résumé

Dans le cadre de la gestion d'un système de production automatisé, nous aimerions savoir quel est le meilleur scénario en ce qui concerne le nombre de produits et l'ordonnancement. Sachant que notre critère principal est le temps d'exécution dont notre objectif est de le minimalisé et donc gagné en productivité.

Notre projet propose donc une méthode d'aide à la décision par la simulation en temps réel des systèmes de production en utilisant l'approche neuro-évolutionnaire dans le cadre du pilotage des systèmes de production automatisés. Pour cela nous avons commencé par définir les systèmes de production et leurs pilotages ainsi que le principe de la simulation qui aide l'utilisateur à prendre la meilleure décision.

Dans ce but nous nous somme tourner vers NEAT (Neuro Evolution of Augmenting Topologies), comprendre son fonctionnement et comment l'utilisé comme une approche pour la simulation.

Abstract

As part of managing an automated production system, we want to know the optimal number of products and scheduling. Knowing that our main focus is the minimization of execution time in order to increase productivity.

Our project proposes a methodology for decision support employing a real-time simulation of production systems involving a neuro-evolutionary approach in the management of automated production systems.

To do this, we first define production systems and their piloting alsothe simulation standard that helps the user to make the best decision. For this purpose, we apply NEAT (Neuro Evolution of Augmenting Topologies) we enhance our understanding of the system and employ the approach for simulation.

ملخص

نسعى، في إطار دراسة تسيير أنظمة الإنتاج الآلية إلى تحديد خطة الإنتاج المثلى من حيث الكمُ المُنتَج و تنظيم مراحل الإنتاج، علما أن مدة الإنجاز هي معيارنا الأساسي، و هدفنا هو تخفيضها و منه رفع الإنتاجية.

كما نهدف من خلال بحثنا هذا إلى اقتراح خطة مساعدة لاتخاذ القرار عن طريق المحاكاة الفعلية لأنظمة الإنتاج، موظفين من أجل ذلك مقاربة عصبية تطورية لتسيير أنظمة الإنتاج الآلية , فقد قمنا أولا بتحديد أنظمة الإنتاج و طرق تشغيلها و التحكم فيها ثم حددنا المبدأ الذي تقوم عليه عملية المحاكاة المساعدة على اتخاذ القرارات المثلى.

ليكون محل الدراسة NEAT) NeuroEvolution of Augmenting Topologies) اعتمدنا لأجل تحقيق ذلك نظاما بغية فهم كيفية اشتغاله و للنظر بعد ذلك في طريقة توظيفه كمقاربة لبناء خطة محاكاة أنظمة الإنتاج الآلية.

Sommaire

Liste des figures

Liste des tableaux

INTRODUCTION GÉNÉRALE

Introduction générale

Les responsables d'entreprise sont actuellement confrontés à un fort accroissement de la complexité de la production. Accroître la productivité en réduisant les coûts est, aujourd'hui, un objectif majeur dans toutes les entreprises, les systèmes de production sont caractérisés par leur dynamique et leur imprévision, les tâches souvent de caractère complexe et sont soumises à des contraintes de temps et d'exigence. Alors, le système de production se donne de nouveaux objectifs à atteindre.

Les recherches entreprises ces dernières années dans le cadre du pilotage des systèmes de production fortement automatisés et intelligent donnent une grande place à l'utilisation des techniques de l'intelligence artificielle. Pour ce qui est du pilotage tactique (conduite opérationnelle du système de production) ces techniques et particulièrement les technique d'apprentissage artificiel tel que les réseaux de neurones sont généralement utilisées pour la prise des décisions temps réel d'ordonnancement, d'affectation ou de séquencement ou en tant que modèle de substitution pour les taches de simulation. Le système de simulation offre principalement une évaluation des décisions possibles à la résolution d'un problème donné mais il reste généralement couteux en temps de calcul ou parfois même non disponible. D'où la nécessité de le remplacé par un réseau de neurones.

L'objectif principal de notre travail consistera principalement en la conception et l'apprentissage du réseau et sa mise en œuvre dans le cadre d'une application d'aide à la décision.

Ce mémoire est divisé en quatre chapitres :

Le premier chapitre est consacré à une revue bibliographique centrée sur les définitions essentielles de pilotage des systèmes de production automatisé et l'évaluation en temps réel des règles d'ordonnancement par la simulation pour l'aide à la décision.

Le deuxième chapitre présent aussi des recherches bibliographiques sur la conception des réseaux de neurone et leur apprentissage, nous abordons par la suite les algorithmes génétiques et le principe des algorithmes évolutionnaire ce qui nous amène au principe de neuro-évolution plus précisément l'approche neuro-évolutionnaire qui est NEAT « Neuro Evolution of Augmenting Topologies» que nous allons utiliser dans notre projet.

Dans le troisième chapitre, nous décrivons, d'une manière détaillée, l'approche neuro-évolutionnaire (NEAT) implémentée, commençant par le traitement des données et son stockage jusqu'à l'arrivé de leur validation dans l'expérimentation.

Dans le quatrième chapitre, nous présentons l'implémentation de notre application ainsi que les tests effectués et les résultats obtenus pour prouver l'efficacité de l'approche neuro-évolutionnaire pour l'aide à la décision par la simulation dans le cadre du pilotage des systèmes de production automatisé

CHAPITRE 1

Pilotage des systèmes de production

Chapitre 1

1.1 Introduction :

La mise en œuvre du pilotage des systèmes de production constitue un enjeu sensible. Par exemple pour rester compétitives sur des marchés de plus en plus incertains, les entreprises ont besoin d'être réactives. Elles doivent souvent faire face à des événements imprévus tels qu'une annulation ou une modification de commande, la prise en compte d'une commande urgente, des aléas du système de production, etc. Ceci nécessite d'avoir des outils de pilotage de la production capables de réagir face aux événements critiques et entraîne la recherche de performances de plus en plus élevées des systèmes de pilotage et des méthodes qui leur sont associées.

L'évolution constatée dans ce domaine depuis une vingtaine d'années est considérable, tant par l'augmentation de la finesse des objectifs poursuivis que par la sophistication des systèmes informatiques qui supportent le pilotage ou encore par la variété des solutions mises en œuvre pour optimiser l'utilisation de ressources de production de plus en plus coûteuses.

1.2 Système automatisé de production

Un système automatisé de production(SAP) est un système réalisant des opérations et pour lequel l'homme n'intervient que dans la programmation du système et dans son réglage. Les buts d'un système automatisé sont de réaliser des tâches complexes ou dangereuses pour l'homme, effectuer des tâches pénibles ou répétitives ou encore gagner en efficacité et en précision.

L'automatisation présente les principaux avantages suivants:

- elle permet de gérer de manière efficace la production,
- elle permet la création d'un suivi performant
- L'automatisation de la production permet de diminuer les temps de réponse et d'augmenter la flexibilité de la structure de production (NOYES, 1987).

1.3 Composition d'un SAP

On peut distinguer trois parties dans un système automatisé (ROHEE, 2005).

- La Partie Commande. P.C.
- L'interface.
- La Partie Opérative. P.O.

❖ **La Partie Opérative : PO**

Il s'agit de la partie qui effectue le travail. C'est elle qui reçoit les ordres de la Partie Commande (P.C.). Dans la PO c'est les actionneurs (des vérins ou des moteurs) qui exécutent les ordres reçus. Elle renvoie à la partie commande des informations sur son état ou sur l'environnement. Sa fonction globale est d'apporter de la valeur ajoutée à la matière première.

❖ **La Partie Commande : PC** (le système de contrôle)

La Partie Commande regroupe l'ensemble des composants permettant le traitement des informations reçues de la PO. Elle élabore les ordres à partir de ces informations et informe l'opérateur de l'état du système. Pilote le fonctionnement du système automatisé.

❖ **L'interface :** (l'opérateur (surveillance, marche/arrêt))

Elle relie la PO et la PC. C'est un système de traduction d'informations entre la Partie Commande PC et la Partie Opérative PO. C'est une interface homme machine (IHM). Elle est équipée d'organes permettant :

- La mise en/hors énergie de l'installation.
- La sélection des modes de marche.
- La commande manuelle des actionneurs.
- Le départ des cycles de fonctionnement.
- L'arrêt d'urgence.
- D'informer l'opérateur de l'état de l'installation.

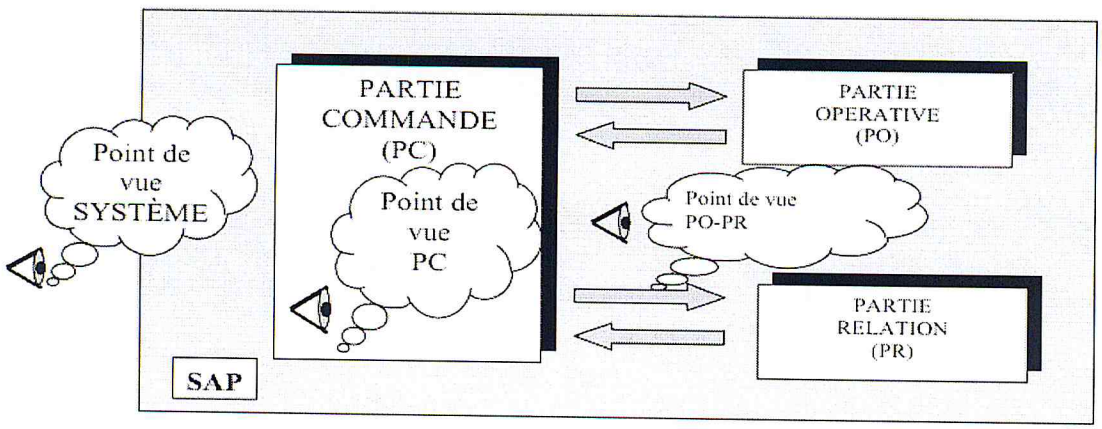


Figure 1: Structure d'un système de production automatisé.

1.4 Gestion d'un système automatisé de production :

De façon générale, la gestion d'un SAP comporte deux niveaux (TRENTESAUX, 1996) :

1.4.1 La gestion prévisionnelle (niveau 1):

Elle assure l'anticipation et la programmation d'un ensemble d'actions ou de décisions. La complexité et l'inertie du système de production conduisent à mettre en œuvre une gestion prévisionnelle.

1.4.1.1 La complexité du système de production :

Plusieurs éléments sont à l'origine de cette complexité. Notons principalement:

- la diversité des types des ressources composant le système en ressources informatiques, mécaniques, électriques ou humaines. La gestion d'un ensemble de ressources hétérogènes est rendue difficile notamment par les différents niveaux nécessaires de connaissance, d'intelligence et les diverses possibilités de communication entre ces ressources.
  • la multiplicité et parfois la contradiction entre les objectifs de production en terme de coût, délai ou qualité. Par exemple, la production organisée suivant un objectif global de qualité impliquera certainement un rendement global plus faible qu'une production organisée selon un objectif portant sur la minimisation des délais.
- la complexité des calculs prévisionnels qu'il est nécessaire la plupart du temps d'effectuer hors ligne.

1.4.1.2 *L'inertie du système de production :*

L'inertie est due principalement aux composantes mécaniques et humaines du système de production. Elle est définie en termes de temps de production, de changement d'outil, de calendriers (postes de travail), etc. Ainsi, la gestion prévisionnelle permet l'intégration de cette inertie par la planification de certaines actions au niveau du système de production. Ce niveau peut être décomposé de manière générale en deux ou trois fonctions selon:

- la structure de pilotage adoptée,
- la présence ou non d'un ordonnancement prévisionnel,

1.4.2 la gestion temps réel (niveau 2):

Elle élabore les décisions et les actions qui sont réalisées en temps-réel et qui sont donc déclenchées par un ensemble d'événements liés à l'état courant du système de production.

Cette gestion se justifie par:

- l'altération des données sur lesquelles sont effectuées les prévisions du niveau. le système de production est soumis à des perturbations, à des modifications structurelles et/ou de son environnement.
- l'accessibilité, la fiabilité ou l'agrégation des données due au volume important d'informations à traiter et parfois contradictoires. (TRENTESAUX, 1996)

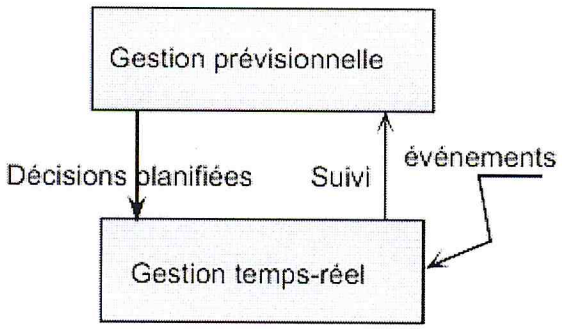


Figure 2: interactions entre les deux niveaux de gestion d'un SAP.

2 L'ordonnancement dans un SAP :

Ordonnancer, c'est programmer dans le temps l'exécution des tâches à réaliser sur une ressource particulière (comme une machine) ou groupe de ressources (comme un atelier). L'ordonnancement fournit donc un calendrier d'organisation du travail pour l'atelier fixant les dates de début et de fin de chaque tâche (MIRDAMADI, 2009).

2.1 Objectifs de l'ordonnancement

L'ordonnancement est un des problèmes majeurs qui se pose au niveau du pilotage d'un atelier de production Le traitement de l'ordonnancement dans la littérature s'est tout d'abord orienté vers une optimisation monocritère. L'environnement manufacturier évoluant rapidement et la concurrence devenant de plus en plus acharnée, les objectifs des entreprises se sont diversifiés et le processus d'ordonnancement est devenu de plus en plus multicritère. Les critères que doit

satisfaire un ordonnancement sont variés. D'une manière générale, on distingue plusieurs classes d'objectifs concernant un ordonnancement (HOUARI, 2012).

- ***Les objectifs liés au temps*** : On trouve par exemple la minimisation du temps total d'exécution, du temps moyen d'achèvement, des durées totales de réglage ou des retards par rapport aux dates de livraison.

- ***Les objectifs liés aux ressources*** : maximiser la charge d'une ressource ou minimiser le nombre de ressources nécessaires pour réaliser un ensemble de tâches sont des objectifs de ce type.

- ***Les objectifs liés au coût*** : ces objectifs sont généralement de minimiser les coûts de lancement, de production, de stockage, de transport, etc (SOUIER, 2012).

2.2 Ordonnancement en temps réel :

Le problème d'ordonnancement consiste à organiser dans le temps la réalisation de tâches, compte tenu de contraintes temporelles et des contraintes portant sur l'utilisation et la disponibilité des ressources requises

- ***Une tâche*** est une activité dont l'exécution peut demander plusieurs opérations ; les tâches et les opérations sont liées par des relations de précédences, selon un ordre total pour les fabrications linéaires ou selon un ordre partiel pour les cas d'assemblage.
- ***Une ressource*** est un moyen technique, financier ou humain destiné à être utilisé pour la réalisation d'une tâche et disponible en quantité limitée (MIRDAMADI, 2009).

Les problèmes d'ordonnancement d'atelier sont des problèmes avec ressources disjonctives, car les ressources principales sont les machines, ne pouvant réaliser qu'une seule opération à la fois et chaque opération concerne un produit ou un lot (plusieurs produits identiques sont regroupés). On distingue trois grands types de problèmes : l'ordonnancement dans un atelier à cheminement unique (flow-shop), l'ordonnancement de divers types de cellules flexibles, (open-shop), l'ordonnancement dans un atelier à cheminements multiples (job-shop) sur lequel on va travailler.

2.2.1 Atelier a acheminement unique : le flow shop :

Dans le cas des organisations de type Flow Shop Simple (Hentous & Guinet, 1997)les tâches ont un même cheminement. Une tâche est ainsi constituée d'opérations visitant différentes machines et enchaînées de manière linéaire suivant une chaîne (Esquirol & LOPEZ, 1999)

Ce sont des ateliers où une ligne de fabrication est constituée de plusieurs machines en série et où toutes les opérations de toutes les tâches passent par les machines dans le même ordre. Autrement dit, on a *m* machines en série et chaque tâche doit être exécutée sur chacune de ces machines.

Ainsi, toutes les tâches doivent s'exécuter sur la machine 1, puis sur la machine 2,…, et ainsi de suite dans le même ordre. Après la fin d'exécution sur une machine, la tâche rejoint la file d'attente de la machine suivante, et ne passe qu'une seule fois sur la machine.

