Contribution de l'apprentissage par simulation à l'auto-adaptation des systèmes de production

Lorena Silva Belisário

▶ **To cite this version:**

Lorena Silva Belisário. Contribution de l'apprentissage par simulation à l'auto-adaptation des systèmes de production. Autre. Université Blaise Pascal - Clermont-Ferrand II, 2015. Français. NNT : 2015CLF22617 . tel-01325229

HAL Id: tel-01325229

https://theses.hal.science/tel-01325229

Submitted on 2 Jun 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre : D.U. 2617
EDSPIC : 718

Université Blaise Pascal - Clermont II

École doctorale des Sciences Pour l'Ingénieur

THÈSE

en vue de l'obtention du grade de

Docteur d'Université

Spécialité : Informatique - Productique

Présentée et soutenue par

Lorena Silva Belisário

Contribution de l'apprentissage par simulation à l'auto-adaptation des systèmes de production

---

Thèse dirigée par Henri Pierreval

préparée au LIMOS - UMR CNRS 6158

soutenue publiquement le 12 novembre 2015

Jury :

Président :	Pr. David R. C. Hill	- Université Blaise Pascal, France
Rapporteurs :	Pr. Patrick Burlat	- École Nationale Supérieure de Mines de Saint-Étienne, France
	Pr. Damien Trentesaux	- Université de Valenciennes et du Hainaut-Cambrésis, France
Examinateur :	Pr. Mauricio Cardoso de Souza	- Universidade Federal de Minas Gerais, Brésil
Directeur :	Pr. Henri Pierreval	- Institut Français de Mécanique Avancée, France

Résumé

Pour rester performants et compétitifs, les systèmes de production doivent être capables de s'adapter pour faire face aux changements tels que l'évolution de la demande des clients. Il leur est essentiel de pouvoir déterminer quand et comment s'adapter (capacités, etc.). Malheureusement, de tels problèmes sont connus pour être difficiles. Les systèmes de production étant complexes, dynamiques et spécifiques, leurs gestionnaires n'ont pas toujours l'expertise nécessaire ni les prévisions suffisantes concernant l'évolution de leur système.

Cette thèse vise à étudier la contribution que peut apporter l'apprentissage automatique à l'auto-adaptation des systèmes de production. Dans un premier temps, nous étudions la façon dont la littérature aborde ce domaine et en proposons un cadre conceptuel dans le but de faciliter l'analyse et la formalisation des problèmes associés. Ensuite, nous étudions une stratégie d'apprentissage à partir de modèles qui ne nécessite pas d'ensemble d'apprentissage. Nous nous intéressons plus précisément à une nouvelle approche basée sur la programmation génétique linéaire visant à extraire des connaissances itérativement à partir d'un modèle de simulation pour déterminer quand et quoi faire évoluer.

Notre approche est implémentée à l'aide d'Arena et μGP. Nous l'appliquons à différents exemples qui concernent l'ajout/retrait de cartes dans un système à flux tiré, le déménagement de machines ou encore le changement de politique de réapprovisionnement. Les connaissances qui en sont extraites s'avèrent pertinentes et permettent de déterminer en continu comment chaque système peut s'adapter à des évolutions. De ce fait, elles peuvent contribuer à doter un système d'une forme d'intelligence. Exprimées sous forme d'un arbre de décision, elles sont par ailleurs facilement communicables à un gestionnaire de production. Les résultats obtenus montrent ainsi l'intérêt de notre approche tout en ouvrant de nombreuses voies de recherche.

Mots-clés : Systèmes de production, auto-adaptation, aide à la décision, extraction de connaissances, apprentissage automatique, simulation, programmation génétique linéaire.

Abstract

Manufacturing systems must be able to continuously adapt their characteristics to cope with the different changes that occur along their life, in order to remain efficient and competitive. These changes can take the form of the evolution of customers demand for instance. It is essential for these systems to determine when and how to adapt (e.g., through changes in capacities). Unfortunately, such issues are known to be difficult. As manufacturing systems are complex, dynamic and specific in nature, their managers do not always have all the necessary expertise nor accurate enough forecasts on the evolution of their system.

This thesis aims at studying the possible contributions of machine learning to the self-adaptation of manufacturing systems. We first study how the literature deals with self-adaptation and we propose a conceptual framework to facilitate the analysis and the formalization of the associated problems. Then, we study a learning strategy relying on models, which presents the advantage of not requiring any training set. We focus more precisely on a new approach based on linear genetic programming that iteratively extracts knowledge from a simulation model.

Our approach is implemented using Arena and μGP. We show its benefits by applying it to increase/decrease the number of cards in a pull control system, to move machines or to change the inventory replenishment policy. The extracted knowledge is found to be relevant for continuously determining how each system can adapt to evolutions. It can therefore contribute to provide these systems with some intelligent capabilities. Moreover, this knowledge is expressed in the simple and understandable form of a decision tree, so that it can also be easily communicated to production managers in view of their everyday use. Our results thus show the interest of our approach while opening many research directions.

Keywords : Manufacturing systems, self-adaptation, decision support, knowledge extraction, machine learning, simulation, linear genetic programming.

Remerciements

Je tiens tout d'abord à remercier Patrick Burlat, David Hill et Damien Trentesaux pour avoir accepté d'évaluer mon travail de thèse et d'assister à ma soutenance.

Je souhaite remercier tout particulièrement mon ancien professeur Mauricio Cardoso de Souza, de l'UFMG, au Brésil. C'est lui qui m'a donné le tout premier élan vers la recherche il y a déjà bien des années. Aujourd'hui et malgré la distance, il n'a pas hésité à m'accompagner dans l'accomplissement de cette thèse.

Mes pensées vont aussi à Leonardo Santiago et Carlos Roberto Venâncio de Carvalho, également de l'UFMG, qui m'ont écouté, soutenu et motivé lors de quelques échanges au cours de ces trois dernières années. J'ajoute également Philippe Mahey, Andréa et Christophe Duhamel que j'ai connus à l'occasion de mon échange universitaire et avec qui j'ai eu le plaisir de garder contact.

Merci à Vincent Barra et Alain Quilliot qui ont participé aux deux comités de suivi de cette thèse et dont les remarques constructives m'ont permis de mieux avancer. Et je ne pourrais certainement pas oublier Claude Mazel qui s'est montré d'une grande patience et m'a rassurée par ses explications dans un moment difficile.

Un grand merci à Alberto Tonda, chargé de recherche à l'INRA, qui a toujours été ouvert à mes questions sur μGP. Lui seul sait combien elles furent nombreuses !

Bien évidemment, merci à Henri Pierreval, mon directeur de thèse. Tout d'abord pour m'avoir proposé ce sujet de thèse et ainsi m'avoir permis d'être là aujourd'hui. Mais aussi pour le temps qu'il a consacré à mon travail et pour le soutien et l'expérience qu'il m'a apportés.

Mes remerciements vont également à tout le personnel de l'IFMA, de l'ISIMA et du LIMOS que j'ai côtoyé pendant ces années.

Une thèse est un moment particulier dans la vie qui, comme beaucoup le savent, n'est pas toujours facile. Je ne pourrais donc jamais oublier toutes ces personnes qui ont cru en moi et qui, chacune à sa façon, m'ont permis d'arriver au bout de cette thèse. Bien qu'il ait fallu subir leurs questions récurrentes et souvent angoissantes du genre « *Ça avance ?* » ou « *Quand est-ce que tu soutiens cette thèse ?* ».

Merci à tous mes amis, plus ou moins proches mais qui ont toujours été là pour me soutenir. Je remercie tout spécialement Jo, mon témoin de mariage, qui a « un peu » subi mes humeurs ces dernières années. Il m'a souvent écouté et conseillé. Il a également apporté une grosse contribution dans l'aspect final de ce manuscrit. C'est un exemple de chercheur pour moi. De plus, il cherche maintenant à s'occuper de ma carrière de chercheuse !

Merci à Clément pour sa disponibilité, à Guillaume pour ses conseils en Latex et aussi à Estelle pour tous les moments partagés. À Akram, Nesrine et Aïcha avec qui j'ai partagé mon bureau. À Achraf et Bamba, jamais très loin non plus. Merci à Christèle et Thérèse que j'ai connues lors d'un module de l'école doctorale et que j'ai le plaisir de revoir par moments. À Maxime, Benjamin et Jonathan pour notre escapade lors de la conférence à Bordeaux. À Sahar, Kaoutar et Karima qui ont rendu le module aux Karellis encore plus agréable. Merci encore à Jean-Charles qui m'a mis son bureau à disposition et m'a même aidé à trouver une façon de structurer l'un des chapitres de ma thèse. À Jean-Philippe qui s'est toujours intéressé à l'avancement de ma thèse lorsque l'on se rencontrait. À Simon, Isabelle, Alexandre, Benoît, Hélène, Cristine, Romain, Florian, Thomas, Nicolas, Christine, enfin, à tous ces amis de JB qui deviennent de plus en plus les miens.

Merci aux brésiliens de passage et avec qui j'ai pu partager quelques moments qui m'ont permis de sentir mon pays un peu plus proche. Et que dire des tous ces brésiliens que j'ai « laissés » mais qui occupent toujours une place si importante dans ma vie ? Danilo, Nathália, Matheus, Gustavo, Leandro, Luciana, Dui, Aretha, Ana Joana, Lucas Oliveira, Laiza, Aline... Je ne peux que leur dire merci pour leurs encouragements mais surtout parce qu'ils ne m'oublient pas.

Enfin, les mots les plus simples et plus forts, je les adresse à ma famille. En particulier à ma mère, mon père et mes frères. Malgré la distance qui nous sépare depuis quelques années, leur confiance, leur tendresse et leur amour ne me quittent jamais. Chacun à leur manière, ils ont toujours été présents lorsque j'en ai eu besoin, dans les moments de joie comme de tristesse. *Obrigada !* Je ne leur ai aussi peut-être pas assez dit... *Amo vocês !*

Merci à ma petite filleule Ana Clara, qui n'est plus si petite que ça et que j'aurais tant aimé voir grandir. Merci à ma marraine qui me rappelle souvent son affection. À tous mes oncles, tantes, cousins, cousines et plus particulièrement à mes grand-mères qui, je le sais, sont très fières de moi. Une pensée également à mes deux grands-pères qui ne sont plus de ce monde mais qui, j'en suis sûre, veillent sur moi.

Merci à ma belle-famille qui m'a reçu les bras ouverts. Ils ont toujours manifesté de l'intérêt pour ce que je fais et ce fut important pour moi de sentir que j'avais leur soutien et leurs encouragements.

Je tiens également à remercier toute personne qui a contribué, de près ou de loin, à l'accomplissement de cette thèse et que, pressée par le temps, j'aurais oublié de mentionner.