2.2.2 Atelier a acheminement libre : open shop :

Dans les organisations de type Open Shop Simple ou Hybride, les tâches ont une gamme non précisée (libre), autrement dit il n'existe aucun ordre d'exécution des opérations. Dans les problèmes d'ordonnancement de production dits à cheminement libre (ou sans contraintes d'enchaînement), ce type d'atelier est moins contraint que celui de type flow shop ou de type job shop.

Les problèmes d'ordonnancement consistent, d'une part, à déterminer le cheminement de chaque tache et d'autre part à ordonnancer les taches en tenant compte des gammes trouvées. Chaque produit à fabriquer doit subir diverses opérations sur des machines, mais dans un ordre totalement libre (Esquirol & LOPEZ, 1999).Open shop n'est pas couramment utilisé dans les entreprises.

2.2.3 Atelier a acheminement multiple : Job shop :

Les ateliers à cheminement multiple (Job-shop) sont des unités manufacturières traitant une variété de produits individuels dont la production requiert divers types de machines dans des séquences variées (TAGHEZOUT, 2011). Les opérations sont réalisées selon un ordre total bien déterminé, variant selon la tâche à exécuter.

Contrairement au type d'atelier précédent, l'atelier job shop se caractérise par un cheminement multiple puisque les opérations de chaque job peuvent emprunter divers chemins.

L'objectif le plus considéré dans le cas d'un atelier à cheminements multiples est le même que celui considéré pour un atelier à cheminement unique. L'objectif est de trouver une

séquence de tâches sur les machines qui minimise le temps total de production (ESQUIROL & LOPEZ, 1999).

La figure suivante représente un atelier de type job shop :

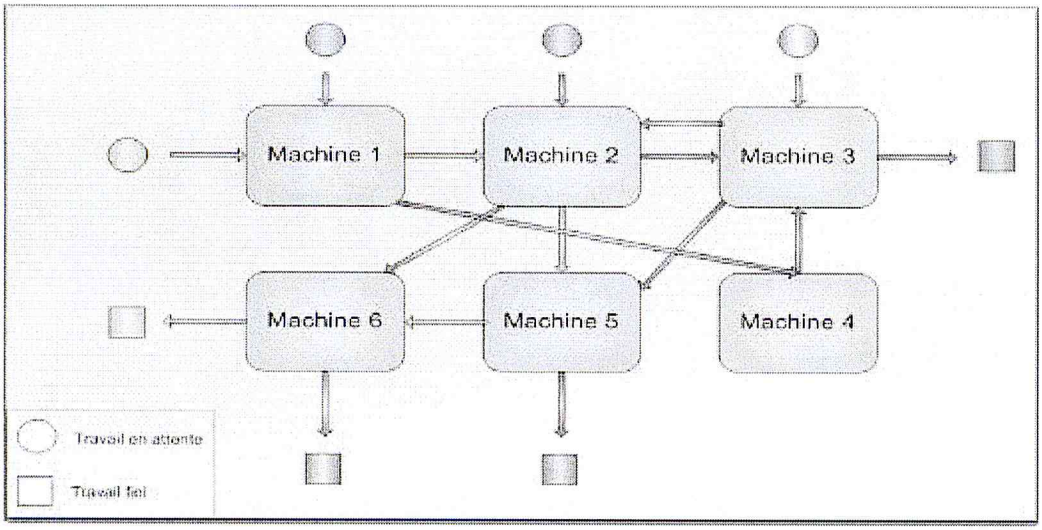


Figure 3 : Organisation d'atelier à cheminements multiples (Job Shop)

2.3 L'utilisation des règles de priorité dans le pilotage des SAP :

Les algorithmes les plus utilisés, notamment dans les logiciels du marché, reposent sur des règles de priorité. Lorsqu'une machine devient disponible et qu'il existe plusieurs opérations en attente d'exécution sur cette machine, on doit choisir l'opération à effectuer. Ce choix se fait selon une règle dite de priorité. Parmi les règles de priorité les plus courantes en ordonnancement, on trouve les règles basées sur des critères portant sur les délais, les durées opératoires, les dates d'arrivées, la taille des files d'attentes, le nombre d'opérations restant, etc. Une règle appliquée en ordonnancement peut être constante (règle statique) ou évoluer en fonction de l'environnement (règle dynamique).

3 Évaluations par simulation dans les SAP :

Dans le contexte d'un atelier de production, la durée d'exécution des opérations peut varier et les machines ne sont pas constamment disponibles pour raison de maintenance, pannes ou réparations. Ces informations peuvent être modélisées par des distributions de probabilités ou par une loi aléatoire. Pour répondre à ces problèmes d'ordonnancement, l'approche par simulation est très souvent envisagée comme une manière simple et rapide de concevoir des solutions d'ordonnancement. La simulation offre aussi la possibilité de tester un nombre

important de règles de priorité ou même les effets des modifications de paramètres sur les résultats obtenus dans un cadre déterministe (SOUIER, 2012).

La simulation donne la possibilité de jouer avec les différents paramètres (par exemple : dimension des aires de stockage, règle de priorité sur les postes…ect) pour trouver une bonne solution.

4 L'aide à la décision :

4.1 Définition d'un système d'aide à la décision :

C'est un système informatique intégré, conçu spécialement pour la prise de décision, et qui est destiné plus particulièrement aux dirigeants d'entreprise. Le système d'aide à la décision est un des éléments du système d'information de gestion. Il se distingue du système d'information pour dirigeants, dans la mesure où sa fonction première est de fournir non seulement l'information, mais les outils d'analyse nécessaires à la prise de décision. Ainsi, il est habituellement constitué de programmes, d'une ou de plusieurs bases de données, internes ou externes, et d'une base de connaissances. Il fonctionne avec un langage et un programme de modélisation qui permettent aux dirigeants d'étudier différentes hypothèses en matière de planification et d'en évaluer les conséquences.

4.1.1 La décision :

La décision est souvent vue comme le fait d'un individu isolé (« le décideur ») exerçant un choix entre plusieurs possibilités d'actions à un moment donné. Aider à décider, c'est tout d'abord aider à clarifier la formulation, la transformation et l'argumentation des préférences.

4.1.2 L'aide à la décision :

L'aide à la décision est définit comme étant l'activité de celui qui, prend appui sur des modèles clairement explicités mais non nécessairement formalisés, et aide à obtenir des éléments de réponse aux questions que se pose un acteur dans un processus de décision, l'éléments concourant à éclairer la décision et normalement à recommander, ou simplement à favoriser, un comportement de nature à accroître la cohérence entre l'évolution du processus d'une part, les objectifs et le système de valeurs au service desquels ces acteurs se trouvent placés d'autre part (YOUNES, 2011).

4.2 Objectif des systèmes décisionnel :

Les systèmes décisionnel doit permettre d'avoir une vision claire, nette et précise du passé, mais aussi de comprendre la situation actuelle et de prévoir, simuler le futur. Les systèmes décisionnels doivent être une aide aux collaborateurs, leur permettant d'être proactifs sur leurs marchés, c'est-à-dire d'analyser, d'anticiper et de capitaliser sur l'expérience pour décider en fonction d'informations disponibles facilement et rapidement.

Un outil décisionnel ne doit pas être un simple outil de contrôle mais bien un outil de partage de la connaissance, de communication, de prévision et de simulation pour l'analyse et l'amélioration de la performance des organisations.

5 Conclusion

Les entreprises ont de plus en plus besoins des systèmes d'aide à la décision en utilisant la simulation temps réel. Certes il existe des très bons outils pour cela par exemple Flexsim, mais ils sont très complexe par son utilisation et coute assez cher. Pour cela nous avons pensé à mettre en œuvre notre propre produit pour la simulation et ainsi l'aide à la décision qui sera simple à utiliser et gratuit.

CHAPITRE 2

Les méthodes de résolution

Chapitre 2

1 Les réseaux de neurones :

1.1 Introduction :

Le rêve de créer une machine dotée d'une forme d'intelligence est présent depuis longtemps dans l'imagination humaine. Alors comment l'homme fait-il pour raisonner, calculer, parler, apprendre, … ? C'est ces questions-là qui mènent les chercheurs à essayer de comprendre le fonctionnement du cerveau humain et essayer de s'y inspirer pour pouvoir trouver de nouvelles techniques de résolutions de problèmes dans le monde informatique. L'intelligence artificielle a apparue et ne cesse de progresser, il existe de nombreux programmes capables de diriger un robot, résoudre des problèmes etc. Néanmoins ils ne sont pas capables de rivaliser avec un cerveau humain. Outre la capacité de calcul incroyable des ordinateurs, mais ces derniers n'ont pas la faculté d'apprendre. Ils ne progressent pas si personne ne les modifie. Voilà ce à quoi les chercheurs ont essayé de remédier. Avec l'avancée dans le domaine de la neurobiologie concernant le fonctionnement du cerveau et des neurones, des mathématiciens ont essayé de modéliser le fonctionnement du cerveau en intégrant ces connaissances en biologie dans des programmes informatiques pour leur donner la possibilité d'apprendre : c'est la naissance des réseaux de neurones.

Les réseaux de neurones ont d'abord été développés pour résoudre des problèmes de contrôle, de reconnaissance de formes ou de mots, de décision, de mémorisation comme une alternative à l'intelligence artificielle, et en relation plus ou moins étroite avec la modélisation de processus cognitifs (capable de connaître ou faire connaître) réels et des réseaux de neurones biologiques. Les réseaux de neurones sont composés d'éléments simples (ou neurones) fonctionnant en parallèle. Ces éléments ont été fortement inspirés par le système nerveux biologique. Comme dans la nature, le fonctionnement du réseau (de neurone) est fortement influencé par la connections des éléments entre eux.

1.2 Historique :

La première modélisation d'un neurone date de 1943. Elle a été présentée par McCulloch et Pitts. L'interconnexion de ces neurones permet le calcul de plusieurs fonctions logiques. En 1949, Hebb propose le premier mécanisme d'évolution des connections, appelées par analogie des synapses. L'association de ces deux méthodes permit à Rosenblatt en 1958 de décrire le premier modèle opérationnel de réseaux de neurones : le perceptron. Celui-ci est capable d'apprendre à calculer un grand nombre de fonctions booléennes, mais pas toutes. Ses limites théoriques furent mises en évidence par Minsky et Papert en 1969. Depuis 1985, de nouveaux

modèles mathématiques ont permis de les dépasser. Cela a donné naissance au perceptron multicouche. (AMMAR Yessin, 2007)

1.3 Définition :

Les réseaux de neurones sont composés d'éléments simples (ou neurones) fonctionnant en parallèle. Ces éléments ont été fortement inspirés par le système nerveux biologique. Comme dans la nature, le fonctionnement du réseau (de neurone) est fortement influencé par la connections des éléments entre eux. On peut entraîner un réseau de neurone pour une tâche spécifique (reconnaissance de caractères par exemple) en ajustant les valeurs des connections (ou poids) entre les éléments (neurone).

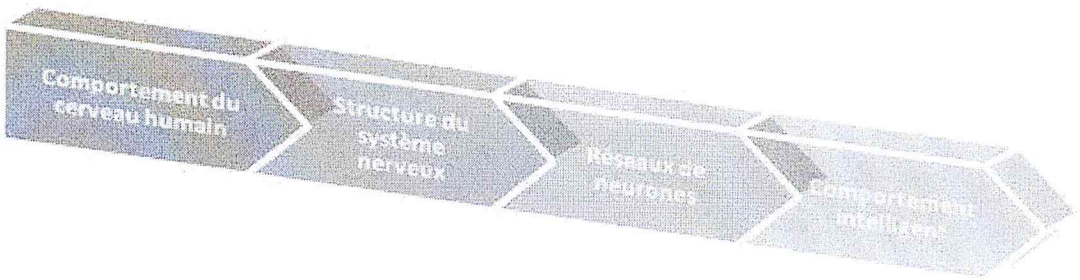


Figure 1:Creation et inspiration des réseaux de neurones

1.3.1 Le neurone biologique :

Le système nerveux compte plus de 1000 milliards de neurones interconnectés. Bien que les neurones ne soient pas tous identiques, leur forme et certaines caractéristiques permettent de les répartir en quelques grandes classes. En effet, il est aussi important de savoir, que les neurones n'ont pas tous un comportement similaire en fonction de leur position dans le cerveau. Dans le cerveau, les neurones sont reliés entre eux par l'intermédiaire d'axones et de dendrites. On peut considérer que ces sortes de filaments sont conductrices d'électricité et peuvent ainsi transporter des messages depuis un neurone vers un autre. Les dendrites représentent les entrées du neurone et son axone sa sortie (REBOUH, 2011).

1.3.2 Neurone formel (artificiel) :

Le neurone formel est une abstraction de la réalité biologique. Il synthétise mathématiquement toutes les informations apportées par les observations d'un vrai neurone à l'exception du critère de temporalité qu'il n'est pas utile de modéliser puisque les valeurs manipulées par les neurones représenterons les fréquences d'émission des stimuli.

Le neurone formel, comme son homologue biologique, a donc pour unique fonction de transmettre un influx nerveux sous certaines conditions. Pour cela, il possède une ou plusieurs entrées (les dendrites) et une sortie (l'axone). Le corps quant à lui ne sert qu'à évaluer si le signal reçu sur les entrées doit être propagé sur la sortie ou non (PANZOLI, 2003).

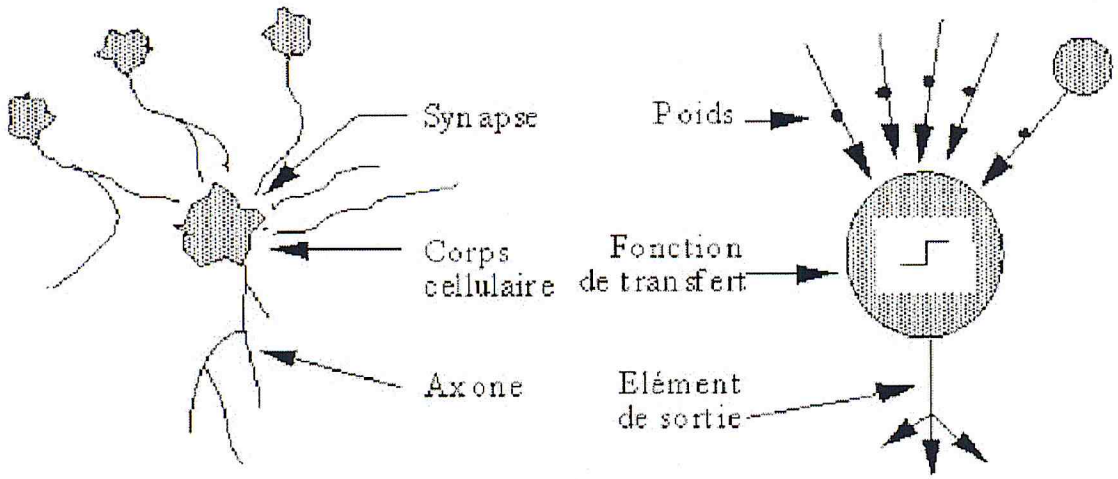


Figure 2:neurone biologique et neurone artificiel

1.4 Du neurone biologique au neurone formel :

Les réseaux de neurones formels sont à l'origine d'une tentative de modélisation mathématique du cerveau humain. Un neurone est essentiellement constitue d'un intégrateur qui effectue la somme pondérée de ses entrées. Le résultat n de cette somme est ensuite transformée par une fonction d'activation f qui produit la sortie a du neurone. Il existe en pratique plusieurs fonctions d'activation classées suivant les valeurs manipulées par le neurone (fonction linéaire, fonction signe, fonction sigmoïde … ect).

La modélisation consiste à mettre en œuvre un système de réseau neuronal sous un aspect non pas biologique mais artificiel, cela suppose que d'après le principe biologique on aura une correspondance pour chaque élément composant le neurone biologique, donc une modélisation pour chacun d'entre eux (GOSSELIN, 1995).

On peut résumer la modélisation du neurone formel à partir du neurone biologique à l'aide du tableau suivant :

Neurone biologique	Neurone artificiel
Axones	Signal de sortie
Dendrites	Signal d'entrée
Synapses	Poids de la connexion

Tableau 1: Modalisation de neurone formel

1.5 Procédure de développement d'un réseau de neurone :

Le cycle classique de développement peut être séparé en sept étapes (CHIKH, 2000) :

1. **la collecte des données :**

L'objectif de cette étape est de recueillir et rassembler un nombre de données suffisants susceptibles d'intervenir en phase d'utilisation du système neuronal. Ces données sont subdivisées en deux, une pour développer le réseau de neurones (l'apprentissage) et l'autre pour le tester.

2. **l'analyse des données :**

Il est souvent préférable d'effectuer une analyse des données de manière à déterminer les caractéristiques discriminantes pour détecter ou différencier ces données. Ces caractéristiques constituent l'entrée du réseau de neurones.

Cette détermination des caractéristiques a des conséquences à la fois sur la taille du réseau (et donc le temps de simulation), sur les performances du système (pouvoir de séparation, taux de détection), et sur le temps de développement (temps d'apprentissage).

3. **la séparation des bases de données :**

Afin de développer une application à base de réseaux de neurones, il est nécessaire de disposer de deux bases de données : une base pour effectuer l'apprentissage et une autre pour tester le réseau obtenu et déterminer ses performances. I l n'y a pas de règle pour déterminer ce partage de manière quantificatif.

4. **le choix d'un réseau de neurones :**

Il existe un grand nombre de types de réseaux de neurones, avec pour chacun des avantages et des inconvénients. Le choix d'un réseau peut dépendre :de la tâche à effectuer (classification, association, contrôle de processus, séparation aveugle de sources...), de la nature des données (dans notre cas, des données présentant des variations au cours du temps), d'éventuelles contraintes d'utilisation temps-réel (temporelles).

5. **la mise en forme des données :**

De manière générale, les bases de données doivent subir un prétraitement afin d'être adaptées aux entrées et sorties du réseau de neurones. Un prétraitement courant consiste à effectuer une normalisation appropriée, qui tienne compte de l'amplitude des valeurs acceptées par le réseau.

6. **l'apprentissage :**

Tous les modèles de réseaux de neurones requièrent un apprentissage. Plusieurs types d'apprentissages peuvent être adaptés à un même type de réseau de neurones. Les critères de choix sont souvent la rapidité de convergence ou les performances de généralisation. Le critère d'arrêt de l'apprentissage est souvent calculé à partir d'une fonction de coût, caractérisant l'écart entre les valeurs de sortie obtenues et les valeurs de références (réponses souhaitées pour chaque exemple présenté).

7. **la validation :**

Une fois le réseau de neurones entraîné (après apprentissage), il est nécessaire de le tester sur une base de données différente de celles utilisées pour l'apprentissage.

Ce test permet à la fois d'apprécier les performances du système neuronal et de détecter le type de données qui pose problème. Si les performances ne sont pas satisfaisantes, il faudra soit modifier l'architecture du réseau, soit modifier la base d'apprentissage.

1.6 Principe de fonctionnement

Le fonctionnement d'un réseau est plus simple, et dépend uniquement du fonctionnement de ses neurones. Celui-ci doit être appréhendé comme un circuit électrique contrôlé par une horloge. A chaque cycle, tous les neurones vont, de manière parallèle, calculer une valeur de sortie en fonction de la somme de leurs valeurs d'entrées, sachant que, pour un neurone A, les valeurs en entrée au cycle n sont les valeurs en sortie au cycle n-1 des neurones connectés à A. On va donc assister, à chaque top d'horloge du réseau, à une propagation de l'influx nerveux de ses entrées vers ses sorties (PANZOLI, 2003).

2 Apprentissage des réseaux de neurones :

L'apprentissage est vraisemblablement la propriété la plus intéressante des réseaux neuronaux. C'est une phase du développement d'un réseau de neurones durant laquelle le comportement du réseau est modifié jusqu'à l'obtention du comportement désiré. L'apprentissage neuronal fait appel à des exemples de comportement (TOUZET, 1992).

L'apprentissage permet aux réseaux de neurones de réaliser des taches complexes dans différents types d'application (classification, identification, reconnaissance de caractères, de la voix, vision, système de contrôle…). Ces réseaux de neurones peuvent souvent apporter une solution simple à des problèmes encore trop complexes ne pouvant être résolus rapidement par les ordinateurs actuels (puissance de calcul insuffisante) ou par notre manque de connaissances.

2.1 L'algorithme d'apprentissage :

C'est un ensemble de règles bien définies permettant de réaliser un tel processus d'adaptation des poids. Au niveau des algorithmes d'apprentissage, il a été défini deux grandes classes selon que l'apprentissage est dit supervisé ou non supervisé.

2.1.1 L'apprentissage supervisé :

Ce cas d'apprentissage pour les réseaux de neurones formels, consiste à calculer les coefficients de telle manière que les sorties du réseau de neurones soient, pour les exemples utilisés lors de l'apprentissage, aussi proches que possibles des sorties "désirées" (FRMLING, 1996).

1.1 L'apprentissage non supervisé :

Dans cette catégorie, la réponse désirée est inconnue. Le réseau est alors supposé découvrir lui-même la meilleure représentation de l'information fournie.

Remarquons cependant que les modèles à apprentissage non supervisé nécessite avant la phase d'utilisation une étape de labélisation effectuée l'opérateur, qui n'est pas autre chose qu'une part de supervision (FRMLING, 1996).

3 Algorithmes évolutionnistes :

Les Algorithmes évolutionnaires sont des méthodes d'optimisation stochastique basées sur une simulation brute de l'évolution naturelle des populations: croisements, mutations, sélections, Précisément, elles font partie du champ de l'Intelligence Artificielle (IA). Il s'agit d'IA de bas niveau, inspirée par « l'intelligence » de la nature. Le principe de base s'inspire de la théorie de Darwin sur l'évolution des espèces qui explique comment depuis l'apparition de la vie les

espèces ont su évoluer de façon innovante et souple dans le sens d'une meilleure adaptation à l'environnement, tout en permettant aux seuls individus bien adaptés à leur environnement de se reproduire.

Malgré la simplicité du processus évolutionnaire, fabriquer un algorithme évolutionnaire efficace est une tâche difficile, car les processus évolutionnaires sont très sensibles aux choix algorithmiques et paramétriques. L'expérience a prouvé que les réussites les plus importantes sont fondées sur une très bonne connaissance du problème à traiter, et une compréhension délicate des mécanismes évolutionnaires.

3.1 Principe de base des algorithmes évolutionnaires

Un algorithme évolutionnaire est une méthode itérative qui utilise des opérateurs de variations stochastiques sur un "pool" d'individu (la population). Chaque individu de la population représente une solution possible qu'est une version encodée du problème. Au départ, cette population initiale est engendrée aléatoirement. À chaque génération/itération de l'algorithme, les solutions sont sélectionnées, rassemblées en paires et recombinées afin de produire de nouvelles solutions qui remplaceront les "moins bonnes" car les individu de faible qualité sont supprimés de la population, tandis que les individus de haute qualité sont reproduits, et ainsi de suite. Le but est de diriger la recherche sur certaines portions de l'espace de recherche et d'augmenter la qualité moyenne à l'intérieur de la population. L'évolution stoppe quand le niveau de performance souhaité est atteint, ou qu'un nombre fixé de générations s'est écoulé sans améliorer l'individu le plus performant. Une fonction d'évaluation associe une valeur numérique d'adaptation (fitness) à chaque individu dans le but de déterminer sa pertinence ou sa qualité par rapport au problème (critère de sélection).

3.2 La boucle évolutionnaire :

La population initiale est créée de manière aléatoire. Cette population est le point de départ du processus d'évolution. Considérons un environnement quelconque dans lequel vit une population primitive et peu adaptée à cet environnement, cette population n'est pas uniforme c'est-à-dire certains individus sont mieux armés que d'autres. Ces individus mieux équipés ont par conséquent une probabilité de survie plus grande que les autres et auront de fait d'autant plus de chances de pouvoir se reproduire. Une boucle consistant en les étapes d'évaluation de sélection de mutation ou de croisement est exécutée un certain nombre de fois chaque opération est appelé une génération. Le cycle principal d'un algorithme évolutionnaire consiste à sélectionner des individus parents, à partir d'une population de départ, qui en se reproduisant

par des opérations de "croisement" et "mutation" engendrent de nouveaux descendants. Le croisement et la mutation ont pour but la génération de nouveau candidat a l'intérieure de l'espace de recherche par la variation des solutions existantes. L'opérateur de croisement prend un certain nombre d'enfant par recombinaison des parents pour imiter la nature stochastique de l'évolution une probabilité de croisement est associé à cette opérateur. La répétition de ce cycle de base produit donc une succession de générations de solutions, jusqu'à satisfaction d'un critère donné de fin de cycle.

3.3 Les méthodes d'algorithmes évolutionnaires

Plusieurs méthodes évolutionnaire ont été proposées, On distingue principalement quatre grandes familles d'algorithmes évolutionnaires comme indiqué par la figure.

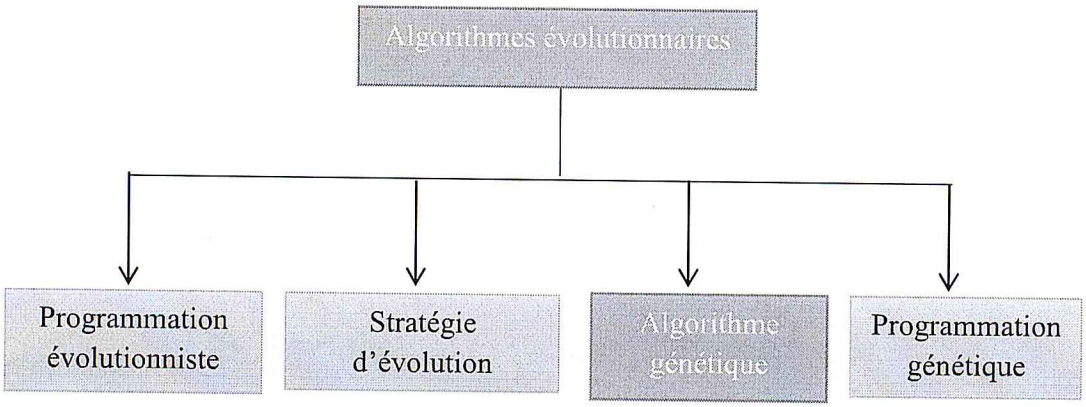


Figure 3: Différentes classes d'algorithmes évolutionnaire

• **Stratégies d'évolution** : le codage des solutions peut être réalisé par des structures de données plus complexes que dans les algorithmes génétiques. Par ailleurs, les opérateurs de mutation ont une place aussi importante que les opérateurs de croisement.
• **Programmation évolutionnaire** : elle est fondée essentiellement sur l'opérateur de mutation et n'utilise pas d'opérateur de croisement. Comme les Stratégies d'évolution, le codage des solutions peut faire intervenir des structures de données complexes. Cette approche a été développée initialement pour faire évoluer des automates à états finis.
• **Programmation génétique :** Apparue initialement comme une extension du modèle d'apprentissage des algorithmes génétiques, La ***(PG)*** permet de générer des fonctions informatiques à partir des principes évolutionnaires.

• **Algorithmes génétiques** : l'opérateur de croisement est considéré0 comme étant le plus important des opérateurs. La mutation est appliquée avec des probabilités très faibles et agit en tant qu'opérateur d'arrière-plan. Le codage des solutions consiste en une représentation généralement binaire des individus (HOUARI, 2012).

4 Les algorithmes génétiques :

4.1 Introduction :

Les algorithmes génétiques sont des techniques de programmation qui s'inspire du principe de l'évolution des espèces décrit par Darwin, Les algorithme génétique ont été initialement développés par John Holland (1975) dans deux buts principaux, Mettre en évidence et expliquer rigoureusement les processus d'adaptation des systèmes naturels et concevoir des systèmes artificiels qui possèdent les propriétés des systèmes naturels. Leurs champ d'application sont très vaste (traitement d'image, optimisation d'emploi du temps optimisation de design apprentissage des réseaux de neurone …. ect). La raison de ce grand nombre d'application est la simplicité de leur mécanisme et la facilité de leur mise en application. Il existe plusieurs types de ces algorithmes mais l'idée essentielle est la même : simuler l'évolution d'une population dans un espace de recherche à l'aide de trois opérateurs: sélection, croisement, mutation.

4.2 Les opérateurs génétiques :

Les algorithmes génétiques utilisent un vocabulaire similaire à celui de la génétique, ils sont fondés sur une représentation chromosomique des solutions du problème. Les algorithmes génétiques sont le type d'algorithme le plus connu et le plus utilisé des algorithmes évolutionnaires, développée dans les années 70 par Holland. Il ne base pas sur un ***individu***, mais sur une ***population*** d'individus qui vont évoluer de génération en génération pour obtenir un résultat se rapprochant de la solution optimale.

Les algorithmes génétiques attribuent à chaque individu un "***fitness***" qui mesure la qualité de la solution qu'il représente, souvent c'est la valeur de la fonction à optimiser. Ensuite, une nouvelle population des solutions possibles est produite en sélectionnant les parents parmi les meilleurs de la "***génération***" actuelle pour effectuer des ***croisements*** et des ***mutations.***

4.3 Principe de fonctionnement des algorithmes génétiques :

Quand on utilise les algorithmes génétique, aucune connaissance de la manière dont résoudre le problème n'est requise, il est seulement nécessaire de fournir une fonction permettant de

coder une solution sous forme de gènes ainsi que de fournir une fonction permettant d'évaluer la pertinence d'une solution au problème donné.

Un chromosome est une suite de gêne, on peut par exemple choisir de regrouper les paramètres similaires dans un même chromosome et chaque gène sera repérable par sa position. Chaque individu est représenté par un ensemble de chromosomes, et une population est un ensemble d'individus. La figure suivante représente le Cycle de l'évolution d'une population.

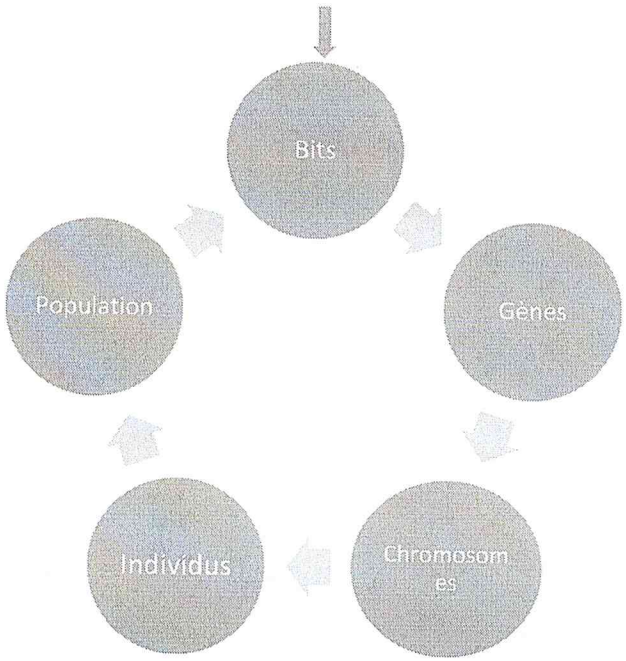


Figure 4: Cycle de l'évolution d'une population.

1.2 Codage de données :

Le codage (ou représentation) d'un individu doit englober les caractéristiques fondamentales du problème, permettre une transformation facile sur l'espace de recherche et générer, si possible, des solutions admissibles. Un bon codage doit ainsi : faciliter la définition et l'application d'opérateurs de variation (transformations génétiques : mutation, croisement,...) permettant de couvrir correctement l'espace des individus ; être cohérent par rapport au problème traité et simple dans sa construction.

1.3 Opérateur de croisement :

Le croisement utilisé par les algorithmes génétiques est la transposition informatique du mécanisme qui permet, dans la nature, la production de chromosomes qui héritent partiellement des caractéristiques des parents. Le croisement a pour but d'enrichir la diversité de la population

en manipulant la structure des chromosomes, et de permettre la *recombinaison* des informations présentes dans le patrimoine génétique de la population (LEILA, 2007).

1.4 Opérateur de sélection :

La sélection permet d'identifier statistiquement les meilleurs individus d'une population et d'éliminer les mauvais.

Cet opérateur est peut-être le plus important puisqu'il permet aux individus d'une population de survivre, de se reproduire ou de mourir. En règle générale, la probabilité de survie d'un individu sera directement reliée à son efficacité relative au sein de la population (LEILA, 2007).