Mais surtout, tous mes remerciements à mon mari, Jean-Baptiste, sans lequel cette thèse n'aurait jamais vu le jour. Jamais je ne pourrai assez te remercier, mon amour. Depuis le début de cette thèse, tu as toujours cru en moi, même plus que moi-même. Tu as vécu avec enthousiasme mes moments de joie et tu m'as soutenu (et supporté) comme seul toi sais le faire pendant mes nombreuses périodes de doutes. Que pourrais-je dire de tes multiples encouragements, de tes précieux conseils ? Sans parler, à la fin, de la relecture

scrupuleuse bien que souvent pénible de ce manuscrit. Merci pour tes corrections et pour m'avoir aidé à clarifier mes mots et même ma pensée parfois confuse. Tu as toujours tout fait pour m'aider. Tu fais tout ce que tu peux pour moi, j'en ai conscience et je t'en suis reconnaissante. Merci pour tout ce que tu m'apportes au quotidien. Ton attention, ton soutien et ton amour me portent et me guident tous les jours.

Table des matières

Table des figures

Liste des tableaux

Introduction Générale

Les systèmes de production de biens et de services fonctionnent de plus en plus rarement dans des environnements stables pour lesquels une organisation définitive optimale peut être recherchée. S'ils veulent rester compétitifs, leur quête de performance doit faire face à une difficulté majeure : la complexité et surtout l'incertitude qui caractérisent leur environnement plus que jamais changeant. Ceci constitue l'objectif de nombreux travaux de recherche étant donnés les verrous scientifiques derrière cet enjeu industriel.

Aujourd'hui, il nous est de plus en plus difficile de trouver une organisation qui reste optimale sur le long terme, c'est même un défi scientifique reconnu. La variation des marchés et des besoins des clients, les nouveautés apportées aux produits ou les aléas de fabrication imposent que les systèmes de production soient capables de s'y adapter au mieux. Ce besoin se ressent aux différents niveaux d'une organisation (stratégique, tactique ou opérationnel) et peut prendre différentes formes (adaptations physiques ou logiques). Ainsi, des capacités peuvent être redimensionnées, des ressources réaffectées et des paramètres de gestions peuvent également évoluer (politique de stockage, etc.).

Malheureusement, cette solution, que l'on peut caractériser par la mise en place d'une « organisation dynamique », demande une capacité d'adaptation à l'environnement, ce qui reste une question complexe. Ceci est dû à la complexité même des systèmes de production, qui se traduit généralement par leurs aspects combinatoire, stochastique, dynamique ou encore multi-objectif. Par ailleurs, les difficultés à gérer ces différents aspects simultanément ne font que s'accroître si l'on cherche à prendre en compte les changements qui peuvent intervenir et l'incertitude liée à l'environnement. C'est donc un enjeu de taille pour les entreprises : comment peuvent-elles améliorer leur compétitivité ou, autrement dit, améliorer en permanence la performance de leur processus ?

Les décideurs (ici gestionnaires de production) doivent gérer un processus dit d'adaptation en fonction des informations venant de l'environnement et de la production. Mais comment déterminer quand un système requiert des changements ? De même, une fois le besoin détecté, comment déterminer quels sont les changements précis à mettre en œuvre ? Afin de mener à bien ce processus d'adaptation, il est souhaitable et parfois même nécessaire d'avoir à disposition des moyens adéquats capables d'aider à la prise de décision. Cette aide à la décision peut se traduire par ce que l'on appelle des logiques décisionnelles permettant de connaître la façon dont le système doit être adapté (voire lui permettant de s'adapter de façon autonome) pour satisfaire au mieux ses objectifs de performance (coûts, satisfaction des clients, etc.). Toutefois, ces logiques sont méconnues. Les « bonnes pratiques » reconnues et pouvant être appliquées sont en effet limitées, surtout si l'on prend en compte les spécificités des systèmes de production : ce qui est valable pour un système ne l'est pas forcément pour un autre. Une solution consiste alors à faire appel à une approche d'intelligence artificielle reposant sur l'apprentissage pour extraire les lo-

giques recherchées. Il s'avère que, malgré son potentiel, ce type d'approche a fait l'objet de peu de recherches dans le domaine. Dans ce qui suit, le terme apprentissage sous-entend l'apprentissage dit artificiel ou automatique.

Dans la plupart des techniques d'apprentissage, l'objectif est d'extraire des connaissances à partir de différents types de données. En général, ces approches utilisent des exemples fournis par des experts du domaine, des observations disponibles ou des bases de données. Les données pertinentes doivent ainsi être disponibles, et cela à un coût raisonnable. Malheureusement, en ce qui concerne la gestion et le contrôle d'un système complexe tel qu'un système de production (mais aussi des hôpitaux, des réseaux de trafic routier ou des chaînes logiques), les informations disponibles dans la pratique s'avèrent souvent insuffisantes pour les décisions que nous souhaitons assister. Les experts n'ont pas toutes les connaissances requises et ne sont donc pas en mesure de fournir des exemples ou des cas (suffisamment représentatifs) à partir desquels il serait possible d'apprendre. De plus, il n'est pas possible de disposer d'informations sur le comportement du système qui permettraient de prédire que, dans une situation donnée, une décision précise va conduire à de bons résultats (par rapport à une mesure de performance donnée), tandis que dans d'autres situations, elle ne sera pas satisfaisante.

Le comportement de nombreux systèmes complexes peut cependant être modélisé en utilisant des méthodes telles que la simulation, la théorie des files d'attente ou encore les réseaux de Pétri. Ces modèles constituent une source intéressante de connaissance : ils permettent de représenter et ainsi de prédire le comportement dynamique d'un système dans différentes situations (y compris des situations inhabituelles) lorsque des observations réelles ne peuvent pas être collectées ou s'avèrent trop coûteuses. Afin d'extraire des connaissances sur la façon de prendre des décisions efficaces relatives à la gestion et au contrôle d'un système, il est alors possible d'apprendre à partir des expériences menées à l'aide de modèles. Pour cela, ces modèles doivent être capables de fournir diverses observations à moindre coût à partir desquelles l'apprentissage sera possible et ce, dans une grande variété de circonstances. On parlera de paradigme d'apprentissage à partir de modèles.

C'est ce que nous proposerons de faire par la suite, sachant que nous ne prétendons pas pouvoir résoudre tous les problèmes d'adaptation avec une telle méthode. L'idée est plus précisément d'étudier comment cela pourrait être mis en place et quelle peut être la contribution de l'apprentissage en tant qu'aide intelligente à l'adaptation des systèmes de production, voire à leur auto-adaptation.

Dans le but de répondre à ces questions, ces travaux de thèse ont été organisés comme suit : dans un premier temps et selon une démarche méthodologique de thèse, nous avons souhaité mieux comprendre les problématiques autour de l'adaptation. Nous avons ainsi réalisé une analyse des travaux extraits de la littérature (Chapitre 1). Cette analyse nous a d'une part incités à proposer un cadre conceptuel du domaine (Chapitre 2) et nous a d'autre part permis de mieux nous positionner. Nous nous sommes alors intéressés

aux approches d'apprentissage, et plus précisément à leur utilisation conjointe avec la simulation (Chapitre 3). Pour permettre notre étude de la contribution de l'apprentissage par simulation dans le domaine, nous avons ensuite formalisé le problème d'adaptation, ce qui nous a permis d'en proposer une approche adéquate (Chapitre 4). Cette approche a enfin été appliquée à différents cas (Chapitre 5) afin de permettre d'identifier son potentiel (mais aussi ses limites) qui feront l'objet d'une discussion.

Chapitre 1

Adaptation des Systèmes de Production

1.1 Introduction

Les travaux de recherche sur les systèmes de production sont nombreux et portent sur divers domaines tels que leur dimensionnement, la définition d'une structure organisationnelle, la planification de la production ou encore la gestion de stock (voir par exemple [Dolgui et Proth 2006, Belisário *et al.* 2009, Salerno 2009, Surbier *et al.* 2013]). On remarque que ces travaux s'intéressent de plus en plus à l'aspect « adaptable » des systèmes, en faisant souvent références aux systèmes du futur [Van Brussel *et al.* 1999, Christo et Cardeira 2007, Yeom et Park 2012]. Il s'agit en réalité d'un besoin actuel.

Depuis les années 1990, la dynamique du marché et la forte concurrence imposent aux entreprises de pouvoir s'adapter à un environnement en constante évolution. Cependant, les systèmes de production existants ne permettent plus de répondre aux besoins, exigences, évolutions et aux nombreux challenges du monde industriel [National Research Council 1998, Reaidy 2003, ElMaraghy *et al.* 2012a]. Par conséquent, des efforts considérables ont été consacrés ces dernières décennies à les rendre plus agiles [Gunasekaran 1999, Tharumarajah 2003, ElMaraghy 2006, Deif et ElMaraghy 2007]. Cette agilité est nécessaire à plusieurs niveaux, comme expliqué dans [Lin *et al.* 2006, Zhang 2011]. Les entreprises doivent souvent faire face à des événements imprévus (et imprévisibles) au niveau opérationnel tels qu'une annulation ou modification de commande, la prise en compte d'une commande urgente, les nouveautés fréquemment apportées aux produits ou encore des aléas de production (pannes, etc.). Au niveau stratégique, elles sont aussi contraintes de s'adapter aux progrès technologiques et de suivre les évolutions du marché [Mirdamadi 2009]. Par ailleurs, elles doivent si possible et de préférence anticiper les tendances internes ou celles du marché, et donc être capables de les détecter. Même si les raisons (pour une reconfiguration) peuvent être des plus diverses et apparaître depuis le niveau de l'organisation virtuelle jusqu'à celui de l'atelier [Barbosa *et al.* 2011], elles peuvent être regroupées en quatre grandes catégories selon ce qu'elles visent, à savoir : la correction, l'adaptation, l'évolution ou l'amélioration du système [Ketfi 2004].

Dans un tel contexte et afin de rester compétitifs, les systèmes de production devraient être en mesure d'adapter leurs installations et leur organisation pour répondre au mieux aux divers changements possibles dans leur environnement grâce à une reconfiguration dynamique dite « à la volée » (sans interruption, reprogrammation ou redémarrage des

processus ou des composants) [Tharumarajah 2003, Leitão *et al.* 2012]. Malheureusement, leur capacité à s'adapter aux circonstances est connue pour être une question complexe qui implique des considérations matérielles et logicielles (*hardware* et *software*), mais aussi organisationnelles et humaines [Gunasekaran 1999, ElMaraghy 2006, Lin *et al.* 2006]. Le problème s'accentue si l'on considère les différentes natures et sources de complexité intrinsèques à ces systèmes [ElMaraghy *et al.* 2012b]. Parmi elles, citons les contraintes souvent involontaires imposées par les systèmes eux-mêmes au niveau du traitement des données ou de leur capacité de détection et d'action [Saenz de Ugarte 2009]. Par conséquent, cela soulève de nombreuses questions de recherche qui sont de plus en plus traitées dans la littérature scientifique.