1.5 Opérateur de mutation :

Une mutation consiste simplement en l'inversion d'un bit (ou de plusieurs bits, mais vu la probabilité de mutation c'est extrêmement rare) se trouvant en un locus bien particulier et lui aussi déterminé de manière aléatoire.

La mutation a pour rôle de maintenir une certaine diversité dans la population et protège les individus contre une perte des informations essentielle contenues dans leurs gènes. Elles permettent d'assurer une recherche aussi bien globale que locale et garantit la convergence vers l'optimum.

5 Neuroevolution :

5.1 Introduction :

La Neuroevolution (NE), est l'évolution artificielle de réseaux de neurones en utilisant des algorithmes génétiques, la neuro-évolution a montré de grandes promesses dans les tâches d'apprentissage de renfort. Les systèmes NE qui évoluent les topologies de réseau et les poids de connexion simultanément ont également été proposés.

Une question majeure dans NE est de savoir si telle évolution des topologies et des poids dans les réseaux de neurones artificiels peuvent améliorer la performance de NE.

D'une part, l'évolution de topologie avec les poids pourraient rendre la recherche plus difficile. D'autre part, ils peuvent améliorer considérablement les performances de NE. Dans Notre projet nous présentons une nouvelle méthode de NE appelé Neuro Evolution of Augmenting Topologies (NEAT) qu'est conçu pour prendre les mesure de la structure comme

un moyen de réduire au minimum la dimension de l'espace de recherche des poids et de connexion.

5.2 Caractéristiques :

Il existe beaucoup d'algorithmes de neuro-evolution. Il y'a certain qui évoluent les poids de connexion pour une topologie fixe (parfois appelé neuro-evolution classique), et d'autre qui évoluent à la fois la topologie du réseau et ses poids. Cet algorithme est appelés TWEANNs.

5.3 TWEANNs :

Les approches évolutives à l'optimisation de réseau neuronal ont été utilisées depuis longtemps. Cependant, dans la dernière décennie, on a connu un développement d'une grande variété de systèmes qui optimisent les deux paramètres (poids) et la topologie des RNA. Ces méthodes sont appelées TWEANNs qui signifie « *Topology and Weight Evolving Artificial Neural Networks* ». Ces systèmes tentent de résoudre le problème habituel des utilisateurs de RNA pour concevoir l'optimisation des topologies et des poids.

Nous allons introduire 3 systèmes de TWEANNs dans ce qui suit (DRCHAL, 2006):

5.3.1 GNARL :

GNARL est une méthode TWEANN basé sur la programmation évolutive c'est-à-dire qu'elle est basée que sur la mutation. Le GNARL fonctionne comme se suit :

- Eu début, une population d'individus choisis au hasard avec un nombre fixe de neurones d'entrée et de sortie est généré. . Ils contiennent également un nombre aléatoire de neurones cachés (typiquement de 0 à 5) et un nombre aléatoire de liaisons d'interconnexion (typiquement 0 à 10). Ces RNA peuvent également contenir des liens récurrents.
- Le fitness de l'ensemble de la population est évaluée.
- Les organismes les plus aptes sont sélectionnés pour la prochaine génération (typiquement une moitié de la population).
- Des nouveaux individus sont créés en utilisant un ensemble d'opérateurs de mutation semblable.
- Ensuite, le système continue avec la sélection à nouveau jusqu'à ce qu'a l'obtention d'une solution.

GNARL utilise deux types de mutation: paramétriques et structurelles.la mutation paramétrique change de manière aléatoire les valeurs de poids en introduisant un bruit gaussien.

Le niveau de bruit est réduit au cours de l'évolution. Les opérateurs de mutation structurels ajoutent ou suppriment des neurones ou des liens à partir de réseaux. La fréquence des mutations structurelles est également abaissée progressivement.

GNARL a été utilisé avec succès pour résoudre des problèmes de simulation de vie artificielle comme la simulation des fourmis qui sont contrôlés par un réseau de neurones (DRCHAL, 2006)

5.3.2 SANE (Symbiotic Adaptive Neuro-Evolution)

SANE qui signifie (Symbiotic, Adaptive Neuro-Evolution) est un système TWEANN très intéressant. Il est basé sur la coévolution des neurones simples avec des plans qui contiennent des informations concernant comment les neurones sont reliés dans un réseau final. Les neurones et les plans sont évolués dans des populations distinctes.

5.3.3 Codage cellulaire :

Un problème important que nous devons traiter lors de la conception de TWEANN est le codage génétique du réseau de neurone.

La plupart des systèmes TWEANNs utilisent le codage direct. Certain utilisent le codage indirecte nous citons parmi eux le codage cellulaire (CE).

CE est un système très puissant, il a réussi à résoudre de nombreux problème (Double Pole Balancing par exemple).Bien qu'il ait été surmonté par NEAT (NeuroEvolution of Augmenting Topologies) qui est un système de codage direct.

6 NeuroEvolution of Augmenting Topologies (NEAT):

Dans notre projet nous intéressons à NEAT donc il sera discuté en détail, Il a été développé par Keith Stanley et Risto Miikkulainen a l'Université du Texas à Austin. NEAT est un algorithme TWEANN il peut évoluer les réseaux de neurones récurrents. NEAT avéré être le plus approprié pour des problèmes dynamiques (contrôle des robots, jouer au jeu, etc.)

6.1 Définition :

NEAT est un algorithme génétique puissant soigné pour des Neuro-Evolution des topologies augmentées qui caractérise vraiment l'ensemble du processus d'optimisation comme des organismes qui sont continuellement évolué à partir de forme minimale de complexité.

NEAT est une technique pour faire évoluer les réseaux de neurones, qui utilise des algorithmes génétiques pour apprendre les structures du réseau ainsi que leur poids (Stanley, 2007).

6.2 Le principe de base de NEAT :

L'algorithme de NEAT commence avec un perceptron sans couche cachée. Dans le processus d'optimisation du réseau de neurones, autant le nombre de nœuds que les connexions et les poids de connexion subissent des mutations (ajouter/enlever un nœud/connexion, modifier les poids synaptiques) et des croisements (fusion des sous-structures du réseau, combinaison des poids de connexion). Les connexions peuvent être récurrentes, i.e. il peut exister des boucles dans la structure du réseau. NEAT considère, entre autres, un codage des génomes à taille variable, et propose des opérateurs de mutation et croisement adaptés à ce codage. Il exploite aussi des mécanismes de spéciation, qui considère des groupes d'individus ou « espèces ». Pour définir l'appartenance des individus aux groupes, une fonction de compatibilité est définie. L'algorithme optimise d'abord la topologie du réseau de neurones pour les individus de chaque espèce de manière indépendante au reste de la population. Ensuite, les différentes espèces sont mises en concurrence et les poids synaptiques sont réglés. La figure ci-dessous représente l'encodage génétique de NEAT.

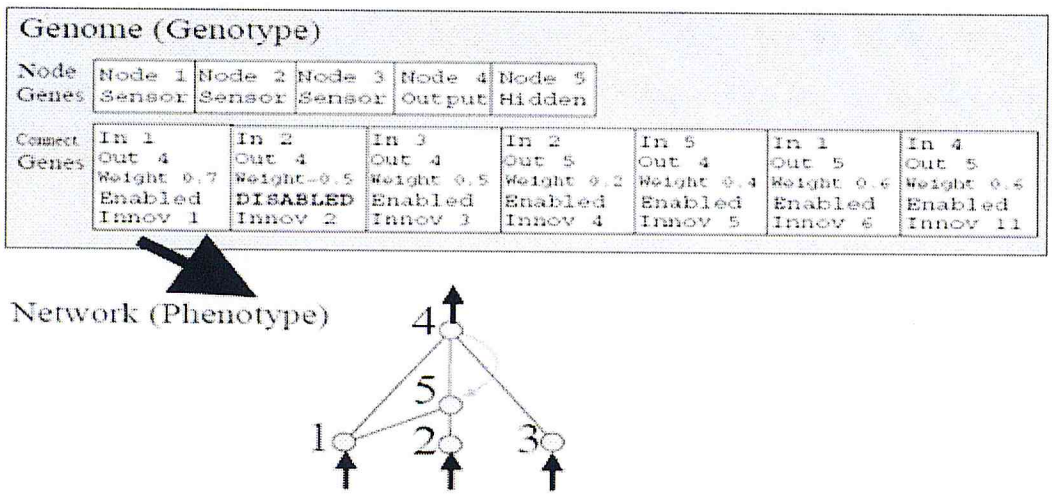


Figure 5:Encodage génétique de NEAT

6.3 Les génotype et Phénotypes :

Les génomes sont composés d'une liste des liens et des connexions entre les neurones qui les composent Chacun de ces neurones est muni :

- d'un identifiant unique qui nous permettra de réaliser le marquage historique.

- du type du neurone (input, bias, output ou hidden),
- d'un indicateur booléen pour savoir si le neurone fait l'objet d'une connexion récurrente (utile lors de l'étape de mutation),

Chaque lien contient :

- un identifiant (numéro d'innovation) servant le marquage historique.
- des identifiants des deux neurones liés,
- d'une pondération,
- et de deux indicateurs booléens pour savoir si le lien est bien actif et si il est récurrent,

6.4 Le numéro d'innovations :

Numéros d'innovation (ou marques historiques) représentent le noyau du système NEAT. Ils permettent d'aligner les génomes par les gènes correspondants. Numéros d'innovation sont utilisés pour retracer l'origine historique d'un gène. Ce suivi a besoin d'un compteur global appelé un nombre global d'innovation.

A Chaque fois qu'une nouvelle innovation apparaît dans la population le compteur est incrémenté et ce nombre est affecté au nouveau gène.

NEAT détient une liste d'histoire Relier à l'innovation qui stocke tous les numéros d'innovation déjà attribués. Ce lien d'innovation "mémoire" est détient dans le cadre d'une seule génération, puis, il est effacé. Cependant, il est possible de conserver pour toute l'évolution. Un tel réglage réduit la probabilité qu'un gène de liaison peut représenter plus de fonctionnalités.

6.5 Gène disjoint/d'excès :

Les gènes sont soit disjoint ou d'excès. Selon si elles se produisent à l'intérieur ou à l'extérieur de la gamme de numéros d'innovation de l'autre parent. Ils représentent la structure qui n'est pas présent dans l'autre génome.

En composant les descendants, les gènes en excès ou disjoints sont toujours inclus à partir du parent qui les possèdent.

Le nombre de gènes en excès et disjoints entre une paire de génomes est une mesure naturel de leur compatibilité. Plus ils sont disjoint moins ils partagent l'histoire d'évolution, et donc ils sont moins compatible

Par conséquent, nous pouvons mesurer la distance de compatibilité de différentes structures dans NEAT comme une combinaison linéaire simple du nombre de gènes excès (E) et disjoints (D). Ainsi que la moyenne des différents poids des gènes croisés (w) selon cette formule :d = $\frac{C1E}{N} + \frac{C2D}{N} + C3.W$ tel que C1, C2, C3 sont des coefficients

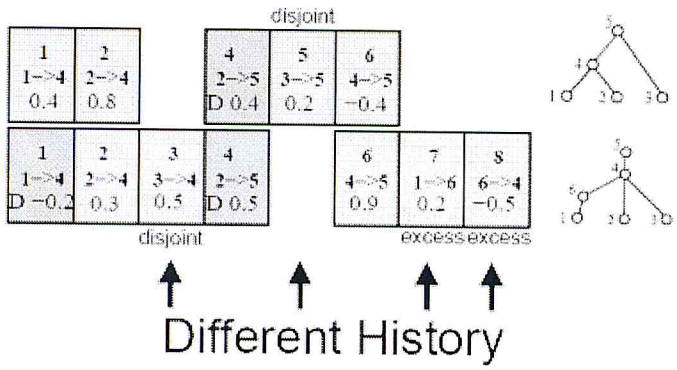


Figure 6: Les gènes disjoints et d'excès

6.6 La spéciation :

Les petites structures peuvent être optimisé plus rapidement que les grandes structures, et l'ajout de nœuds et de connexions diminue habituellement l'aptitude du réseau, les structures récemment augmentés ont peu d'espoir de survient plus d'une génération, même si les innovations qu'il représente pourrait être cruciale pour résoudre la tâche dans le long terme. La solution est de protéger l'innovation par la spéciation de la population.

La spéciation permet aux organismes de faire une concurrence principalement au sein de leurs propres niches au lieu la population en général. De cette façon, les innovations topologiques sont protégées dans une nouvelle niche où ils ont le temps d'optimiser leur structure par une concurrence dans cette niche. L'idée est de diviser la population en espèces telles que topologies similaires sont dans la même espèce (Risto Miikkulainen, 2002).

6.7 Le fitness sharing :

Le *fitness partagé* est la technique la plus souvent utilisé dans l'approche NEAT, ce fitness est calculé à l'intérieur de chaque espèce et le croisement des individus se fait également entre les membres de la même espèce. Afin qu'une espèce soit performante, mais pas forcément

optimale, le *fitness* d'un individu peut être plus faible que le fitness des autres membres de la même espèce. Pour assurer l'évolution de la population, seuls les meilleurs individus de chaque espèce sont conservés en fonction d'un seuil de survie et obtiennent le droit de se reproduire (Stanley, 2007).

6.8 Les opérations génétiques de NEAT :

Il y a cinq opérateurs génétiques dans NEAT. Il s'agit de deux types d'opérateurs de mutation: mutation paramétrique et les mutations structurelles. La mutation paramétrique modifie les paramètres de réseau. Les coefficients de pondération. Mutation structurelle contrairement produit des innovations de topologie. Le croisement est représenté par l'opérateur de croisement (DRCHAL, 2006).

1.6 Mutation par ajout d'un lien:

C'est le premier opérateur de mutation structurelle. Il ajoute un nouveau gène de lien vers le génome afin de relier deux neurones non liées au préalable (RNA sont des graphiques orientés). La situation est illustrée à la figure. Seuls les gènes de liaison sont représentés. Le nombre écrit dans chaque gène de lien est le numéro d'innovation. Cependant, les neurones non connectés 3 et 5 sont reliés par un nouveau lien. Cette nouvelle liaison se voit attribuer un nouveau numéro d'innovation. Numéros de l'innovation seront discutés par la suite (DRCHAL, 2006).

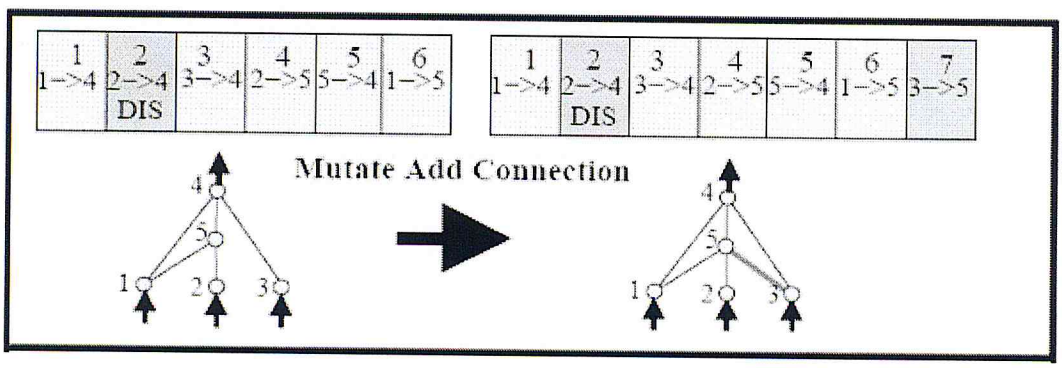


Figure 7:Mutation par ajout d'une connexion

1.7 Mutation par ajout d'un neurone:

Cet opérateur ajoute un nouveau neurone au réseau. La figure montre le processus de mutation. Il peut être décrit par les étapes suivantes:

1. Sélectionner un gène de lien au hasard.
2. Désactiver-la en ajustant son indicateur de validation.
3. Insérer un nouveau neurone.

4. Brancher ce nouveau neurone au réseau.
5. Faire en utilisant deux nouveaux liens, chacun attribuer un nouveau numéro d'innovation. S'assurer que le nouveau lien de structure liaison-neurone est placé à droite sur le lien ancien (désactivé).
6. Relier également le nouveau neurone au neurone de polarisation (qui n'est pas représenté).
7. Définir le poids de la nouvelle liaison menant au nouveau neurone à 1.0.