Ce besoin de s'adapter rapidement et de façon continue à un environnement changeant implique différents paradigmes, à savoir : la reconfiguration, l'auto-organisation, l'(auto-)adaptation ou encore l'(auto-)évolution. Les systèmes qui visent à intégrer de tels paradigmes à leur fonctionnement, de plus en plus présents dans la littérature, constitueront alors l'essence de ce chapitre. L'objectif de la première partie est de présenter le cadre général dans lequel s'inscrit notre travail. Il s'agit de l'adaptation (ou auto-adaptation) des systèmes de production, impliquant leur pilotage et la capacité à déterminer comment les faire évoluer. Dans ce qui suit, nous donnerons quelques définitions proposées autour de ces systèmes qui peuvent s'adapter. L'objectif n'est pas de détailler des concepts déjà largement développés dans la littérature mais d'en fournir les bases puis une vision générale de la bibliographie du domaine. Celle-ci sera faite dans une deuxième partie en suivant deux grands axes : « théorique » et « pratique ». Cette démarche globale permettra de mieux nous positionner par rapport aux différentes contributions existantes.

1.2 Concepts liés aux changements dans les systèmes de production

Dans le contexte actuel créé par la compétition au niveau mondial, les changements technologiques et la dynamique du marché, les systèmes de production ne peuvent plus rester constants au cours du temps : ils doivent continuellement évoluer en raison d'un environnement complexe, turbulent et surtout incertain. Et cela tout au long de leur vie, tel qu'illustré dans le cycle de vie d'[ElMaraghy 2006] repris par la Figure 1.1. Ils sont ainsi dotés d'une capacité de réponse face aux changements et/ou perturbations, et leurs gestionnaires visent à en tirer le meilleur profit. Selon les auteurs, cette propriété est couramment qualifiée d'agilité, d'adaptabilité ou encore de flexibilité des systèmes de production.

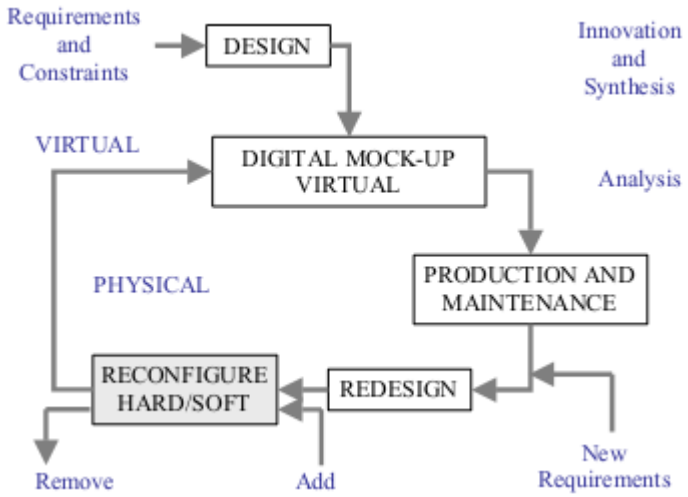

FIGURE 1.1 – Cycle de vie des systèmes de production selon [ElMaraghy 2006]

1.2.1 Définitions de base

Des comparaisons entre les concepts d'agilité et adaptabilité peuvent être trouvées dans [Katayama et Bennett 1999, McCullen *et al.* 2006]. Selon [Lee 2004, Charles 2010], ils diffèrent surtout dans la période de réponse considérée : l'agilité combine flexibilité, visibilité, réactivité, vélocité et efficacité de sorte à répondre rapidement à des incertitudes et changements à court terme, alors que l'adaptabilité désigne les changements (plus profonds) à moyen-long terme. La flexibilité, quant à elle, est présentée comme un fondement de toute organisation agile [Christopher et Towill 2000, Swafford *et al.* 2006]. Souvent définie comme l'aptitude d'un système à répondre aux changements internes ou externes efficacement et rapidement [Garavelli 2003], elle décrit le moyen qu'utilise un système pour atteindre l'agilité et l'adaptabilité, pouvant prendre plusieurs formes, et a fait l'objet de plusieurs états de l'art [Beach *et al.* 2000, Jain *et al.* 2013, Seebacher et Winkler 2013]. Par ailleurs, et toute aussi importante, la résilience mesure la capacité de revenir à un état désirable après perturbation [Christopher et Peck 2004]. Des comparaisons ont aussi été faites entre la flexibilité, l'adaptabilité et l'efficacité [Bordoloi *et al.* 1999] ou même entre l'agilité et la résilience [Charles 2010]. En ce qui concerne l'agilité, trois types de stratégie (rapide, réactive ou proactive) ont été identifiées par [Zhang 2011], qui conclue que le choix parmi elles est fait en fonction des caractéristiques du système et des types de changements qui y interviennent.

Malgré les nombreuses publications se référant au sujet, il manque un consensus sur les notions couvertes par ces concepts [Saenz de Ugarte 2009]. Certains auteurs les utilisent comme des synonymes, d'autres les différencient clairement ; ils semblent par exemple être définis de manière identique (ou presque) quand appliqués à l'entreprise au sens large, et très différemment quand ils se réfèrent à la stratégie de production [Sherehiy *et al.* 2007]. De plus, leur utilisation combinée avec d'autres principes, comme par exemple la production *lean* et agile proposée dans [Naylor *et al.* 1999], peut encore être source de

nouvelles divergences dues aux particularités liées à l'application. Toutefois, ils s'accordent sur le fait que la principale force motrice derrière ces concepts est le changement qui, dans l'ère compétitive d'aujourd'hui, a lieu à une vitesse beaucoup plus rapide que jamais auparavant [Lin *et al.* 2006]. Nous sommes par conséquent intéressés par le moyen de tirer profit de la capacité d'un système à modifier rapidement ses caractéristiques pour faire face à des circonstances changeantes.

1.2.2 Systèmes présentant des aptitudes pour changer

Les avantages de l'agilité et de l'adaptabilité sont exploités par différents types de systèmes, et ce à travers la reconfiguration, l'auto-organisation, l'auto-adaptation ou encore l'auto-évolution. Quelle que soit leur orientation, ils ont pour objectif commun de permettre aux systèmes de production de rester globalement efficaces sur une longue période de temps, même si des prévisions adéquates ne sont pas disponibles. En général, cela exige que les composants du système puissent être réutilisés dans différentes configurations, pour différentes tâches, et éventuellement sur différentes installations. Comme décrit dans [Gunasekaran 1999], cela implique quatre dimensions, à savoir :

- Stratégies : Les objectifs à long terme d'une entreprise afin de déterminer les politiques opérationnelles et commerciales appropriées ;
- Technologies : Les exigences concernant le matériel (par exemple, équipements et outils) et les technologies de l'information (ordinateurs et logiciels) ;
- Systèmes : Les considérations faites principalement à l'égard des logiciels ou systèmes d'aide à la décision (le besoin d'un système de conception rapide de produits, d'un système adaptatif de planification et de contrôle de la production et d'un système d'information reconfigurable à courte durée) ;
- Personnes : Les facteurs humains impliquant la gestion et la motivation du personnel pour favoriser les changements.

D'autres variantes existent dans [Zhang 2011, Aravind Raj *et al.* 2013] par exemple, mais on y retrouve les mêmes éléments-clés.

En termes d'ingénierie mécanique et automation, les travaux de la littérature explorent des approches par composants (citons en exemple les technologies *plug-and-play*). Selon [Koren *et al.* 1999], le contexte économique, caractérisé par une concurrence à échelle mondiale couplé à une incertitude liée aux changements sociaux et technologiques rapides, a forcé les industriels à faire face à un nouveau défi de capacité de réponse. Ce défi requiert un nouveau type de système de production dit reconfigurable et conçu afin d'être capable d'effectuer des changements rapides de sa structure ainsi que de ses composants matériels et logiciels. L'objectif est un ajustement rapide de la capacité de production et/ou de son adaptation à un nouveau produit ou une nouvelle famille de produits (fonctionnalité) en réponse à des changements soudains dans le marché ou dans la réglementation. Cette notion qui positionne les systèmes de production reconfigurables entre les systèmes dédiés

et flexibles est largement utilisée [ElMaraghy 2006, Lamotte 2006, Baqai 2010, Essafi 2010, Kanso et Berruet 2010, Bensmaine *et al.* 2013, Borisovsky *et al.* 2013]. Par ailleurs, une comparaison entre les paradigmes flexible et reconfigurable est dressée par [ElMaraghy 2006]. De plus, des systèmes dits reconfigurables et agiles émergent lorsque l'accent est mis sur leur réactivité [Chalfoun *et al.* 2012].

La mise à disposition d'une installation ayant les capacités et fonctionnalités nécessaires, ainsi que sa mise à jour au besoin, augmente le temps de vie d'un système [Koren *et al.* 1999, ElMaraghy 2006] (Figure 1.1), avec la possibilité de choisir son organisation très tard dans la phase de conception ou même dynamiquement durant son exploitation [Kanso et Berruet 2010]. Par contre, la flexibilité du système doit être limitée pour gagner en temps [Koren *et al.* 1999] ; son aspect reconfigurable doit être pris en compte dès sa conception ; et il faut accessoirement un mécanisme permettant le déclenchement de la reconfiguration, la recherche (et le choix) d'une nouvelle configuration et sa mise en place [Kanso et Berruet 2010]. La modularité des composants et leur compatibilité sont alors essentielles pour faciliter les reconfigurations (dont les possibilités sont complètement définies par la technologie). Cela est obtenu grâce à des unités ou des éléments fonctionnels standardisés (par exemple, des modules *plug-and-produce*) et des interfaces logicielles aussi standardisées. La compatibilité permet la réutilisation, par laquelle les unités du système partagent des normes communes d'interaction et d'interface et sont facilement insérées ou retirées. Un système ouvert avec une certaine redondance (ou propriété d'évolutivité) permet en plus de changer graduellement la capacité d'une manière rapide et économique. Pour plus de détails, se référer à [Koren *et al.* 1999, Tharumarajah 2003, Deif et ElMaraghy 2006, ElMaraghy 2006].

Ici toutefois, nous nous intéressons plutôt à la reconfigurabilité dans un sens large et général. En termes d'ingénierie industrielle et de gestion, pour certains systèmes, on ne cherche pas seulement à les reconfigurer, on cherche aussi à les adapter de manière plus générale. Dans cette optique, des chercheurs ont exploré différents types de systèmes capables de changer eux-mêmes leurs installations de production et leur organisation, conformément à l'évolution de facteurs internes et/ou externes. Ces changements doivent être décidés par un composant de contrôle du système de production, responsable donc du processus d'adaptation. [Fent 2001] illustre ce processus par la Figure 1.2 ci-dessous, qui montre que les entreprises au sein d'un environnement dynamique sont tenues d'observer, d'analyser, d'apprendre et de s'adapter en permanence. Il s'agit ici plutôt de déterminer quand et comment réadapter le système.

Cette idée ressort aussi au sein des systèmes reconfigurables, même s'ils sont plutôt associés au point de vue composants et interfaçages : l'utilisation en ligne des potentialités de reconfiguration amène à avoir des modules fournissant assez d'informations pour permettre de décider de changer de configuration, d'autres modules aidant au choix des éléments de cette configuration, et une ou plusieurs versions de commandes associées à une configuration [Kanso et Berruet 2010].

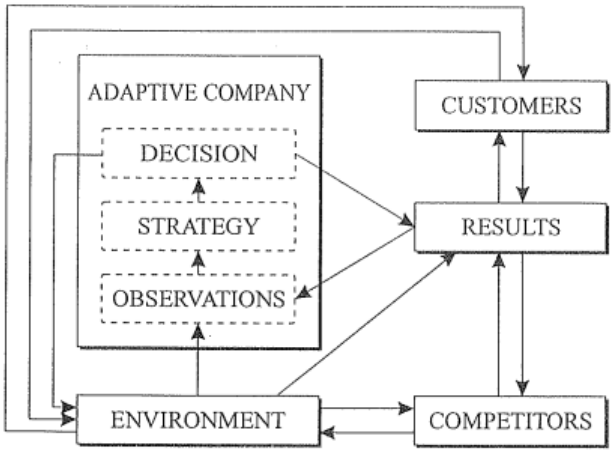

FIGURE 1.2 – Processus d'adaptation selon [Fent 2001]

Mais surtout, c'est dans ce contexte qu'émergent les différentes propriétés dites d'« auto-x » [Trentesaux 2002]. Ainsi, un système à auto-organisation par exemple est caractérisé par son aptitude à adapter dynamiquement son organisation et son comportement (par exemple, créer et/ou supprimer des processus ou des relations) à l'évolution des conditions internes et/ou externes, sans intervention extérieure, de telle sorte que les objectifs globaux soient poursuivis [Mesarović *et al.* 1980, Le Moigne 1994, Meinadier 1998, Leitão 2008, Leitão *et al.* 2012]. D'autres définitions sont aussi avancées, comme celle de [Bonabeau et Theraulaz 1994] adoptée par [Reaidy 2003], où l'auto-organisation caractérise tout processus dans lequel des structures émergent au niveau collectif, à partir de la multitude d'interactions entre individus, sans être codées explicitement au niveau individuel. D'après ces auteurs, l'objectif de l'auto-organisation est de permettre l'évolution dynamique et autonome d'un système existant, en fonction du contexte, de façon à en assurer la viabilité. Elle permet aux entités composant le système de s'adapter à leur environnement soit par spécialisation des fonctions (apprentissage), soit par modification de la topologie du groupe et des interactions correspondantes. Selon [Leitão *et al.* 2012], cela permet en plus d'atteindre plusieurs propriétés cruciales telles que l'auto-configuration (ajouter/supprimer/modifier des composants), l'auto-optimisation (s'adapter de façon proactive pour répondre aux stimulus environnementaux) et l'auto-rétablissement (diagnostiquer les écarts dus à des conditions inattendues et agir pour normaliser). L'auto-organisation correspond alors à un processus dans lequel une entreprise est en mesure de contrôler activement le cours de ses transformations internes pour maintenir sa cohérence interne, tout en conservant l'organisation des interconnexions au sein de son environnement opérationnel [Tharumarajah 2003].

D'après [Meinadier 1998], un système de production à auto-apprentissage est quant à lui capable de modifier ses caractéristiques pour mieux s'adapter à son environnement en tenant compte de son expérience antérieure. Dans ce contexte et selon l'[APICS 1998], ce concept est relativement proche de celui d'adaptabilité. Ainsi, également nommé système auto-adaptatif, il est défini dans [Trentesaux et Tahon 1995, Trentesaux 1996, Trentesaux

2002] en termes de réactivité et de flexibilité, c'est-à-dire par sa capacité à pouvoir intégrer au mieux l'aspect dynamique du processus de fabrication, donc à gérer les différents événements (aléatoires ou non) qui peuvent survenir durant le processus en fonction des expériences passées (apprentissage, mémorisation, etc.).

Citons encore le concept d'auto-évolution qui, selon [Shin *et al.* 2009, Shin *et al.* 2012], hérite des éléments fondamentaux de celui d'auto-organisation. Ces auteurs définissent un système de production auto-évolutif comme un réseau organique de ressources de production qui est capable d'adapter activement sa structure organisationnelle à l'environnement où il fonctionne. En outre, ses constituants (non seulement le système mais aussi ses moyens de production) sont capables de réguler de manière autonome leurs propres objectifs en fonction de l'environnement ; les politiques décisionnelles s'adaptent donc progressivement. Cela revient à la notion d'auto-finalisation de [Le Moigne 1994], où le système est capable de définir et de modifier ses propres finalités. À ce propos, [Trentesaux 2002] signale que l'auto-finalisation concerne l'auto-détermination des objectifs et non celle des finalités. En effet, dans le cadre du pilotage où il se place, un système de pilotage présente toujours la même finalité (celle de pilotage). Il suggère alors qu'un terme plus approprié (mais non usité) serait par exemple l'auto-objectivation.

Notons que le préfixe « auto » précise que le système réalise lui-même la tâche, ce qui requiert une forme d'intelligence et d'automatisation. Ce préfixe est utilisé à tort et par abus de langage dans certains cas où un système n'est adaptable que par une intervention extérieure.

1.2.3 Positionnement relatif des différentes aptitudes d'un système de production pour répondre aux changements

Il est possible de retrouver une certaine hiérarchie parmi les termes que nous venons d'introduire. Dans [Trentesaux 2002], la typologie des systèmes de pilotage fondée sur les niveaux de complexité des systèmes en définit trois types :

- Le pilotage élémentaire pour la recherche d'efficacité, où le système est complètement déterminé, avec absence de toute décision ;
- Le pilotage auto-organisé ou adaptatif ou à apprentissage pour la recherche d'efficacité et d'efficience, où il y a la possibilité de modification des moyens mis à disposition pour atteindre les objectifs assignés (si non adéquation) ;
- Le pilotage auto-finalisé pour la recherche d'efficacité, d'efficience et de pertinence, où il y a en plus la possibilité d'adapter les objectifs aux moyens existants.

En revanche, aucune différenciation n'est faite entre l'auto-organisation et l'auto-adaptation et, puisque l'auteur se place dans le cadre précis du pilotage, les systèmes reconfigurables ne sont pas non plus analysés.

Nous proposons ainsi d'établir une nouvelle hiérarchie qui semble plus adaptée à notre

problématique d'adaptation, bien qu'une certaine analogie puisse être faite avec la typologie évoquée pour le pilotage. Dans ce but, en nous basant sur les différentes définitions trouvées dans la littérature, les systèmes auto-organisables représentent le plus bas niveau d'aptitude de changements. Ils sont suivis par les auto-adaptatifs qui ajoutent le fait de tenir compte de leur expérience passée. Enfin, les auto-évolutifs sont auto-adaptatifs mais comprennent également une régulation d'objectifs. Le concept de système reconfigurable est alors à part, pouvant être combiné aux trois autres si l'on s'intéresse à l'aspect des composants modulaires pour une machine ou un logiciel.

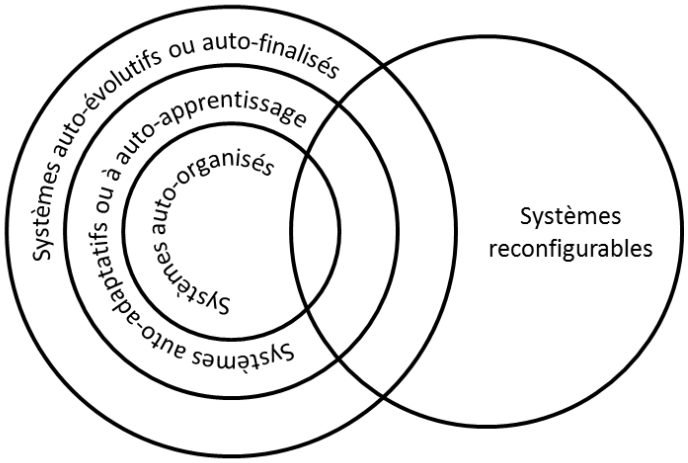

FIGURE 1.3 – Hiérarchie des termes

Après cette analyse, synthétisée par la Figure 1.3, le cas d'auto-adaptation nous semble être le plus approprié pour cette thèse. Notons que dans ce qui suit, le terme « adaptable » sera utilisé de façon la plus générale possible pour faire référence indifféremment à toutes ces notions.

1.3 Analyse de la littérature sur l'adaptation des systèmes de production

Les travaux dans la littérature sur les systèmes de production adaptables sont de plus en plus nombreux. Dans ce que suit, nous trouverons d'une part les apports plutôt conceptuels et d'autre part le plan technique avec des approches de résolution développées.

1.3.1 Modèles et architectures relatifs aux systèmes adaptables

Différentes facettes de l'adaptation des systèmes de production sont étudiées dans la littérature d'un point de vue conceptuel. Certains auteurs s'intéressent à comment déterminer si un système est adaptable, d'autres à comment les concevoir, à comment flexibiliser

les opérateurs ou encore à comment faire en sorte que les équipements industriels soient adaptés aux changements. Du point de vue automatique, un de leurs objectifs est que les moyens existants puissent être réutilisés et non pas redéployés à chaque fois. Ces travaux ne sont donc pas concurrents mais plutôt très complémentaires.

Au-delà des contextes production/adaptation, nous pouvons citer la proposition d'un modèle générique pour la tâche de configuration dans [Mittal et Frayman 1989], qui est indépendante du domaine d'application. Les auteurs cherchent à limiter la complexité de cette tâche (commune à plusieurs domaines), déterminer les connaissances nécessaires à sa réalisation et ainsi permettre l'utilisation de méthodes plus efficaces de résolution. Dans le domaine de la production plus précisément, une ontologie a été développée dans [Usman *et al.* 2013] afin d'éviter des problèmes d'interprétation de concepts. Même si elle ne couvre pas notre problématique, l'approche reste cependant intéressante (et pourrait être élargie). De même, nous pouvons trouver dans [Benkamoun *et al.* 2014] une architecture inspirée du domaine de l'ingénierie des systèmes pour représenter des systèmes de production pendant leur configuration (ou reconfiguration), et cela, à plusieurs niveaux et selon différentes dimensions/perspectives. Les auteurs mettent en avant les capacités du système pour aider à sa reconfiguration depuis la phase de conception.

Des travaux plus spécifiques ont aussi été réalisés. Par exemple, [Koren *et al.* 1999] introduisent les systèmes reconfigurables et considèrent les besoins physiques et un certain nombre de principes pour la reconfigurabilité. Des aspects sur leur conception, tels que le moment de l'envisager et le coût d'investissement, sont pris en compte dans [Koren et Shpitalni 2010]. Différents types d'approches pour concevoir des systèmes reconfigurables et/ou faciliter la reconfiguration sont identifiés dans [Benkamoun *et al.* 2013].