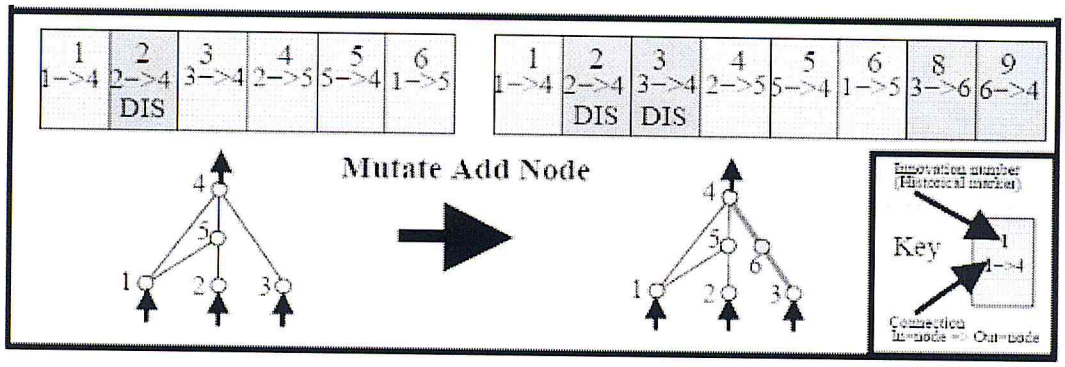


Figure 8: Mutation par ajout d'un neurone.

1.8 Les mutations de liaison à bascule :

Lorsque la mutation de liaison à bascule est appliquée à un génome, un gène de lien au hasard est choisi. Puis, son indicateur de validation est annulé. Le but de cet opérateur est de permettre à nouveau aux gènes handicapés liés par « Ajouter neurone mutation ». Ensuite, c'est le seul opérateur qui peut réduire la topologie.

1.9 La mutation de poids:

La mutation de poids est en fait le seul représentant d'une mutation paramétrique dans NEAT originale.

Les poids peuvent être décalés ou encore remplacés par une nouvelle valeur (mutation froid). Tous les gènes d'un génome peuvent subir une mutation, mais la probabilité de mutation des gènes ajoutés plus tard (celles à la fin d'un génome) est plus faible.

6.8.1 Le croisement :

L'opérateur de croisement est un opérateur de croisement diffèrent du croisement classique des algorithmes génétiques, cette différence est dans les faite de qu'il ne produit qu'une seule

progéniture (2 parents donnent 1 enfant), contrairement au croisement classique qui en produit deux. Par conséquent, cette opérateur de croisement choisit une information génétique en alternance des deux parents, et supprime les morceaux inutilisés (ce qui n'est pas le cas du croisement classique qui distribue toutes les informations parentales sur les enfants). Dans la figure qui suit on peut voir un exemple de l'accouplement dans NEAT. Cet accouplement inclut le mécanisme d'alignement du gène précité. La première étape consiste a aligné les deux parents par leur numéros d'innovations des liens correspondant. Ceci est effectué sur les gènes des liens communs entre les deux parents. On peut aussi avoir des gènes d'excès qui vont être ceux du gène ayant le plus grand numéro d'innovation maximum, comme on peut aussi avoir des gènes dis disjoints qui vont correspondre aux autres gènes non commun entre les deux parents.

Les marques historiques permettent à NEAT d'effectuer le croisement utilisant les génomes linéaires sans avoir besoin d'une analyse topologique couteuse.

NEAT effectue l'opérateur de croisement comme si on n'avait aucune analyse topologique exhaustive. La viabilité de la progéniture est garantie ou mieux préservée, parce que l'opérateur de croisement adopte la topologie du parent (DRCHAL, 2006) la figure suivante représente cet opérateur de croisement.

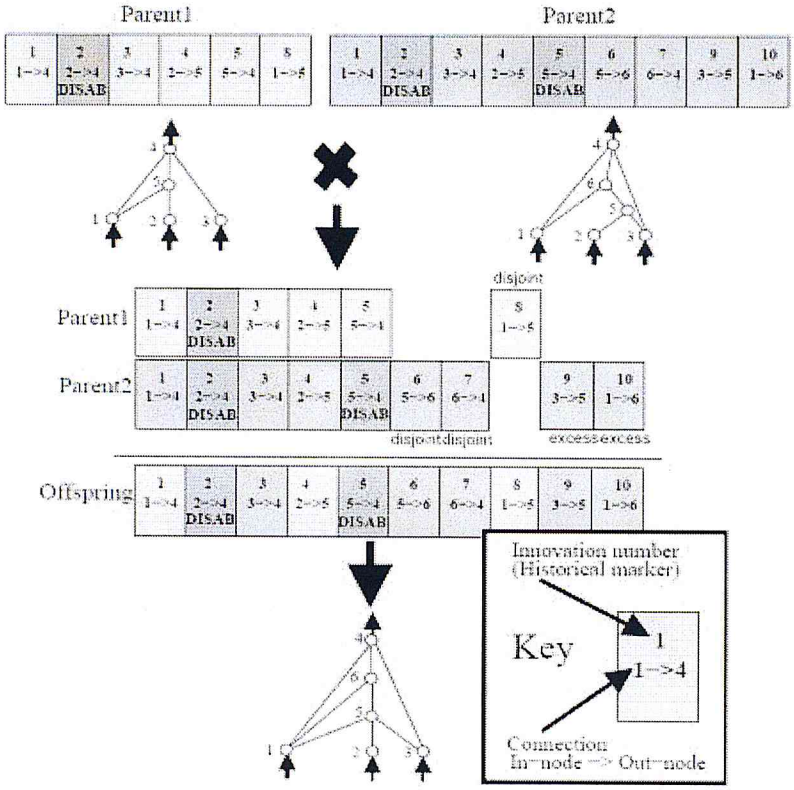


Figure 9: l'opération du croisement.

7 Conclusion

Dans ce chapitre, nous avons pu découvrir et exploré la neuro-évolution en particulier le model NEAT (neuro-evolution of augmenting topologies) proposé par O.Stanley. La principale conclusion est que NEAT est un algorithme génétique puissant pour faire évoluer artificiellement les réseaux de neurones. NEAT démontre que l'évolution topologique avec des poids peut être un avantage majeur. Et que cette évolution est bien plus efficace que les méthodes de neuro-évolution jusqu'à présent. Les études montrent que les marques historiques, la protection de l'innovation par la spéciation, et la croissance incrémentale de structure minimale travaillent tous ensemble pour produire un système qui est capable d'évoluer des solutions de complexité minimale.Moyennant ce model (NEAT), nous pourrions avoir des résultats en temps réel des temps d'exécution des différents scénarios, et ainsi permettre au client de choisir le meilleur scénario.

CHAPITRE 3

Adaptation de l'approche NEAT pour la simulation

1.1 Introduction :

Notre but est de réaliser un simulateur en utilisant une approche neuro-évolutionnaire en tant que modèle de substitution pour l'aide à la décision dans le cadre du pilotage des systèmes de production automatisés.

La première étape de notre travail a donc consisté à générer les données nécessaires à l'entrainement de notre réseau de neurones: c'est l'objet du prochain paragraphe dans lequel nous décrivons la manière dont ces données ont été obtenues et leurs caractéristiques, ainsi que les structures utilisées dans notre programme pour stocker ces données.

Par la suite nous présentons les détails de l'expérimentation avec NEAT.

1.2 L'utilisation de réseau de neurone (NEAT) dans la simulation :

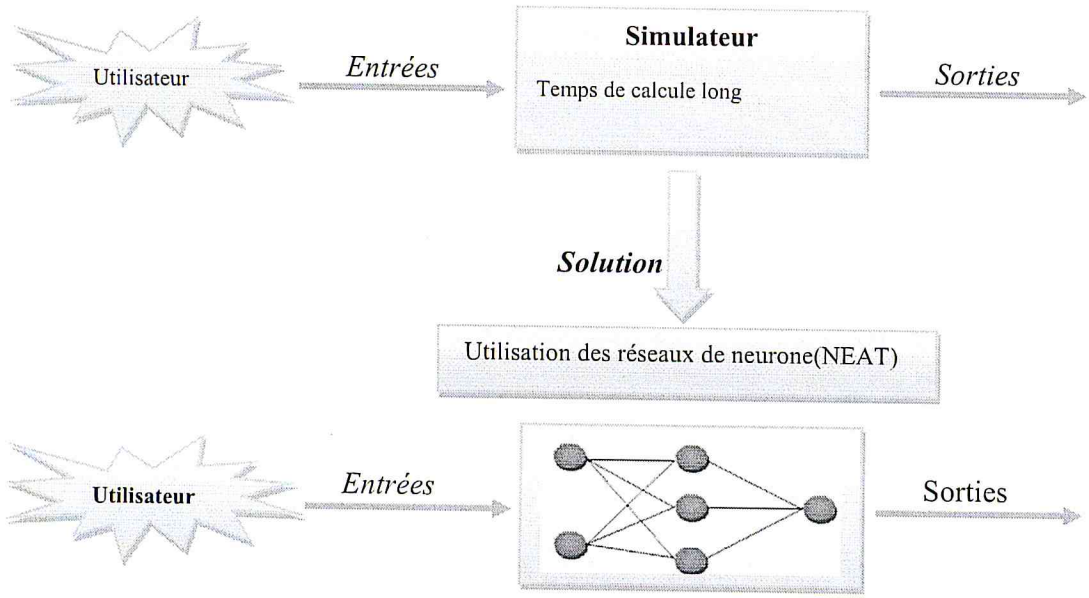


Figure 1: l'utilisation d'un réseau de neurone dans la simulation

La figure ci-dessus représente le problème envisagé lors de la simulation dans les systèmes de production.

1.3 NEAT (Neuro-Evolution of Augmenting Topologies) :

NEAT (*Neuro-Evolution of Augmenting Topologies*) est présenté comme une méthode d'évolution de structures neuronales qui optimise simultanément l'**architecture** et les **poids de connexion** d'un réseau de neurones. L'algorithme commence avec un perceptron sans couche

cachée. Dans le processus d'optimisation du réseau de neurones, autant le nombre de nœuds que les connexions et les poids de connexion subissent des mutations (ajouter/enlever un nœud/connexion, modifier les poids synaptiques) et des croisements (fusion des sous-structures du réseau, combinaison des poids de connexion). Les connexions peuvent être récurrentes, *i.e.* il peut exister des boucles dans la structure du réseau. Par conséquent, les réseaux de neurones résultants intègrent l'effet mémoire. NEAT considère, entre autres, un codage des génomes à taille variable, et propose des opérateurs de mutation et croisement adaptés à ce codage. Il exploite aussi des mécanismes de *spéciation*, qui considère des groupes d'individus ou « espèces ». Pour définir l'appartenance des individus aux groupes, une fonction de compatibilité est définie. L'algorithme optimise d'abord la *topologie* du réseau de neurones pour les individus de chaque espèce de manière indépendante au reste de la population. Ensuite, les différentes espèces sont mises en concurrence et les *poids synaptiques* sont réglés.

1.4 L'algorithme de NEAT :

Algorithme de NEAT

```
Début
    Initialiser (n=nombre de génération, m=nombre d'organisme) ;
    Générer la population (taille_population, nombre_entré, nombre_sortie) ;
      Pour (nombre de génération= 1 jusqu'à n)
            //traitement des organismes
        Pour (nombre de organisme= 1 jusqu'à m)
          Faire
            Lire (entrées) ;
            Lire (sorties) ;
           Activer (network()) ;
           Evaluation de fitness ();
           Calculer (sorties du réseau) ;
           Calculer (erreur) ;
           Spéciation () ;
      Fin d'organisme ;
```

Sélection () ;

Opérateur génétique (croisement (), mutation) ;

Fin de génération ;

Fin

1.5 Présentation des données de la base d'apprentissage :

Nos données représentent les différents scénarios qui peuvent être réalisé par la cellule. Ces scénarios seront composés des différentes règles d'ordonnancement choisis pour chaque type de produit, le nombre de produits pour chaque type, les règles d'ordonnancement pour les machines et le nombre de chariots qui peut ce trouvé au même temps dans la cellule (chariot : transporte un lot de produits dans la cellule).

Nous avons au totale 7 types de produits « A, B, E, I, L, P, T, » différents comme présenté dans la figure suivante :

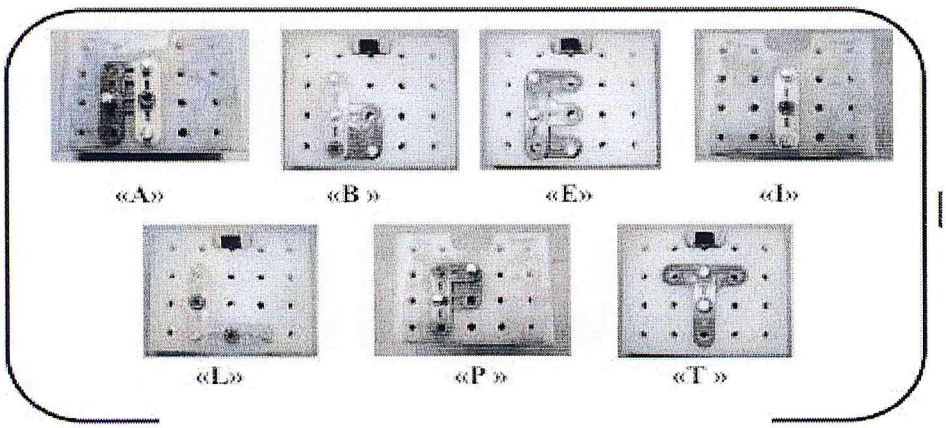


Figure 2: Les différents jobs qui peuvent être réalisé

Pour chaque produit nous avons le choix entre 5 règles d'ordonnancement, chaque règle portera un numéro (un entier) [1,5] :

Numéro	Nom	Description
1	RP1	la machine moins occupée avec ma plus courte distance.
2	RPSPT	La machine qui réalise le plus petit temps d'exécution.
3	RPLPT	La machine qui réalise le plus grand temps d'exécution.

4	RP2	la machine moins occupée.
5	RP_Trajet	La machine qui a la plus courte distance.

Tableau 2:Les régles d'ordonnencement des produits

Comme nous devrions donner pour chaque type de produit le nombre de produits qui sera entré dans la cellule (un entier) [0,10] donc on aura au maximum 10 produits pour chaque type.

En ce qui concerne les machine nous avons au totale 4 machine, pour chacune d'elle nous donnerons aussi une règle d'ordonnancement, donc un totale de 4 règles pour les machine qui pour chacune on donnera un numéro (un entier) [1,4]. Nous présentons ci-dessous ces règles d'ordonnancement des machines.

Numéro	Nom	Description
1	SPT	Le produit qui a le plus petit temps d'exécution.
2	LPT	Le produit qui le plus grand temps d'exécution.
3	FIFO	Le premier arrivé le premier servi
4	LIFO	Le dernier arrivé le premier servie.

Tableau 3:Les règles d'ordonnancement des machines.

Pour le nombre de chariots qui peut se trouver au même temps dans la cellule on a le choix entre 3 valeur : 4 produits, 8 produits ou bien 10 produits.

Chaque ligne de scénario aura une sortie spécifique, qui représentera le temps pris par la cellule pour exécuté le scenario en ms (temps d'exécution).

Après une recherche faite sur les réseaux de neurones, plus particulièrement, le traitement des données par les réseaux de neurone et leur présentation nous avons trouvé qu'il sera préférable que les données soient présentées dans un intervalle de [0,1]. Pour ce fait nous avons choisis la représentation suivante :

Chaque ligne de scénario est une ligne de 20 valeurs toutes comprises dans l'intervalle [0,1].

Donc chaque scénario représente les entrées de notre réseau de neurone.

Regle_du_produit_A , nombre_de_produit_A , Regle_du_produit_B , nombre_de_produit_B , Regle_du_produit_E, nombre_de_produit_E , Regle_du_produit_I , nombre_de_produit_I , Regle_du_produit_L , nombre_de_produit_L , Regle_du_produit_P , nombre_de_produit_P , Regle_du_produit_T , nombre_de_produit_T , regle_machine1 , regle_machine2 , regle_machine3 , regle_machine1 , nombre_de_chariots.