[Chalfoun *et al.* 2012] proposent quant à eux une description structurelle et fonctionnelle des systèmes reconfigurables et agiles, et définissent des besoins (fonctionnels ou non) liés à la reconfigurabilité. Ils utilisent pour cela le langage de modélisation de systèmes SysML [Holt et Perry 2008]. Cela aboutit à la proposition d'un modèle générique où la structure et la configuration sont détaillées dans leurs dimensions physique et logique [Chalfoun *et al.* 2013].

Un système interactif d'aide à la reconfiguration des systèmes de production est proposé dans [Mrabet *et al.* 2001] pour faire face aux insuffisances constatées en reconfiguration et plus particulièrement au niveau de son automatisation. La méthodologie se base sur la définition d'indicateurs qui correspondent à des défaillances bien particulières de façon à pouvoir les localiser et ainsi suivre le « modèle de reconfiguration » adéquat. Ce dernier détaille les procédures à appliquer et des états intermédiaires par lesquels il faut faire transiter le système en tenant compte des contraintes entre les états possibles pour chaque couple d'éléments du système. Les auteurs supposent donc que l'expertise permettant de définir les indicateurs et les modèles de reconfiguration existe.

[Caskey 2001] cherche à assister le choix de stratégies de contrôle selon la condition dans laquelle se trouve un système et le critère de performance considéré. Il se base sur

trois composants qui interagissent entre eux : l'évaluation dite ponctuelle qui donne la performance suite à une condition et des stratégies précises ; la recherche qui trouve la meilleure stratégie à mettre en place pour une condition donnée (en utilisant des plans d'expérience pour déterminer les combinaisons à tester par le module d'évaluation) ; et la généralisation qui cherche à étendre la connaissance (obtenue par le module de recherche) aux conditions possibles qui n'ont pas été évaluées.

En ce qui concerne l'auto-organisation (ou d'autres concepts proches, tels que l'adaptabilité), il est souligné dans [Bousbia et Trentesaux 2002] que peu de travaux ont été faits tant d'un point de vue conceptuel que d'un point de vue de conception. Selon les auteurs, le manque évident de concepts de base (définitions, typologies, etc.) et de modèles formels fait qu'il est encore difficile de mettre en évidence comment des mécanismes d'auto-organisation peuvent être intégrés aux systèmes de contrôle de la production ou même quels sont leurs véritables apports.

D'autres contributions ont été publiées depuis. Dans [Tharumarajah 2003] par exemple, nous trouvons une vue d'auto-organisation des entreprises manufacturières, soulignant les processus sous-jacents et les mécanismes généraux pour leur conception et fonctionnement, ainsi qu'un aperçu des problèmes structurels et opérationnels auxquels ces entreprises seront confrontées pour devenir auto-organisées. Nous y retrouvons quelques principes évoqués pour les systèmes reconfigurables, tels que la réutilisation et l'évolutivité (Section 1.2.2).

Pour aider à leur implémentation, nous pouvons trouver par exemple dans [Salerno 2009] une méthodologie et des règles de conception pour des structures organisationnelles avec un besoin élevé de reconfiguration rapide pour faire face à des situations imprévisibles. La proposition se base sur deux concepts-clés, à savoir : les événements remplaçant les tâches comme critère de répartition du travail et la communication pour coordonner ce dernier. Le but est de concevoir des groupes de travail non figés mais redéfinis selon chaque nouveau besoin. L'auteur considère que la conceptualisation n'est pas suffisante et propose des règles pour rendre possible l'emploi voire le perfectionnement de ces concepts.

Un cadre d'auto-évolution est proposé dans [Shin *et al.* 2009]. Il ne prescrit pas la transformation de la structure physique, mais traite plutôt de la structure de contrôle des moyens de production (procédure d'organisation logique) indépendamment de leur configuration matérielle. Les auteurs mettent en évidence les caractéristiques dynamiques des critères de décision qui demanderaient une adaptation des objectifs. Ceci est aussi au centre des travaux de [Topçu 2014]. L'auteur suggère la conception d'un modèle qui connecte les tâches (ou plans d'action) les plus cohérentes avec chaque objectif donné, les tâches entre elles et les objectifs entre eux. Ces connexions sont faites selon des éventuels liens (excitateurs, inhibiteurs ou encore hiérarchiques), pondérés selon leur force d'interaction (positive ou négative). Puisque les objectifs et/ou leur priorité (ou encore les différents liens établis) peuvent changer selon le contexte, un système d'apprentissage est nécessaire pour assurer la mise à jour du modèle, avec de nouveaux objectifs, des chan-

gements de priorité et/ou du poids des interactions. Cette proposition, générique et non pas dédiée aux systèmes de production, se base sur les théories de la cohérence [Thagard et Millgram 1995].

Un autre défi qui suscite l'intérêt des chercheurs (mais aussi et sûrement des industriels) est de réunir l'adaptation et l'optimisation d'un système, l'une étant la plupart du temps développée au détriment de l'autre. Ainsi, [Barbosa *et al.* 2011] proposent de s'inspirer d'une part des principes de biologie et de la vie artificielle pour établir une approche de systèmes reconfigurables combinant l'adaptation rapide et l'optimisation globale, et d'autre part de la théorie de la complexité pour garder une certaine stabilité dans le système. L'optimalité et l'adaptabilité sont aussi considérées dans [Jimenez *et al.* 2013] où une typologie des différentes structures trouvées dans la littérature est proposée. Citons encore [Cardin *et al.* 2015] qui cherchent à identifier les principales voies de recherches d'après un état de l'art du domaine.

Dans cette même optique d'adaptation, il existe aussi des travaux dans d'autres domaines et notamment concernant les systèmes logiciels : [Yeom et Park 2012] proposent une architecture auto-organisable basée sur des agents pour des composants distribués ; [Macías-Escrivá *et al.* 2013] analysent ce qui existe dans la littérature concernant les différentes dimensions de l'auto-adaptation. Bien que faites d'un autre point de vue que celui des systèmes de production, ces contributions peuvent tout de même être une source d'inspiration applicable à notre domaine.

1.3.2 Méthodes utilisées pour permettre l'adaptabilité

Malgré l'importance d'avoir un support au niveau conceptuel, l'un des principaux défis liés à la conception de ces systèmes adaptables réside bien évidemment dans le développement d'outils d'aide à la décision nécessaires pour gérer leur processus d'adaptation intrinsèque. Par cela, on entend les logiques décisionnelles qui fournissent les instructions afin d'adapter le système, à savoir, le(s) changement(s) à faire selon les différentes circonstances. Ces logiques sont en quelque sorte faites « sur mesure », car leur conception dépend fortement du système considéré et plus précisément : où et comment il est possible d'agir, quelles sont les informations disponibles et quels sont les objectifs (et hypothèses) établis. Il nous a donc semblé important d'analyser comment ceci est traité dans la littérature.

Ici encore, nous retrouverons le problème de l'adaptation traité selon différentes facettes. Certains auteurs s'intéressent à comment définir une nouvelle configuration pour le système sans forcément mettre en avant les liens avec la configuration en cours ; d'autres cherchent comment mettre en place un type spécifique de changement et d'autres encore se penchent sur le processus de décision d'une façon plus large. Bien que certains travaux soient plutôt concurrents, d'autres n'en restent pas moins là aussi complémentaires. Nous les regroupons selon le principe ou l'outil principal sur lequel ils se basent.

1.3.2.1 Optimisation

Des méthodes d'optimisation sont utilisées dans [Essafi 2010] non pas pour reconfigurer mais pour concevoir des lignes d'usinage mono-produit reconfigurables et équilibrées. Les décisions concernent le nombre de machines et de stations, la gamme opératoire à utiliser pour la fabrication du produit, l'affectation des opérations aux machines/stations (et donc les modules à installer) et leur agencement. Par contre, l'auteur signale que dans les cas où une reconfiguration doit être faite, l'écart entre la nouvelle et l'ancienne configurations doit être considéré puisque le critère à minimiser devient le changement du nombre de machines par station. Dans le cas d'une substitution de produit par exemple, il faut chercher à affecter les opérations de la nouvelle gamme opératoire aux stations déjà installées sur la ligne (et éventuellement ajouter de nouvelles stations).

Le même type de problème est résolu avec un algorithme génétique dans [Borisovsky *et al.* 2013]. [Bensmaine *et al.* 2013] utilisent aussi un algorithme génétique, mais plus précisément une approche de type NSGA-II, toujours pour une ligne mono-produit. Le but est de sélectionner des machines reconfigurables parmi un ensemble disponible de sorte à minimiser le coût et le temps total d'exécution (*makespan*). Pour le calcul de ces deux critères de performance, des aspects liés à la reconfiguration (changements d'outil, de module et/ou d'axe) sont pris en compte, mais ils sont connus à l'avance puisqu'ils sont intrinsèques à la ligne et donc définis en même temps que celle-ci est conçue. Autrement dit, la sélection faite induit les réglages qui seront nécessaires au cours de la production : elle ne détermine pas seulement le nombre et la nature de machines, elle détermine aussi l'affectation des opérations aux machines et leur ordonnancement. Un ensemble de solutions non-dominées est fourni comme résultat, ce qui offre de la flexibilité au décideur.

La résolution simultanée de deux problèmes dynamiques est prise en compte dans [Aryanezhad *et al.* 2009] : la formation de cellules de production et l'affectation des opérateurs. Selon le plan de production, une méthode d'optimisation est proposée pour décider périodiquement du nombre de machines et de leur positionnement, du nombre d'opérateurs, de leurs compétences et de leur affectation. Ceci est fait pour minimiser le coût global qui comprend l'achat et le déplacement des machines d'une part ainsi que le recrutement, le licenciement et la formation des opérateurs d'autre part.

Toujours en vue d'une optimisation, une procédure à deux niveaux est proposée dans [Chen *et al.* 2014]. Lors d'une adaptation de la ligne de produits d'une entreprise, les auteurs s'intéressent au portefeuille de produits et à leur prix mais aussi et surtout au choix simultané de ces deux facteurs ayant des conséquences au niveau de la demande du marché et des coûts. Ils utilisent pour cela le modèle de choix discrets *Mixed Logit* pour l'estimation des demandes selon les préférences des clients, la modélisation des coûts étant basée sur la méthode ABC (pour *Activity-Based Costing*) avec les coûts matériel, de main d'œuvre et des activités sous-jacentes. Ils adoptent un algorithme génétique pour la sélection du portefeuille de produits et un algorithme à évolution différentielle pour le choix des prix.