Exemple d'une ligne de scenario :

Entré de notre réseau de neurones (valeur des neurones de la couche d'entrée):

0.1, 0.1, 0.3, 0.2, 0.3, 0.1, 0.3, 0.4, 0.7, 0.4, 0.5, 0.3, 0.5, 0.3, 0.75, 0.25, 0.75, 0.25, 1, 0

Sortie (valeur du neurone de sortie désiré du réseau de neurones): 0.12750

Pour avoir un tel résultat nous somme passer par une normalisation qui a été faite comme suit :

Regle_du_produit= numéro de la règle [0,5]/ nombre totale de règles (5) ;

nombre_de_produit= nombre de produits [0 ,10]/nombre totale de produits (10) ;

regle_machine1= numéro de la règle [0,4]/ nombre totale de règles (4) ;

Pour le nombre de chariots qui peuvent se trouver au même temps dans la cellule on a décidé de le codifier en 2 valeur comme suit :

4 produits ⟶ 0, 0 ;

8 produits ⟶ 1, 0 ;

10 produits ⟶ 1, 1 ;

Et pour la normalisation de la sortie on va juste diviser le temps d'exécution pas 1000.

1.6 Stockage des données dans notre programme :

Pour pouvoir stocker les données utilisées dans notre programme, nous avons utilisé un fichier texte contenant à la fois les valeurs d'entrées et de sorties normalisées. Ce fichier étant préalablement créé avec l'aide du logiciel FLEXSIM. Le programme lit les différentes valeurs ligne par ligne pour construire la base d'apprentissage de notre réseau (chaque ligne représentant un scénario). Notre fichier comporte 20000 scénarios (lignes) dont certaines ont été dédié à l'entrainement ou l'apprentissage (19900 lignes donc 99.5% des données) et le reste a été consacré pour faire la validation de notre réseau.

1.7 La simulation avec NEAT :

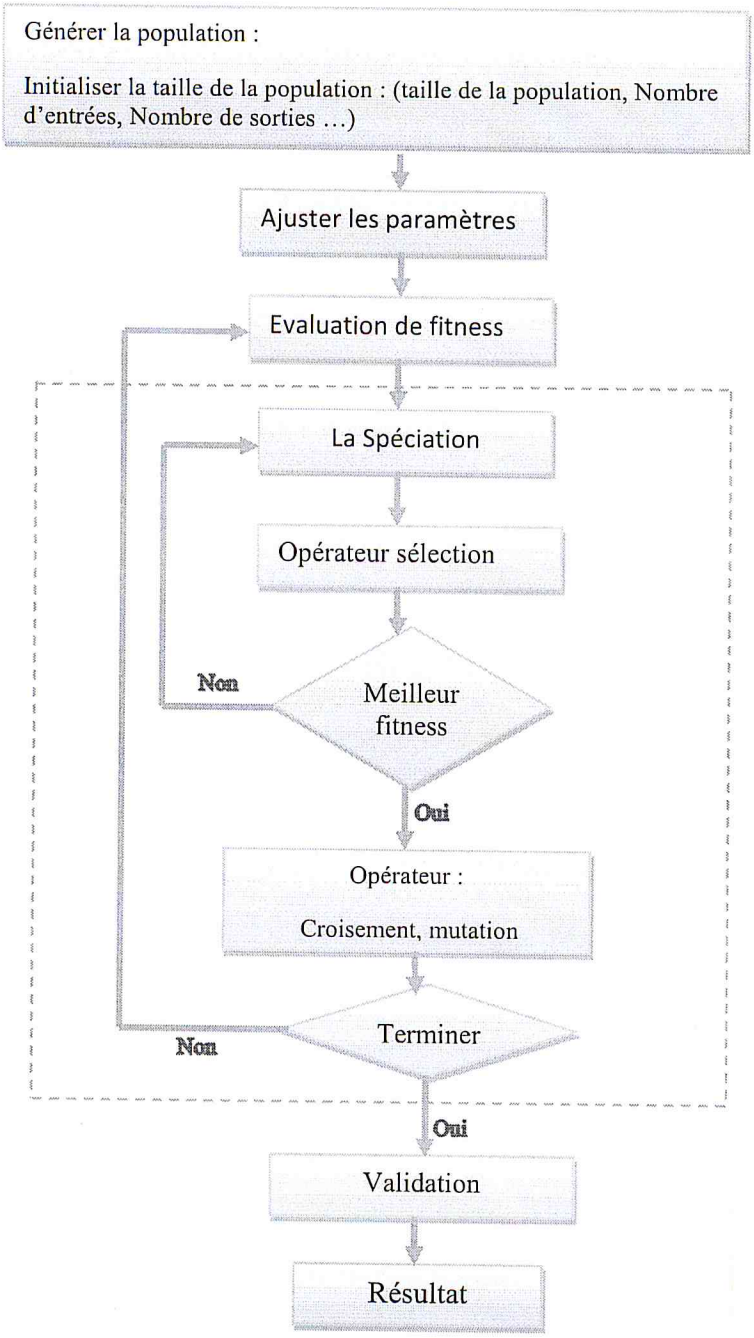


Figure 3: Principe de fonctionnement de NEAT.

Ci-dessus est présenté l'organigramme du fonctionnement de NEAT. Il s'agit de simuler l'évolution d'une population à laquelle on applique différents opérateurs (Croisement, mutations…) et que l'on soumet à une sélection, à chaque génération. Si les paramètres sont bien initialisés alors la population tend à s'améliorer.

Un tel algorithme ne nécessite aucune connaissance du problème : on peut représenter celui-ci par une boîte noire à laquelle on lui donne des entrées (les variables) et des sorties (les fonctions objectif). L'algorithme ne fait que lire les entrées et les sorties (sortie désirée), construire son réseau de neurone pour calculer ses sorties à partir des entrées lu et les comparer avec les sorties désirée.

1.7.1 Population

Cette étape consiste à initialiser notre population. En neuro-évolution et en NEAT une population est définit comme étant un ensemble de génome et organisme et leur espèces, sachant qu'un organisme est l'ensemble des génotypes et des phénotypes.

Nous avons choisi une population de 100 organismes (le nombre d'organisme d'une population définit la taille de celle-ci), chaque organisme de la population à 20 entré et une sortie, dont une taille maximum pour chaque organisme égale à 30 nous avons préféré d'avoir des organismes sans nœuds récurent et un pourcentage de liaison entre deux nœuds égale à 0.5, ces critères cité précédemment ont pour but de donner une définition complète d'une population. Pour le programmer en JAVA avec l'aide de la bibliothèque JNEAT nous avons dû faire appel à la classe « population » qui nous a permis de définir tout cela en une seule ligne de code qui est la suivante :

```
Population neatPop = new Population (100/*taille de la population */, 20 /*les entrées*/, 1/* les sorties*/, 30 /* taille max*/, false /* récurrence */, 0.5 /*probabilité de connecté 2 noeuds*/);
```

1.7.2 Ajustement des paramètres :

L'implémentation de NEAT dans notre programme fait usage d'une grande quantité de paramètres pour contrôler le système. Les paramètres permettant d'ajuster les réglages en fonction des expériences. Certains des paramètres du système de NEAT sont sensibles à la taille de la population, par conséquent nous les avons modifiés.

Voici un récapitulatif de ces paramètres avec les valeurs que nous avons utilisées dans nos premières expérimentations. Ces valeurs sont celles indiquées par Stanley et permettent d'après lui de résoudre le problème du XOR dans 100% des simulations.

Comme le taux de mutation et de *croisement* est particulièrement délicat à ajuster et que les valeurs idéales pour ces paramètres peuvent varier d'un problème à l'autre, nous avons lancé l'algorithme plusieurs fois en faisant varier ses deux paramètres. Le taux de *mutation* idéal pour les poids a été estimé empiriquement entre 0.8 et 0.9.

Nous avons donc lancé l'algorithme successivement avec des taux de *croisement* de 0.1, 0.3, 0.5 et 0.7. Le taux de *croisement* idéal a été estimé empiriquement entre0.3 et 0.4

Paramètre	***Valeur***	***Signification***
trait_param_mut_prob	0.8	Prob. de la mutation d'un seul param de trait
trait_mutation_power	0.9	poids de mutation sur un seul param de trait
linktrait_mut_sig	0.1	mutation_num change pour un changement de trait dans une connexion.
nodetrait_mut_sig	0.1	mutation_num change sur uneconnexion reliant un nœud qui a changé son trait
recur_prob	0.0	Prob. qu'une mutation d'une connexion qui ne doit pas être récurrent sera récurrente
weight_mut_power	0.05	Probabilité de mutation du poids d'une connexion
disjoint_coeff	1.0	Poids des gènes disjoints dans le calcul de la compatibilité
excess_coeff	1.0	Poids des gènes en excès dans le calcul de la compatibilité
mutdiff_coeff	0.4	Poids de l'écart moyen des pondérations dans le calcul de la compatibilité
compat_threshold	0.6	Limite de compatibilité entre deux génomes (utilisé pour la spéciation)

age_significance	1.0	L'âge significatif dans le cycle de l'époque
survival_thresh	0.2	Taux de survie appliqué aux individus d'une espèce pour déterminer la part des individus au *fitness* le plus élevé qui prend part à la croisement
mutate_only_pro	0.25	Probabilité d'un croisement sans croisement
mutate_random_trait_prob	0.1	Probabilité d'effectuer une mutation qui touche n'importe quelle connexion.
mutate_link_trait_prob	0.1	Probabilité de mutation d'une connexion trait.
mutate_node_trait_prob	0.1	Probabilité de mutation d'un nœud trait.
mutate_link_weights_prob	0.9	Prob.de mutation des poids des connexions
mutate_toggle_enable_prob	0.0	Prob.de mutation de type ena->dis | dis-ena des gènes
mutate_gene_reenable_prob	0.0	Prob. de changer le statut du gène à « ena »
mutate_add_node_prob	0.03	Prob. d'ajouter un nœud à la structure de génome.
mutate_add_link_prob	0.08	Prob. d'ajouter une connexion à la structure de génome
interspecies_mate_rate	0.001	Prob. de croisement hors les niches

mate_multipoint_prob	0.3	Prob. de croisement dans plus d'un point de deux génomes
mate_multipoint_avg_prob	0.3	Prob. de croisement dans plus d'un point de deux génomes avec le média.
mate_singlepoint_prob	0.3	Prob. de croisement dans un point unique de deux génomes
mate_only_prob	0.7	Prob. de croisement sans mutation.
recur_only_prob	0.0	Prob. de forcer sélection desconnexionsseulement qui sont naturellement récurrente
pop_size	100	La taille de population.
dropoff_age	15	Age où les espèces commencent à être pénalisées
newlink_tries	50	Le nombre d'essais mutate_add_linkqui tentera de trouver une connexion ouvert
print_every	1	Indiquer d'imprimer la population dans un fichier toutes les n générations
babies_stolen	0	Le nombre de babies_stolen qui siphonne les champions.
num_runs	1	Nombre d'exécution d'une expérimentation
num_trait_params	8	Numéro de trait.

Tableau 4:Ajustement des paramètres de NEAT

1.7.3 Evaluation du fitness :

En utilisant les sorties calculer par notre réseau on peut faire l'évaluation des organismes en calculant l'erreur commise par chaque organisme et son fitness.

Sachant que NEAT favorise les organismes ayant le meilleur fitness (le plus élevé), et notre but est d'obtenir à la fin des organismes avec une marge d'erreur très réduite. Donc plus l'erreur est petite plus l'organisme est meilleur et plus apte pour participé dans le croisement, pour cela il lui faudra un fitness élevé.

C'est pourquoi nous avons décidé d'opter pour que le fitness soit inversement proportionnelle avec l'erreur. Dont voici la formule :

$$Fitness = \frac{1}{\text{Erreur}}$$

L'erreur est commise par le réseau pour chaque organisme de la population. On calcule l'erreur pour chacun de scénario en faisant la différence entre la sortie désiré et celle du réseau de neurone, on somme l'erreur des défirent scénarios et on termine par devisé sur le nombre total des scénarios pour avoir une moyenne d'erreur pour tout l'organisme.

$$E = \frac{\sum |Sd - Sr|}{\text{N}}$$

E : Erreur

Sd : Sortie désiré

Sr : Sortie du réseau

N : nombre total des scénarios

Une fois le fitness calculé, les organismes de notre population vont être varié, pour cela on les distribue dans les espèces auxquelles elles appartiennent. Pour une meilleure évaluation de la population on calcule le max_fitness et le fitness_moyen de chaque espèce, en faisant appel à la classe « species » de la bibliothèque JNEAT contenant les fonctions adéquate pour cela.

1.7.4 Sélection & croisement :

Les différents opérateurs génétiques (sélection, croisement et mutation) se font en fonction des taux établi pendant le paramétrage de l'algorithme.

Pour la sélection et le croisement on choisit et on sélectionne les meilleurs organismes ayant le meilleur fitness pour l'accouplement (croisement) pour cela on a fait appel à la classe

« population » de NEAT qui contient une fonction qui s'occupe de la sélection et du croisement, cette fonction est appelé « epoch » à laquelle on donne une population et elle s'occupera de l'améliorer de génération en génération.

Cette fonction « epoch » fait elle-même appel à la classe « spacies » plus particulièrement sa fonction « ajust_fitness » qui marque l'organisme champion de l'espèce (celui qui a le plus grand fitness original) et marque les organismes qui peuvent être éliminé de cette espèce (ceux qui ne sont pas assez bon pour devenir parents). Ensuite « epoch » s'occupe elle-même du croisement

1.7.5 La validation :

La validation est faite pour voir si notre population finale est bonne à être utilisé ou pas. Pour cela, nous avons opté pour deux méthodes de validation.

La première consiste à prendre le meilleur organisme de toute la population, celui qui a le fitness le plus élevé et donc qui fait le moins d'erreurs. Lui donné nos scenarios gardé spécialement pour la validation et vois qu'est-ce qu'il nous donne comme résultat. Si les résultats son bon, alors le génome de cette organisme est sélectionné pour être utilisé dans notre application.

La deuxième méthode consiste à prendre un ensemble des meilleurs organismes de la population, ceux qui ont les meilleur fitness. Leur donné nos scénarios pour la validation et faire une moyennes des résultats donnés. Si les résultats son bon, alors les génomes de ces organismes sont sélectionnés pour être utilisé dans notre application.

1.7.6 Condition d'arrêt :

Pour savoir si notre apprentissage est terminé ou pas encore nous avons deux condition d'arrêt. La première c'est nous qui l'avions donné et qui est le nombre de génération qui est égale à GG, une fois ce travail est fait pour toutes ces générations on arrête notre apprentissage. La deuxième condition est donné par NEAT dans la fonction « epoch » qui arrête tout en cas d'une stagnation (les populations ne s'améliores plus de génération en génération).

En cas de stagnation de la population, Stanley préconise de ne faire reproduire que les deux meilleures espèces (celle dont le taux de descendance prévu était le plus important) et de leur faire générer chacune 50% de la population. Il conviendra de vérifier que cette technique d'élitisme ne risque pas de faire disparaître une espèce moins développée qui pourrait contenir le meilleur individu de la population.

2 Conclusion

A l'issue de ce chapitre nous avons pu voir en détail les méthodes et procédures qu'on a empruntées pour concevoir notre projet. En commençant par la présentation des données de la base d'apprentissage, qui ont étaient utilisées dans notre expérimentation NEAT. Cette dernière décrite étape par étape, de notre population initial jusqu'à l'arrivé à la validation, en passant par l'évaluation (paramètres et fitness) et le croisement.

Dans le chapitre suivant, nous allons implémenter et mettre en œuvre ce que nous avons proposé dans l'étude de réalisation de notre système.

Chapitre 4

Expérimentation tests et résultat

Chapitre 4

1.1 Introduction :

Après avoir présenté, dans le chapitre précédent, les différentes procédures pour la réalisation de notre projet en utilisant le réseau de neurone NEAT (*NeuroEvolution of Augmenting Topologies*).

Nous allons voir dans ce chapitre la définition des outils de développement utilisés pour l'implémentation de notre système et aussi la présentation de l'application en faisant quelque test. Enfin, pour conclure nous aimerions présenter nos perspectives.

1.2 Environnement de développement :

1.2.1 Le langage de programmation choisi (JAVA) :

Pour la réalisation de notre projet nous avons utilisé le langage de programmation Java (SUN), sous l'éditeur NetBeans. (www.netbeans.org).