Parmi les méthodes analytiques développées, certaines se consacrent à changer le nombre de cartes dans des systèmes à flux tiré. Dans [Rees *et al.* 1987], ceci est fait au début de chaque période sur la base des prévisions de la demande et des observations du temps de passage des pièces dans l'atelier (*lead time*). Dans [Gupta et Al-Turki 1997], des prévisions sont aussi utilisées mais ici avec la notion de périodes glissantes. Les auteurs se basent sur la moyenne et l'écart-type des temps de traitement pour estimer le temps nécessaire afin de satisfaire la demande et ainsi, si besoin, le nombre de cartes supplémentaires à rajouter et quand. Ces cartes sont retirées à la fin de la période où la décision a été prise. [Guion *et al.* 2011] et [Talibi *et al.* 2013] considèrent aussi la disponibilité d'un plan de production et, à chaque période, une heuristique basée sur le stock final et les délais de réapprovisionnement des boucles kanban détecte des éventuelles ruptures de stock (date et quantité). Des cartes supplémentaires peuvent ainsi être rajoutées. Ils utilisent des plans d'expérience et la simulation pour déterminer des facteurs de contrôle ainsi que pour étudier l'impact des cartes supplémentaires sur différents critères de performance.

1.3.2.2 Simulation

Un autre type d'approche implique l'usage de la simulation. [Wu et Wysk 1989] suggèrent par exemple de simuler périodiquement le système pour prendre en compte les modifications dans son état et ainsi changer les règles de priorité utilisées lorsque c'est nécessaire. À la fin de chaque période, la meilleure combinaison de règles de priorité est choisie pour la période suivante en fonction des objectifs de production. Une variante consistant à utiliser la simulation pour le pilotage en temps réel est développée dans [Berchet 2000, Habchi et Berchet 2003] : l'idée est d'appliquer au système des actions correctives évaluées au préalable par simulation, ce qui correspond à un mécanisme de régulation du système. Des indicateurs sont alors définis, chacun contenant des inducteurs associés à sa dérive évalués selon un ordre prédéfini. Chaque inducteur contient quant à lui des plans d'action aussi évalués selon un ordre prédéfini. La simulation est ainsi utilisée pour les essayer, un par un, jusqu'à ce que la dérive soit éliminée. Le plan d'action retenu est alors appliqué sur le système réel.

Dans cette même optique, la simulation en ligne est proposée dans [Mirdamadi 2009] afin de permettre un pilotage réactif ou correctif. Elle est cette fois-ci couplée à un outil d'exécution d'atelier de type MES (pour *Manufacturing Execution System*), ce qui d'après [Castagna *et al.* 2001] en fait un puissant outil d'aide au pilotage. Le travail se concentre cependant sur la synchronisation nécessaire entre la simulation et l'atelier réel pour l'étape d'observation en temps réel, cela dans le but d'avoir une image complète et fidèle du comportement du processus opérationnel. Les étapes suivantes de projection et de correction ne sont pas mises en place, la première devant mesurer les conséquences des événements en se projetant dans le futur proche, la seconde aidant au besoin dans le choix d'une solution adaptée. La simulation en ligne est également adoptée dans [Cardin et Castagna 2009].

Des problèmes sur le long terme ont aussi fait l'objet de publications, comme dans [Pujo 2001]. L'auteur s'intéresse aux changements de site de production et plus précisément à comment effectuer un tel déménagement sans interrompre la production. Une solution de déménagement progressif entraînant la création de stocks tampon est proposée et, encore une fois, c'est la simulation qui est utilisée pour l'analyse des scénarios possibles. Pour décrire la méthode de façon complète et précise, le formalisme DEVS (pour *Discrete Event System Specification*) est ensuite utilisé dans [Pujo *et al.* 2006]. Ces travaux ne portent pas sur l'aspect décisionnel mais se concentrent sur la mise en place d'une décision supposée déjà prise.

1.3.2.3 Optimisation via simulation

[Saenz de Ugarte 2009] utilise non seulement la simulation mais aussi l'optimisation au sein d'applications MES et ERP (pour *Enterprise Resource Planning*) dans le but d'avoir un modèle intégré de planification et d'ordonnancement réactif. Ici, le processus décisionnel est déclenché par des événements imprévus qui peuvent provenir de différentes sources (machine, outil, transport, opérateur, demande, fournisseur) et sont donc détectés par l'intermédiaire de différents systèmes (gestion de cycle de vie, gestion de la chaîne logistique, ERP, MES, etc.). Vient ensuite l'analyse des données pertinentes dont l'état courant du système. Ici encore, les données de l'ERP et du MES sont nécessaires. Les décisions peuvent alors être prises puis propagées à chaque niveau de l'entreprise impliqué et/ou impacté. Cette approche est concentrée sur le court terme : l'optimisation (par un algorithme génétique) propose des solutions d'ordonnancement et la simulation est utilisée pour les évaluer.

[Zambrano Rey *et al.* 2014a] utilisent une approche d'optimisation via simulation pour définir d'une part la gamme à utiliser pour chaque produit (parmi des gammes alternatives) et d'autre part leur ordonnancement. Ils proposent une hybridation entre un algorithme génétique et un contrôleur distribué pour les dates de mise à disposition des pièces dans le système. À chaque fois qu'une perturbation est détectée, les opérations en cours sont finalisées et la solution initialement proposée est ensuite recalculée. Le calcul d'une nouvelle solution implique alors un délai pendant lequel le système reste bloqué. Les auteurs ont travaillé sur la cellule du pôle AIP-PRIMECA de l'Université de Valenciennes et du Hainaut-Cambrésis en France, cellule qui a fait l'objet d'un *benchmark* dans [Trentesaux *et al.* 2013]. Pour ce travail en particulier, [Zambrano Rey *et al.* 2014a] la considèrent dans un environnement juste-à-temps où les déviations par rapport aux dates de livraison définissent la performance.

1.3.2.4 Cartes de contrôle ou seuils

Des cartes de contrôle sont utilisées par exemple dans [Takahashi et Nakamura 1999b, Takahashi et Nakamura 2002]. Les auteurs se servent plus précisément des cartes du type

moyenne mobile pondérée exponentiellement (EWMA) pour contrôler la moyenne des temps inter-arrivées des demandes et changer le nombre de cartes dans un système à flux tiré. Pour construire leur carte, une analyse par simulation sur l'attente des clients est faite selon le nombre de cartes et le taux d'arrivée. L'objectif est d'avoir une attente moyenne au plus proche d'un niveau de référence. Ils utilisent aussi une carte de même type pour la variance et analysent le compromis entre l'attente et les encours dans [Takahashi *et al.* 2004]. Pour adapter ce type de système, [Hopp et Roof 1998] proposent également un contrôle statistique pour le taux de sortie vérifié par rapport à une valeur de référence à chaque demande livrée.

Toujours dans le domaine des systèmes à flux tiré, nous trouvons aussi plusieurs méthodes basées sur des règles simples avec des seuils déclenchant des décisions comme celle de [Tardif et Maaseidvaag 2001]. Leur but est de minimiser les coûts de stockage et de demandes en retard, la décision se basant sur le stock final. Ils utilisent une méthode de recherche locale gloutonne pour essayer d'optimiser leurs seuils et les performances sont évaluées par simulation. De nombreux travaux s'en sont inspirés : [Framinan *et al.* 2006, Shahabudeen et Sivakumar 2008, Sivakumar et Shahabudeen 2008, Takahashi *et al.* 2010, Le Pallec Marand *et al.* 2013, Pierreval *et al.* 2013, Korugan et Gupta 2014].

1.3.2.5 Théorie du contrôle

Nous pouvons encore citer la théorie du contrôle comme étant à la base de plusieurs travaux comme ceux de [Wiendahl et Breithaupt 1999] pour régler la capacité et le taux d'entrée d'un système. Ici, l'objectif est que les encours restent à un niveau souhaité et les problèmes de demandes en retard soient résolus au plus vite. Dans [Deif et ElMaraghy 2006], la modélisation et l'analyse se basent en plus sur l'application de l'analyse de rétroaction (*feedback*). Le mécanisme de régulation est conçu grâce à des méthodes d'analyse de fonctions de transfert. Des variations de la demande sont détectées par des dérives dans les encours et/ou le taux de production (par rapport à des valeurs de référence). Des ajustements sont alors faits au niveau des encours (taux d'entrée dans le système) et/ou de la capacité (nombre de machines ou composants) en tenant compte des dates de livraison. Les fonctions de transfert sont à nouveau utilisées dans [Deif et ElMaraghy 2007] où des actions correctives sont appliquées en cas de divergences dans les niveaux désirés de production, stock et encours. Les auteurs considèrent que des prévisions de demande sont disponibles, ce qui permet de sélectionner via l'analyse de régression (sur des périodes glissantes) quelle politique de planification et contrôle de la production adopter et pour combien de temps.

[Tamani *et al.* 2009] proposent quant à eux un mécanisme de contrôle flou de type Takagi-Sugeno pour réguler les flux de production. Tout en respectant leur capacité maximale, les taux de production des machines sont ainsi ajustés en fonction des niveaux de stocks, des encours et du déficit ou excédent de production. Ce contrôle implique des ajustements au niveau local où les performances sont considérées de façon myope. Il peut

aussi impliquer l'intervention d'un superviseur lorsque des déviations de la performance sont constatées par rapport à des indicateurs globaux. Deux mécanismes d'agrégation sont suggérés pour traiter l'aspect multi-objectif : soit la définition d'une action indépendante pour chaque indicateur de performance puis l'agrégation des actions, soit l'agrégation des mesures des indicateurs de performance puis la détermination d'une action relative à l'objectif agrégé. Le premier s'applique s'il n'y a pas de préférence ni d'interaction/contradictions entre les objectifs, le second dans le cas contraire. Ce travail est une extension de [Tamani *et al.* 2008].

1.3.2.6 Apprentissage

Les objectifs sont aussi le centre d'intérêt de [Shin *et al.* 2012] mais cette fois concernant leur caractère dynamique. Les auteurs proposent ainsi un mécanisme pour leur régulation : un système d'inférence floue basé sur un réseau de neurones mis à jour par un signal de renforcement de l'environnement. Deux réseaux de neurones sont en fait utilisés. Le premier (de régulation d'objectif) encapsule les règles floues et est mis à jour en permanence par un signal de renforcement interne. Ce signal est délivré par le second réseau (d'évaluation) qui calcule la compatibilité entre les objectifs et la situation actuelle de l'environnement. Ils appliquent ce mécanisme à un problème de planification de production dans le but d'organiser dynamiquement des ressources, l'allocation des commandes étant faite selon les objectifs individuels des ressources.

L'apprentissage par renforcement est aussi adopté dans [Aissani *et al.* 2008] pour le pilotage en temps réel, et plus précisément pour l'allocation des tâches aux machines. Des agents assurent une amélioration continue de la performance globale du système en apprenant (par des récompenses) le meilleur comportement à adopter dans leurs divers rôles (l'auto-organisation, la planification des tâches, etc.). Nous avons ici un problème relevant plus de la planification dynamique que de l'adaptation. D'autres aspects tels que le multi-objectif sont ensuite intégrés dans [Aissani 2010] alors qu'[Aissani *et al.* 2012] adaptent et étendent l'approche au contexte des entreprises multi-sites.