- Nous avons choisi le langage JAVA pour les raisons suivantes :
- Nous sommes bien familiarisées avec les notions du langage JAVA;
- L'application peut s'exécuter sur n'importe quel système d'exploitation à condit iond'avoir la machine virtuelle java installée sur la machine (portabilité);
- La disponibilité de la documentation et de l'assistance (forums) ;

JAVA est un langage de programmation moderne développé par Sun Microsystems, c'estun l angage de programmation orienté objet basé sur le langage C++ mais avec desfonction nalités qui en rendent la programmation plus simple et plus sûre. Il bénéficied'une grand e bibliothèque qui met à la disposition du développeur plusieurs paquetagesprêts à l'utilisation.

1.1 Les différentes bibliothèques disponibles pour NEAT :

1.1.1 NEAT4J:

NEAT4J est un framework Java qui implémente l'algorithme du NEAT tel que proposé par Kenneth O Stanley.

NEAT4J permet de créer nos propres expériences très facilement en permettant de former la population, puis utilisez le meilleur individu dans votre application.

1.1.2 ANJI:

La bibliothèque ANJI contient le code source Java pour implémenter NEAT. Notez qu'ANJI n'est pas basée directement sur le code source original de Kenneth Stanley de sorte qu'il peut

différer à certains égards. Il comprend les implémentations d'expériences pour XOR et Tic-Tac-Toe.

1.1.3 JNEAT

C'est une version JAVA de la méthode neuro-évolutionnaire des topologies augmentés, écrite par Ugo Vierucci, basée sur la version originale en C++ de Kenneth Stanley. Elle comprend une interface graphique, et les implémentations d'expériences pour XOR et la parité de 3 bits.

1.2 Implémentation avec JNEAT :

Dus au manque ou **l'inexistence** d'information et de documentation concernant NEAT4J, nous avons commencé par l'exclure des bibliothèques potentielles. On devait donc choisir encore entre ANJI et JNEAT notre choix s'est tourné vers JNEAT pour son utilisation la plus courante et aussi, par le faite qu'elle soit recommandée par le créateur de NEAT lui-même O.Stannley.

1.2.1 Utilisation :

Pour pouvoir utiliser la bibliothèque JNEAT nous avons générer son JAR après avoir sélectionné les packages importants dont on a besoin vu qu'elle comporte plusieurs package qui n'ont pas tous un rôle dans nos expérimentations. Pour pouvoir exécuter notre expérimentation il nous a fallu importer le JAR « JNEAT » dans l'environnent de développement.

1.3 Présentation de notre application :

Notre application est un simulateur qui consiste à faire l'évaluation en temps réel des règles d'ordonnancement c'est-à-dire un simulateur qui offre principalement une évaluation des décisions possibles à la résolution d'un problème donné en temps réel.

1.3.1 L'interface principale du simulateur :

Cette interface a pour rôle la simulation de la cellule de production de valencienne comme représenté dans la figure suivante :

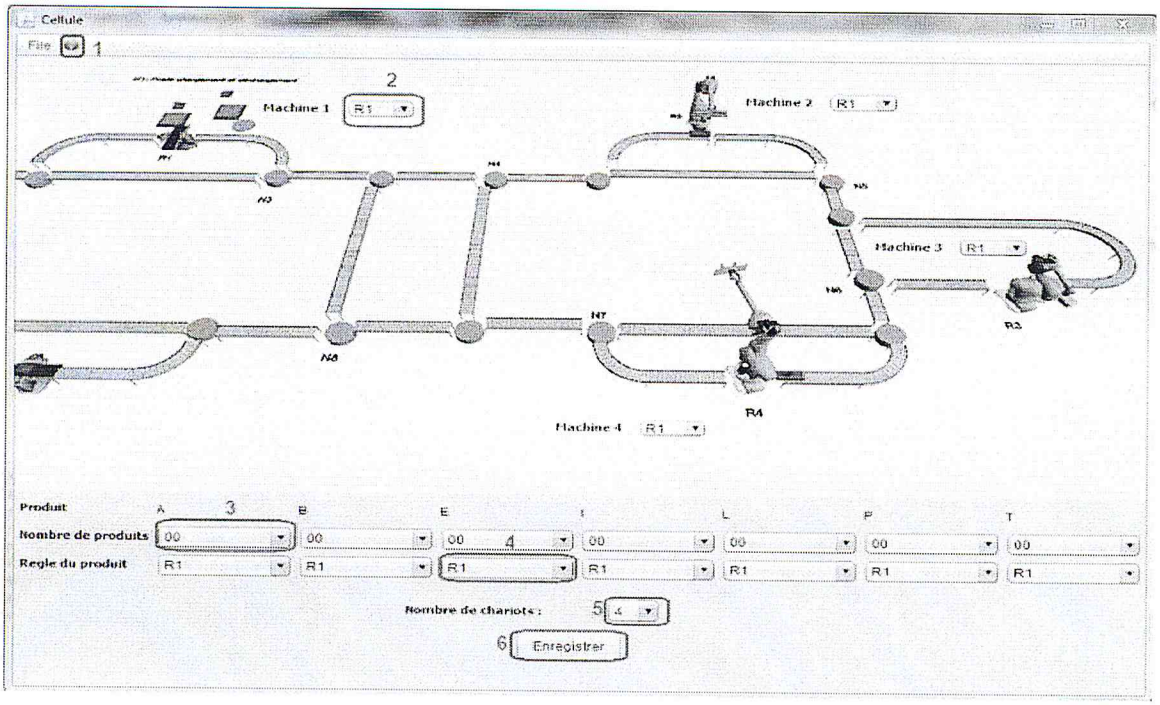


Figure 1 : Interface de la cellule de production de valencienne

Sur cette interface l'utilisateur pourra insérer les données nécessaires pour créer un scénario potentiel, et ainsi simulé le comportement de la cellule de production. C'est ainsi qu'il choisira les données à partir de notre interface comme suit:

1. En cliquant sur le bouton (1) il aura accès à une explication des règles pour une meilleure compréhension et donc proposé un scénario concret. La figure suivante représente.

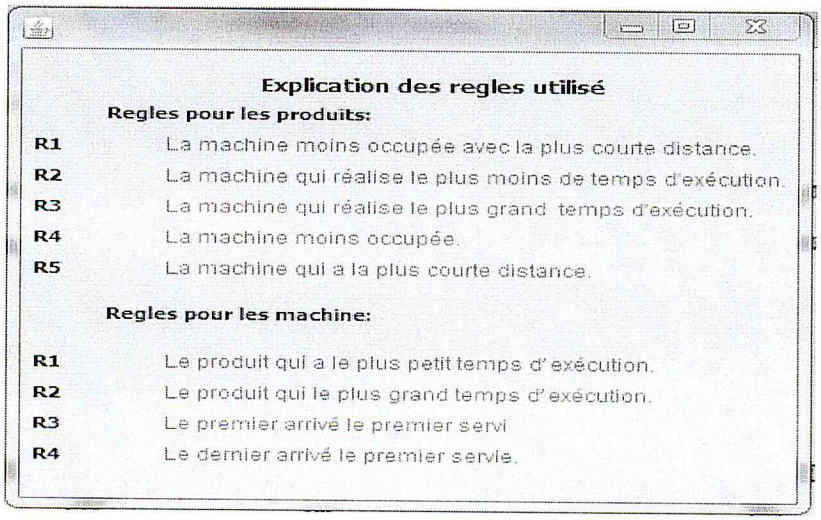


Figure 2: La fenêtre d'explication des règles.

2. En cliquant sur le bouton (2), l'utilisateur aura accès à une liste pour choisir les règles d'ordonnancement utilisé par les machines : (voir la figure).

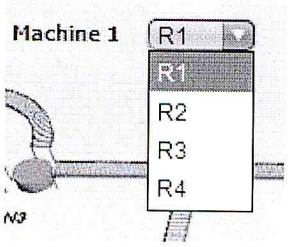


Figure 3: Le bouton de choix des règles machine.

3. En cliquant sur le bouton (3), l'utilisateur aura accès à une liste (voir la figure) pour choisir le nombre de produits qu'il voudrait faire entrer dans la cellule pour chaque type de produits :

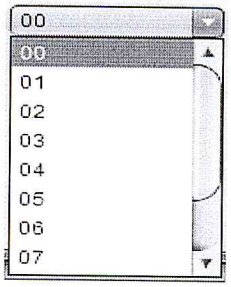


Figure 4 : bouton choix de Nombre de produit pour chaque type.

4. En cliquant sur le bouton (4), l'utilisateur aura accès à une liste pour choisir les règles d'ordonnancement utilisé pour chaque type de produit.
5. En cliquant sur le bouton (5), l'utilisateur aura accès à une liste pour choisir le nombre de chariots qui seront dans la cellule simultanément.
6. En cliquant sur le bouton (6) d'enregistrement. Notre simulateur calcule le temps d'exécution du scénario choisie et l'envoie à l'utilisateur dans une interface de résultats (voir la figure 20).

1.3.2 L'interface des résultats :

L'interface de résultat renvoie à l'utilisateur les résultats du temps d'exécution du scénario choisie.

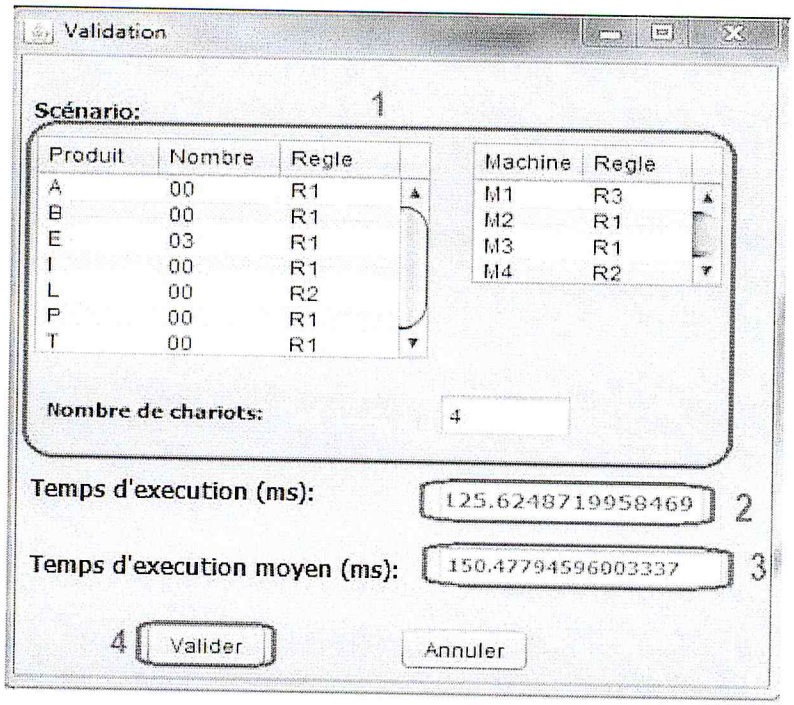


Figure 5: Interface des résultats.

1. On voit dans la partie (1), le scénario choisi par l'utilisateur dans l'interface précédente.
2. On voit dans la partie (2), le résultat calculé par le meilleur réseau de neurones.
3. On voit dans la partie (3), le résultat calculé par la moyenne des résultats des meilleurs réseaux de neurones.
4. En cliquant sur le bouton (4) d'enregistrement, le scénario et son résultat seront enregistrés dans un tableau (voir la figure21).

1.3.3 Interface de récapitulation :

Dans cette interface on peut retrouver un récapitulatif des scénarios testés précédemment et leur résultats :

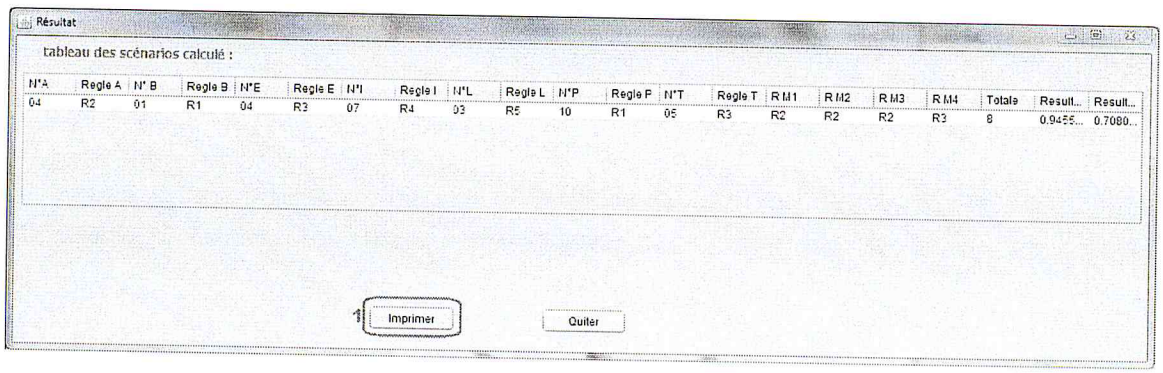


Figure 6: Interface de récapitulation.

1. Le bouton (1) dans la figure précédente permettra à l'utilisateur d'imprimer le tableau à l'aide de la fenêtre représenté dans la figure suivante:

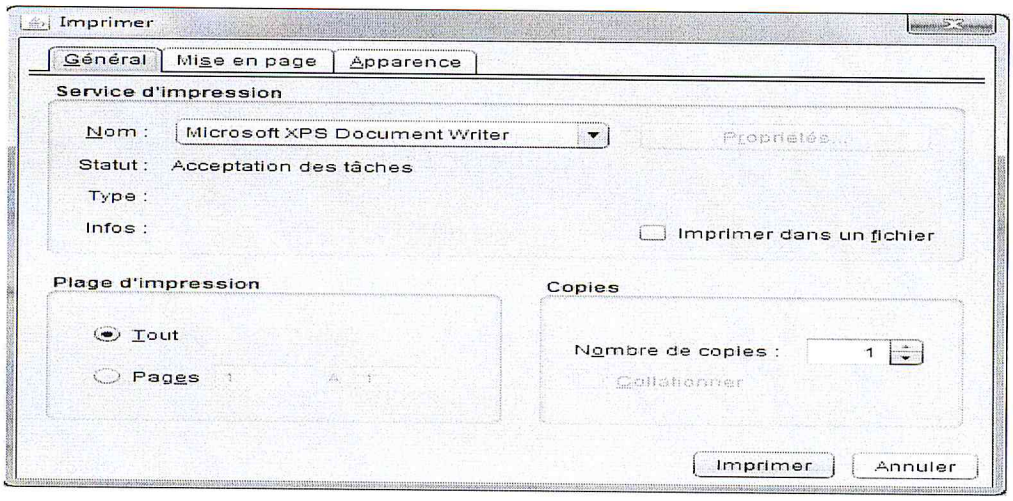


Figure 7: Fenêtre d'impression

Ainsi nous venons de faire le tour de notre application en montrant son fonctionnement.

1.4 Test et résultat

1.4.1 Apprentissage et validation :

Pour l'apprentissage de notre réseau de neurone nous avons utilisé une base d'apprentissage de 19900 exemples. Dans l'espoir d'avoir de meilleurs résultats de génération en génération. Voici ci-dessous un graphe exprimant l'évolution de l'erreur et celle du fitness de génération en génération.

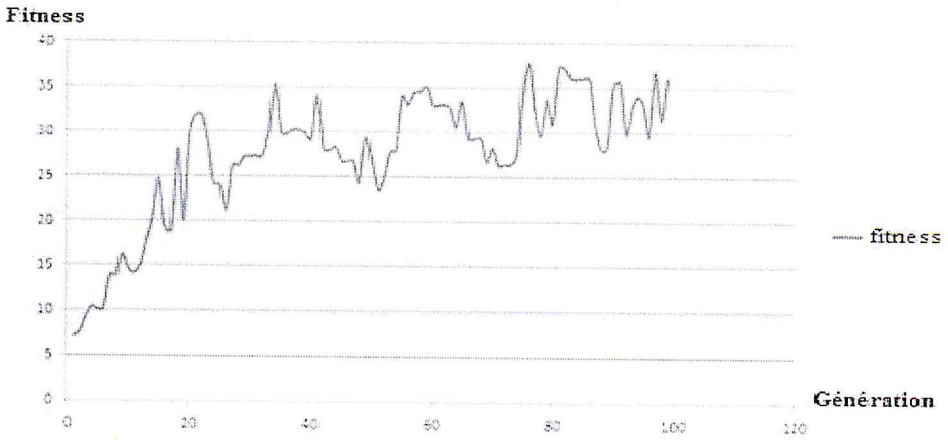


Figure 8: L'évolution du fitness.