Un autre type d'approche d'apprentissage sans base d'exemples est proposé pour le pilotage en temps réel dans [Mouelhi-Chibani 2009, Mouelhi-Chibani et Pierreval 2010]. Le but est de sélectionner dynamiquement la règle de priorité à utiliser pour l'ordonnancement des pièces (parmi des règles existantes). Pour cela, un couplage est fait entre la simulation, l'optimisation et l'apprentissage par réseaux de neurones. Les choix sont faits selon le nombre de pièces (et leur type) dans chaque file d'attente. De plus, les valeurs minimale, moyenne et maximale sont utilisées pour les temps opératoires, les marges libres (*slack time*) et les dates de livraison (*due date*).

Dans [Metan *et al.* 2010], nous avons une approche qui combine simulation, fouille de données et cartes de contrôle pour sélectionner périodiquement des règles de priorité pour le pilotage. Une carte de contrôle est utilisée pour suivre la performance de l'arbre de

décision. Ce dernier est mis à jour si nécessaire par un nouveau processus d'apprentissage intégrant les données plus récentes. La carte de contrôle peut aussi passer par un processus de mise à jour si elle ne s'avère plus assez performante. Par exemple, cela peut se produire si deux mises à jour successives de l'arbre de décision ne suffisent pas à remettre le processus en équilibre. Ce travail est une extension de [Metan et Sabuncuoglu 2005].

Des modèles de simulation ont aussi été utilisés plus tôt pour évaluer les performances des règles de priorité à partir des différents états d'un atelier. Les données simulées servent ainsi à générer un ensemble d'apprentissage. Dans [Pierreval 1992b], un réseau de neurones est entraîné à prédire la combinaison adéquate de règles à utiliser selon l'état du système et le(s) critère(s) de performance à optimiser. À noter qu'ici, chaque station de travail peut utiliser une règle différente. Les combinaisons considérées sont en fait classées de façon pondérée, ce qui permet d'évaluer la « certitude » du résultat et laisse ainsi le choix final au décideur. Dans [Caskey 2001] et également pour un réseau de neurones, l'ensemble d'apprentissage est construit à l'aide d'un algorithme génétique qui retourne, pour certains états du système, la meilleure règle à appliquer à chacune des machines d'un atelier. Dans [Piramuthu *et al.* 1993], l'apprentissage par induction est utilisé pour la reconnaissance de *patterns*. C'est alors un problème de classification où des règles conditionnelles (*if-else*) sont ensuite utilisées pour choisir parmi des règles de priorité existantes. Ces types d'approche sont aussi appliqués pour changer le nombre de cartes dans les systèmes à flux tiré. Les décisions sont prises soit périodiquement où des prévisions sont considérées comme disponibles [Wray *et al.* 1997, Markham *et al.* 1998], soit lorsque des variations dans la demande sont détectées [Takahashi et Nakamura 1999a].

Dans le cas de sous-utilisation de la capacité du système, [Yildirim *et al.* 2006] utilisent des réseaux de neurones couplés à la simulation. Leur but est d'ajuster : le nombre de machines actives dans chaque station de travail, la règle de priorité pour l'ordonnancement et le coefficient utilisé pour le calcul des dates de livraison. En fait, ce sont des réseaux parallèles : un réseau est construit pour chaque paire <règle, coefficient>. Pour P règles, K coefficients et L nombres de machines dans chacune des M stations de travail, cela nous donne $R = P \cdot K$ réseaux. Leur structure est la suivante : une couche d'entrée avec les performances souhaitées par le gestionnaire, une couche cachée et une couche de sortie avec le nombre de machines dans chaque station de travail. Les R solutions proposées (une par réseau) sont simulées pour faire le choix final, ce qui est toujours moins coûteux que de simuler L^M alternatives pour chaque combinaison <règle, coefficient>.

Des réseaux de neurones et la simulation sont à nouveau utilisés dans [Araz et Salum 2010], avec en plus un système d'inférence floue. Ici, l'affectation des opérateurs et les règles de priorité sont redéfinies à chaque intervalle fixe de temps ou suite à des seuils franchis par les mesures de performance. Les critères de performances sont liés au retard, au temps de passage des pièces dans l'atelier (*flow time*), au temps d'attente dans le système et aux encours. Les données utilisées comprennent les taux d'utilisation des machines et des opérateurs, le nombre moyen de pièces dans les files d'attente, le temps moyen de transfert des opérateurs et les performances courantes. À cela se rajoutent trois

paramètres, à savoir : le taux d'arrivée, le nombre de types de pièce dans le système et le coefficient utilisé pour le calcul des dates de livraison. La simulation est utilisée pour générer des ensembles d'apprentissage et de validation pour le réseau de neurones. Ces ensembles sont constitués des différentes combinaisons de variables de décision et d'états du système. Le réseau de neurones est quant à lui utilisé pour estimer les mesures de performance, donc en tant que méta-modèle de simulation. Enfin, pour une évaluation multicritère, le système d'inférence floue agrège les valeurs de performance pour fournir une performance globale. Celle-ci permet de déterminer quelle combinaison de variables de décision utiliser pour la prochaine période de production ou jusqu'au prochain événement déclencheur.

1.3.2.7 Autres

D'autres approches sont inspirées non seulement de la biologie (comme du comportement de neurones trouvé ci-dessus), mais aussi des lois physiques. C'est dans cette optique que [Leitão *et al.* 2012] utilisent une structure distribuée modélisée par des systèmes multi-agents implémentant un mécanisme basé sur des champs potentiels. Ce mécanisme est aussi mis en place dans [Pach *et al.* 2012]. Dans les deux travaux, l'application concerne l'allocation dynamique de tâches et le routage dynamique de palettes, l'objectif étant de minimiser le temps total d'exécution (*makespan*). Chaque ressource émet des champs attractifs selon ses fonctionnalités (les services offerts) et sa disponibilité, leur intensité se réduisant avec la distance (ou le temps de transport). Cela permet à un produit de devenir en quelque sorte actif : il décide dynamiquement de son affectation à la ressource qui émet le plus grand champ potentiel. Cette affectation ne pourra changer qu'à certains points précis de l'acheminement du produit, points dits de divergence puisqu'ils matérialisent la possibilité de suivre différents chemins si les champs ont changé entre temps. À chaque réaffectation, l'itinéraire est changé en conséquence. L'état des files d'attente et la date de libération des ressources sont utilisés comme information additionnelle dans [Pach *et al.* 2012] pour mesurer leur attractivité. Les deux travaux ont été implémentés dans la cellule du pôle AIP-PRIMECA mentionnée précédemment (Section 1.3.2.3).

Cette même cellule, en versions réelle et virtuelle (adaptée pour permettre des changements de la structure physique), a encore été reprise dans [Barbosa *et al.* 2015] pour évaluer une nouvelle approche. Les auteurs présentent un mécanisme d'auto-organisation bidimensionnel prenant en compte des composants comportemental (niveau micro) et structurel (niveau macro) pour permettre de réagir de manière souple ou drastique. En plus de l'allocation des produits aux machines remise en cause dans un exemple lors de pannes, nous trouvons l'agencement des machines modifié dans un deuxième exemple suite à l'arrivée d'un nouvel ordre de fabrication. Chaque élément du système, modélisé par un agent, surveille son état et son environnement de façon à détecter des déclencheurs et des opportunités pour évoluer. Deux types de données sont utilisées : locales plus précises et à jour ; globales pour une solution plus stable à long terme et pour moins de myopie.

Des mécanismes de renforcement règlent progressivement le compromis entre leur usage. L'élément affecté auto-organise son comportement par des règles et mécanismes internes. Pour répondre à des changements plus profonds, il auto-organise aussi ses relations avec d'autres éléments et/ou leur organisation, ce qui peut être fait logiquement ou physiquement. L'idée est de surveiller constamment l'état du système et d'en extraire des faits devant être interprétés par une base de règles. Si cela abouti à des propositions de changement, celles-ci doivent être validées par un stabilisateur de nervosité (inspiré de la théorie du contrôle) qui cherche à éviter un comportement chaotique. Un processus d'apprentissage permet de générer, de supprimer et d'adapter les connaissances qui constituent la base de règles. Il comprend un apprentissage social pour propager l'information par un mécanisme de type phéromone et un apprentissage par renforcement pour évaluer les décisions passées. Ces dernières ont un impact positif ou négatif dans les choix futurs du même ordre. Leur évaluation est faite sur la base de l'état du système lors de la prise de ladite décision, du choix fait et de la performance atteinte. Le critère pris en compte dans les exemples est ici encore la minimisation du temps total d'exécution.

Pour le pilotage et plus précisément l'allocation dynamique de ressources, [Reaidy 2003] utilise aussi des systèmes multi-agents. Le mécanisme est cette fois-ci basé sur des protocoles de négociation et sur la théorie des jeux. En utilisant des informations telles que les tâches prioritaires et les temps de traitement sur des différentes machines, chaque produit choisit de façon consensuelle une ressource uniquement pour sa prochaine tâche, et ce lors de la fin de la tâche courante. Dans [Borangiu *et al.* 2010], c'est une structure holonique basée sur la coopération et la négociation qui gère l'allocation des produits aux machines. Ils utilisent des informations telles que la capacité, la fonctionnalité et la disponibilité des ressources, les temps de transport et de traitement ou encore la priorité des produits.

Nous pouvons encore trouver dans [Baqai 2010] l'analyse de la co-conception d'un processus d'usinage et de la configuration d'un système reconfigurable en tenant compte des contraintes technologiques et des interactions entre le processus et les ressources. Un nouveau produit à faire entraîne une analyse des gammes possibles (et des structures respectives). Sera ensuite choisie celle qui minimise la reconfiguration et ainsi son coût (nombre de machines, positionnement, configuration, affectation des opérations aux machines). L'auteur s'intéresse alors aux changements de la structure physique et des modules utilisés par les machines. Pour ce faire, il utilise une approche de conception axiomatique avec un domaine de performances intégré pour le lien entre les besoins fonctionnels et les éléments physiques. D'autre part, il utilise une approche Fonction-Comportement-Structure (ou FBS pour *Function-Behavior-Structure*) adaptée pour formaliser et évaluer les solutions. Pour ce qui est de la co-conception, [Aldanondo *et al.* 2008, Vareilles *et al.* 2008, Pitiot *et al.* 2013, Pitiot *et al.* 2014] proposent l'utilisation de modèles basés sur des contraintes pour concevoir progressivement un produit et la planification de son processus de production.

1.4 Conclusion

Dans ce chapitre, nous avons présenté le contexte général d'adaptation dans lequel s'inscrit ce travail de thèse avec les différents concepts et paradigmes qui l'entourent. Pour une meilleure analyse, les contributions de la littérature ont été découpées en deux grandes dimensions, l'une plutôt conceptuelle et l'autre orientée vers la pratique.