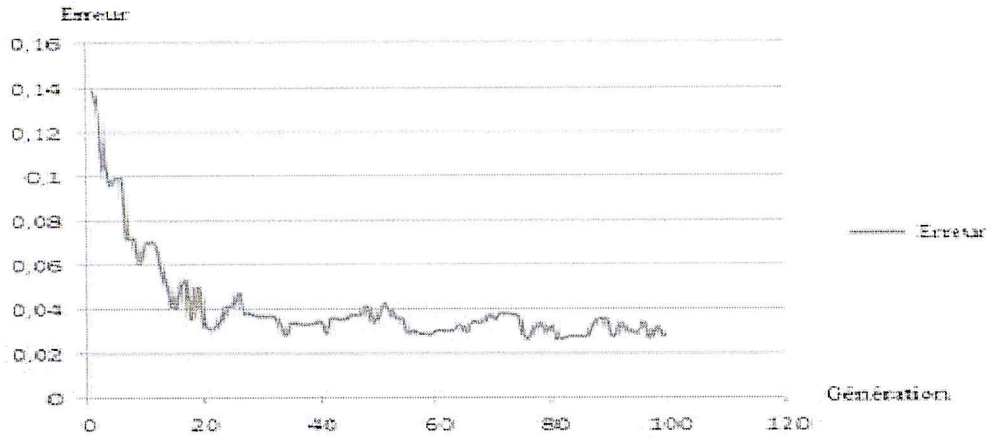


Figure 9: L'évolution de l'erreur

On peut très bien voir que le fitness augmente en avançant dans les générations et que l'erreur diminue. Après cette étape d'apprentissage nous somme passé à la validation, le tableau suivant représente les résultats de la validation pour les mêmes valeurs des sorties désirés.

La première colonne est la validation faite à partir du meilleur réseau avec le meilleur fitness. Et la seconde colonne est une autre manière de validation qui a été calculé par une moyenne des meilleurs organismes de la dernière génération.

Sorties désirés	Sorties de la 1ere validation	Sorties de la 2eme validation
0.0706	0.10266742804759552	0.09194096677758394
0.2258	0.22087945057653732	0.2521633497488449
0.2264	0.20763573311201072	0.21677943322940735
0.0236	0.07333477123458916	0.05990678903655463
0.1966	0.16307655429621204	0.1760283097133637
0.1226	0.11688073089578621	0.12367565497650068
0.0276	0.07889250749489966	0.08296844819992276
0.1056	0.1388123153805211	0.10504159513112836
0.2347	0.24503646019248126	0.2364227678532037
0.1302	0.13767856097420336	0.16129554395984114
0.0826	0.11842841357734096	0.12640668585752055

0.1386	0.14440358478577092	0.13302650578474273
0.1587	0.16270401697137055	0.20600445048202454
0.1808	0.18902865036252145	0.20224496357940316
0.3765	0.2970123279552554	0.26643219389316947
0.0542	0.08835439968772484	0.10193516858803553
0.1566	0.13321066950754953	0.14044318930324223
0.1645	0.15728427557739458	0.20253790538553929
0.0526	0.07043591434290002	0.08634916354424445
0.1105	0.1353597962438388	0.12785866609602645

Tableau 5 : La validation

1.4.2 Jeu d'essaies de notre application :

Si l'utilisateur veut tester un scénario il doit premièrement attribuer des règles aux 4 machines, ensuite choisir les nombre correspondant à chaque type de produits et sa règle de priorité. Dans les tableaux suivants nous donnons les règles de test réalisé par le simulateur.

Machine	Règle
Machine1	R4
Machine2	R3
Machine3	R3
Machine4	R2

Tableau 6: Règle des machines

Produit	A	B	E	I	L	P	T
Nombre	03	03	02	07	10	09	07
Règle	R3	R1	R5	R4	R1	R2	R1

Tableau 7: Initialisation des produits

La réalisation de ce test sur le simulateur est donnée dans la figure ci-dessous :

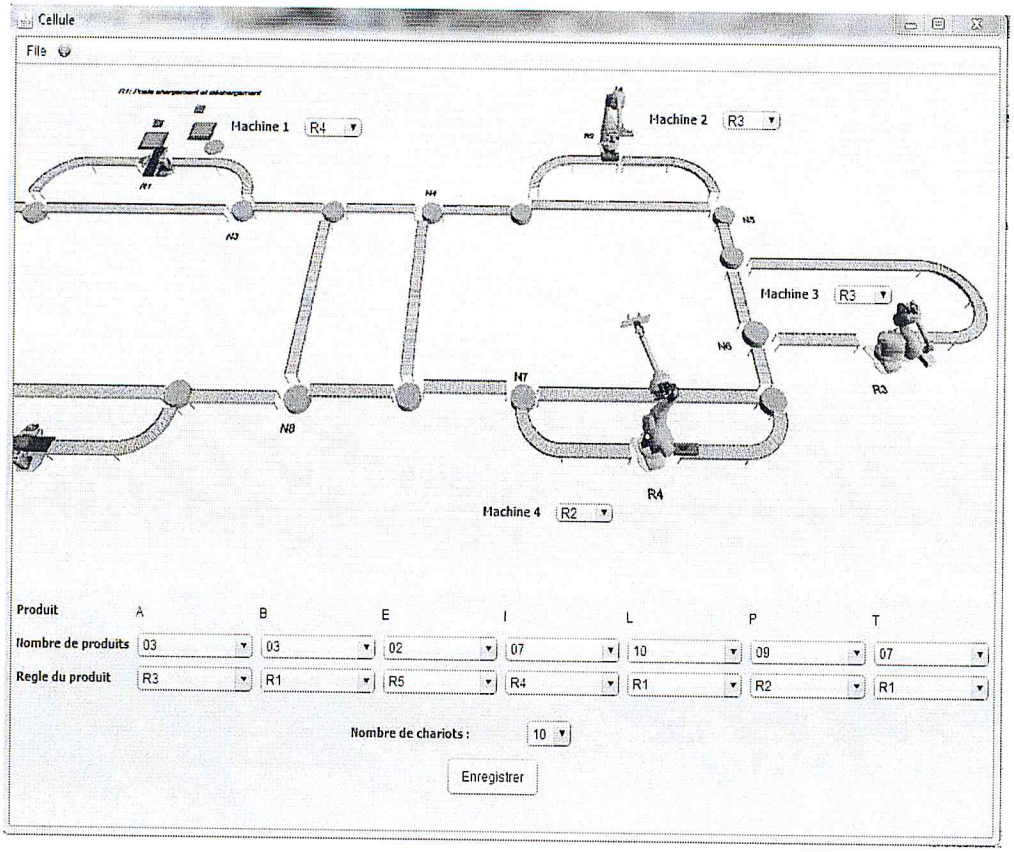


Figure 10: Un jeu de test sur la cellule de valencienne

Après avoir enregistré les entrées, le simulateur calcule le temps d'exécution ainsi que le temps d'exécution moyen (temps d'exécution en ms). La figure suivante représente les valeurs résultantes pour le scénario choisi précédemment.

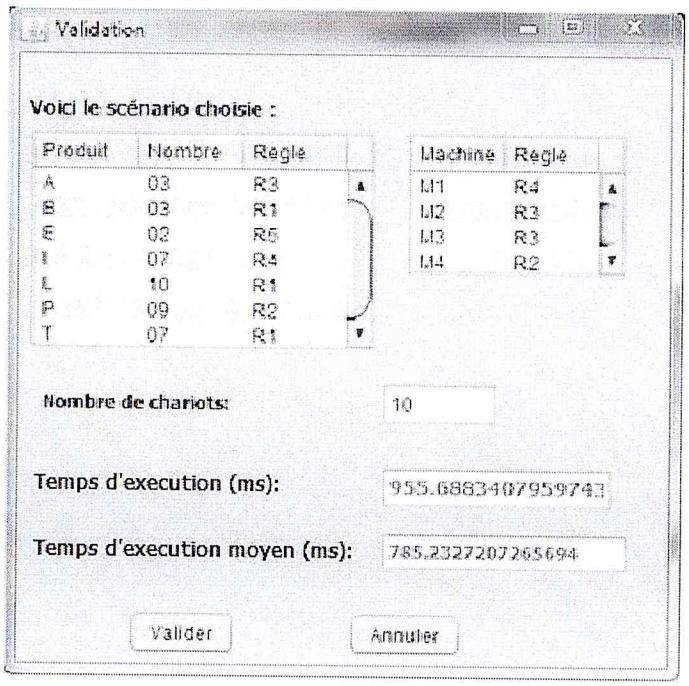


Figure 11: Calcule de temps d'exécution

On remarque ra que le premier scénario prendrai moins de temps et sera donc le scénario choisi par l'utilisateur.

Si l'utilisateur de notre application voudrai choisir entre deux scénarios, tout ce qu'il doit faire c'est de lancer la simulation de ces derniers, le simulateur va lui renvoyé le temps d'exécution de chacun des deux scénarios. Avec une simple comparaison l'utilisateur peut choisir le scénario avec le moins de temps d'exécution dans la cellule de production.

1.5 Etat de notre projet :

A la fin de la réalisation de notre projet nous avons pu identifier certain points essentiels pour décrire l'état de notre projet :

Premièrement l'étape la plus importante est d'ajuster les paramètres de façon à obtenir des meilleurs résultats. Dans notre cas et après plusieurs essaies de contrôle nous avons constaté que les paramètres sont bien ajusté.

Deuxièmement, Sachant que NEAT favorise les organismes ayant le fitness le plus élevé et qu'il est inversement proportionnel avec l'erreur et après avoir testé notre programme nous pouvons bien confirmer cette hypothèse par le fait qu'il augmente de génération en génération pour avoir des meilleurs résultats.

Troisièmement, malgré l'obtention d'une erreur descendante au fil des générations et que cette erreur est réduite jusqu'à 0.02, nous cherchons encore comment avoir de meilleurs résultats. Rappelons que cette erreur et la différence entre la sortie désirée et la sortie du notre réseau.

Finalement NEAT se révèle au final être une approche valable pour la simulation mais nous devons peut être apporté des modifications à notre implémentation pour améliorer les résultats de sortie obtenus.

2 Conclusion

Dans ce chapitre nous avons présenté brièvement l'environnement de programmation. Nous avons aussi décrit l'interface de notre simulateur avec toutes les fonctionnalités qu'elle permet d'accomplir, ainsi que les résultats des différentes expérimentations réalisées. L'objectif est d'explorer les performances des stratégies qu'on a utilisées, d'un côté, et de valider notre implémentation d'une approche neuro-évolutionnaire adaptées à la résolution du problème de prise de décision dans les systèmes de production automatisés.

CONCLUSION GÉNÉRALE

Conclusion générale

Durant ces dernières années l'utilisation de l'intelligence artificielle s'est imposée d'une manière très impressionnante dans le domaine du pilotage des systèmes de production, cela est dû à la croissance des problèmes de pilotage et l'imprévision dans ce domaine.

Prendre une décision n'est jamais facile ou bien évident à faire surtout pour gérer un système de production, ou, une mauvaise décision coûtera des pertes à la société. C'est pourquoi nous proposons un outil d'aide à la décision par le biais de la simulation en temps réel. Cette simulation évalue un ensemble de décision pour trouver la solution d'un problème de production mais à cause du cout de ses simulateurs l'utilisation des réseaux de neurone dans la simulation paraissent nécessaire.

Ce projet a pour but d'expliqué l'objectif principal de notre travail qui consiste principalement la conception et la mise en œuvre d'une approche neuro-évolutionnaireen particulier NEAT en tant que modèle de substitution pour l'aide à la décision par la simulation dans le cadre du pilotage tactique des systèmes de production automatisés.

Durant le présent projet de fin d'études, nous avons traité le pilotage des systèmes de production automatisée et l'évaluation en temps réel des règles d'ordonnancement par la simulation pour l'aide à la décision. Ensuite, nous avons décrit d'une manière détaillée l'approche neuro-évolutionnaire NEAT « NeuroEvolution of Augmenting Topologies» utilisé dans notre projet passant par les algorithmes évolutionnaire. Puis la conception et à l'explication de la procédure abordé pour acheminé ce projet avec la méthode NEAT étape par étape. Enfin nous avons consacré une partie de présentation de notre interface de notre outil, tests et résultats.

Ce travail nous a permis de constater que l'approche NEAT donnait des résultats proches de la réalité avec une erreur réduite, mais ces résultats peuvent être améliorés.

Les principales perspectives de recherche qui apparaissent à l'issue de ce projet concernent la réutilisabilité de notre travail est de pouvoir minimiser encore le temps d'exécution et accompagner notre solution d'une application androïde qui jouera sur la portabilité de l'outil d'aide à la décision.

Bibliographie

AMMAR Yessin, M. (2007). *MISE EN ŒUVRE DE RESEAUX DE NEURONES POUR LA MODELISATION DE CINETIQUES REACTIONNELLES EN VUE DE LA TRANSPOSITION BATCH/CONTINU.* Tunisie: Ecole Nationale d'Ingénieurs de Sfax,.

CHIKH. (2000). *utilisation de RNA qui implémante FPGA pour la roconnaissance du diabete.* Tlemcen, algérie.

Drchal, J. (2006). *Evolution of Recurrent Neural Networks.* CZECH TECHNICAL UNIVERSITY in PRAGUE.

Esquirol, &., & LOPEZ. (1999). *l'ordonnancement.* paris: Economica.

FRMLING, K. (1996). *Modélisation et apprentissage des préférences par réseaux de neurones pour l'aide a la décision multicritère.* French: Environmental Sciences. INSA de Lyon.

GOSSELIN, B. (1995). *APPLICATION DE RESEAUX DE NEURONES ARTIFICIELS A LA RECONNAISSANCE AUTOMATIQUE DE CARACTERES MANUSCRITS.* Faculté Polytechnique de Mons.

Hentous, H., & Guinet. (1997). *A new heuristic to minimize completion.* lyon: International conference on industrial engineering and production.

HOUARI. (2012). *Planification et Ordonnancement en temps réel d'un Job shop en utilisant l'Intelligence Artificielle.* Algérie: Université de Tlemcen.Faculté de Technologie.

LEILA, S. (2007). *Optimisation MultiObjectifs par Programmation généique.* Algérie: l'université de BATNA; Faculté des sciences de l'ingénieur - Département d'informatique.

MIRDAMADI, S. (2009). *Modélisation du processus de pilotage d'un atelier en temps réel.* france: 'Institut National Polytechnique de Toulouse.

NOYES. (1987). *Approches méthodologiques pour l'aide à la conception et à.* Toulouse,France: Institut national polytechnique de Toulouse.

PANZOLI, D. (2003). *Simulation comportementale par réseau de neurones et apprentissage par algorithme génétique.* France .

PANZOLI, D. (2003). *Simulation comportementale par réseau de neurones et apprentissage par algorithme génétique.*

REBOUH, R. (2011). *Formulation des Bétons avec Ajout par l'Utilisation des Réseaux de Neurones.* algérie : Université Hassiba Ben Bouali de Chlef.

Risto Miikkulainen, K. O. (2002). *Efficient Reinforcement Learning through Evolving Neural Network Topologies.* San Francisco: In Proceedings of the Genetic and Evolutionary Computation Conference.

ROHEE, B. (2005). *Répartition dynamique d'activités sur un Automate Programmable Industriel.* France: Université de NANCY I.

SOUIER, M. (2012). *Métaheuristiques pour la manipulation de routages alternatifs en temps réel dans un Job Shop.* Algérie: Université Abou Bakr Belkaid Faculté des sciences de l'ingénieur.

Stanley, K. O. (2007). *Evolving Neural Networks through Augmenting Topologies.* Massachusetts Institute of Technology.

TAGHEZOUT, N. (2011). *Conception et Développement d'un système multi-agent d'Aide à la Décision pour la gestion de production dynamique.* Université Toulouse.

TOUZET, C. (1992). *LES RESEAUX DE NEURONES.*

TRENTESAUX, D. (1996). *Conception d'un système de pilotage distribuée, sup ervisée et multicritère pour les systèmes automatisés de production.* france: Institut National Polytechnique .

YOUNES, M. (2011). *Aide à la décision d'ordonnancement inter-entreprises dans le.* Algérie: UNIVERSITE HADJ LAKHDAR BATNA.