Malgré l'intérêt croissant des chercheurs pour les problèmes liés à l'adaptation, le vocabulaire reste confus si l'on s'intéresse à comment caractériser les systèmes qui visent à intégrer les aptitudes nécessaires. Ceci peut être expliqué par l'absence d'une spécification claire, unifiée et formelle des éléments et processus impliqués dans l'adaptation de ces systèmes. Les auteurs ne s'accordent pas toujours sur les termes utilisés. De plus, nous avons constaté un manque d'uniformité dans la façon dont les contributions techniques sont présentées : toutes ne fournissent pas les mêmes types d'informations alors qu'elles en négligent parfois certaines qui pourraient s'avérer importantes. Ces aspects n'ont fait que mettre en évidence la nécessité d'un cadre conceptuel pour aider à mieux comprendre et exploiter le domaine. C'est pourquoi le chapitre suivant sera consacré à notre proposition, publiée dans [Belisário et Pierreval 2013].

Sur le plan pratique, nous pouvons dresser diverses constatations à partir des approches existantes qui témoignent du fort besoin d'aide à la décision pour l'adaptation des systèmes de production. Tout d'abord, ces approches se concentrent très souvent sur le pilotage à très court terme. Par ailleurs, plusieurs travaux considèrent que des prévisions et/ou plans de productions sont disponibles, ce qui permet la décomposition du problème en périodes et facilite l'usage de méthodes d'optimisation. Cela peut être à l'origine de plusieurs sous-classes de problèmes au sein de la même problématique. La simulation a aussi largement été utilisée, ce qui n'est pas sans lien avec sa capacité à représenter la dynamique des systèmes de production ainsi qu'à prendre en compte leur complexité. Cependant, tout le potentiel de la simulation n'est pas exploité. De plus, nous pouvons remarquer que ces dernières années, les méthodes d'apprentissage issues de l'intelligence artificielle regagnent de l'intérêt. C'est pourquoi nous nous intéresserons à ces deux points dans le Chapitre 3.

Chapitre 2

Proposition d'un Cadre Conceptuel pour l'Adaptation des Systèmes de Production

2.1 Introduction

Nous avons vu dans le chapitre précédent que les systèmes de production visant à s'adapter à un environnement changeant sont de plus en plus présents dans la littérature. Bien que de nombreux chercheurs aient publié sur le sujet, ils ont souvent mis l'accent sur le développement de solutions ou de considérations opérationnelles pour des problèmes spécifiques d'adaptation. Au niveau conceptuel, des paradigmes importants ont été introduits. Citons l'identification de facilitateurs à l'adaptation et de quelques caractéristiques structurelles et opérationnelles que les systèmes adaptables devraient intégrer.

Cependant, ces systèmes tout comme leur processus d'adaptation restent jusque-là insuffisamment définis. Les nombreux travaux sur le sujet s'avèrent utiles pour la compréhension des problèmes d'adaptation sans pour autant fournir un ensemble unificateur et générique des piliers nécessaires pour parvenir à trouver une solution à ces problèmes. Autrement dit, la littérature manque encore d'une caractérisation générique permettant non seulement de mieux communiquer, spécifier et classer les problèmes dans ce domaine, mais aussi d'aider à identifier d'éventuelles directions de recherches. Tous ces facteurs étant considérés et renforcés par le manque de certaines informations dans de nombreuses contributions techniques, nous proposons un cadre conceptuel pour combler cette absence de généricité [Belisário et Pierreval 2013]. Afin de montrer qu'il recouvre bien la littérature du domaine, il sera confronté avec celle-ci.

Ce chapitre s'organise comme suit. Nous présenterons tout d'abord l'orientation globale de notre cadre conceptuel, puis nous mettrons l'accent sur chacun des composants qu'il regroupe. Nous en profiterons pour énoncer un certain nombre de considérations basées sur ou extrapolant la littérature. Elles porteront essentiellement sur les difficultés et/ou précautions concernant ces composants. Cette démarche aidera à mieux comprendre la problématique d'adaptation qui nous intéresse.

2.2 Principes

Nous nous intéressons à fournir une définition structurée et unifiée des éléments de base et des processus impliqués dans l'adaptation des systèmes de production. Dans le domaine de la simulation, le formalisme DEVS pour la spécification des systèmes à événements discrets [Zeigler 1976] est bien connu, suffisamment générique et donc largement utilisé pour spécifier une grande variété de modèles. De même, une description conceptuelle des modèles est proposée dans [Bel et Dubois 1985]. Ces approches de type systémique ont partiellement inspiré le cadre conceptuel que nous développerons par la suite.

Nous cherchons à décrire les systèmes de production du point de vue de leur adaptabilité, soit « la capacité d'un *système de contrôle* de *modifier ses propres paramètres* en *réponse à un changement mesuré dans les conditions opératoires* » [APICS 1998].

Notre approche peut ainsi se résumer autour de trois questions-clés :

- *Comment décider de l'évolution ?*
- *Qu'est-ce qui peut évoluer ?*
- *Quand évoluer ?*

Dans ce qui suit, nous considérons que le système de production est contrôlé à l'aide d'un composant responsable de la gestion du processus d'adaptation (donc, le *système de contrôle*). Il est ici nommé « superviseur » et doit être défini par sa structure organisationnelle, son mécanisme de changement et ses objectifs.

La Figure 2.1 illustre alors le lien entre le superviseur (processus décisionnel) et le système de production. En mettant en avant les éléments essentiels à l'adaptation, elle schématise l'approche adoptée pour la définition de notre cadre conceptuel. À travers les informations de l'environnement et les indicateurs parvenus de la production, le superviseur décide quant au moment et à la nature d'une adaptation pour atteindre les objectifs fixés. Sa décision est transmise au système de production pour une mise en place (qui doit bien souvent être validée au préalable par un gestionnaire) et peut être évaluée à l'aide d'indicateurs. C'est un processus permanent, basé sur des plans de production et/ou sur l'évolution des indicateurs, qui vise l'adaptation continue. Le processus recommence à chaque nouveau besoin de changement identifié. D'autres interactions entre l'environnement, l'entreprise et le système de production (liées à leur influence mutuelle) ne sont volontairement pas représentées.

Sur la base de cette analyse globale, nous suggérons que les systèmes adaptables introduits précédemment (Section 1.2.2) soient décrits par un 7-uplet $\langle O, C, I, T, S, M, G \rangle$, où :

- O : Objet(s) modifiable(s) ;
- C : Type(s) de changement ;
- I : Information(s) utilisée(s) ;

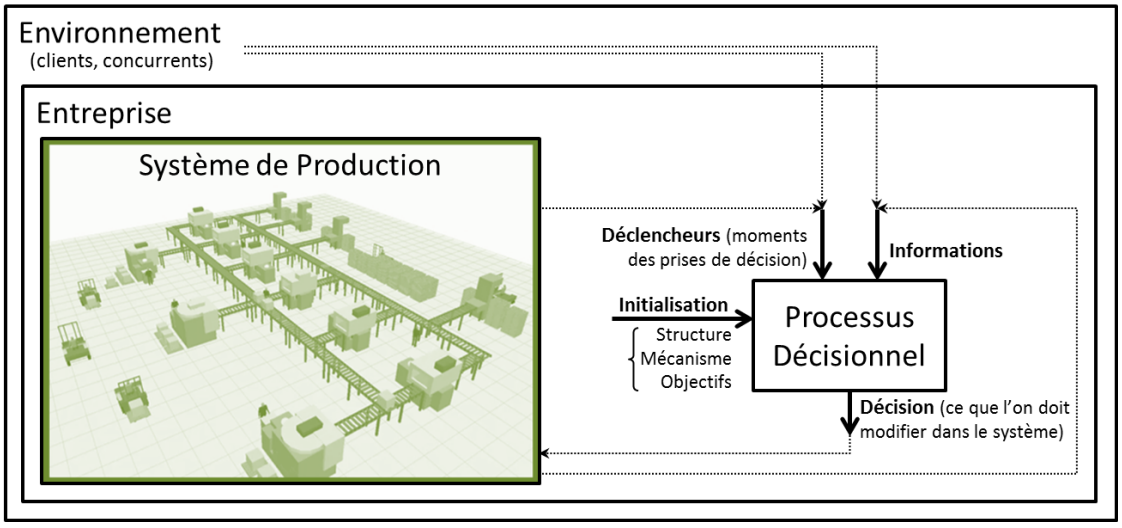

FIGURE 2.1 – Interactions entre les différents éléments nécessaires à l'adaptation continue d'un système de production

- T : Stratégie de déclenchement ;
- S : Structure organisationnelle du superviseur ;
- M : Mécanisme de changement utilisé par le superviseur (ou principes pour l'obtenir) ;
- G : Objectif(s).

Dans les sous-sections suivantes, nous nous concentrerons sur chacun de ces composants, introduits en lien avec la question à laquelle ils répondent. Des références importantes illustreront la façon dont ils sont traités dans la littérature. D'autre part, ces références nous permettront de voir la capacité du cadre conceptuel proposé (et du formalisme qui en découle) à catégoriser les approches publiées.

2.3 Composants du cadre conceptuel

2.3.1 Qu'est-ce qui peut évoluer ?

2.3.1.1 Objets modifiables

Dans un système de production et dans le but de mieux répondre à une situation donnée, de nombreuses décisions sont à prendre concernant ses multiples composants qui peuvent être adaptés. Il nous faut donc tout d'abord identifier ces composants qui permettent des changements, par exemple grâce à leur modularité. Pour cela, le système ne doit pas être considéré dans sa globalité : il doit être décomposé.

Nous définissons l'ensemble $O = \{o_j \mid j = 1..J\}$ des objets modifiables du système

qui permettent des modifications dans leur structure ou dans leurs relations avec d'autres objets (la précision « modifiable » pour un objet sera la plupart du temps omise par la suite). Ce ne sont pas nécessairement des objets physiques (comme le sont par exemple les machines, les opérateurs et les produits) : ils peuvent aussi être logiques (comme les règles de priorité, la gestion des stocks ou les logiciels). Toutefois et indépendamment de leur nature, ils sont caractérisés par leurs attributs qui définissent leur état courant (par exemple, leur emplacement), état qui peut changer avec le temps. Chaque objet $o_j \in O$ est alors caractérisé par un vecteur $A_j = \{a_{j,k} \mid k = 1..K_j\}$ qui représente ses K_j attributs ; et chaque attribut $a_{j,k}$ prend sa valeur dans un ensemble $E_{j,k}$.

Ces attributs peuvent être fixes ou variables. Ce sont ceux intrinsèquement variables (et contrôlables) qui permettent les changements et qui sont donc remis en question dans le cadre d'une adaptation. Puisque le but ici est justement de les exploiter, chaque vecteur non-nul A_j associé à un objet o_j contient uniquement ses attributs dits variables. K_j fait ainsi référence au nombre d'attributs variables d'un objet (la précision « variable » pour un attribut sera elle aussi la plupart du temps omise par la suite). Par conséquent, chaque ensemble $E_{j,k}$ contient au moins deux éléments comme valeurs possibles pour un attribut. Dans le cas contraire, on n'aurait pas la possibilité de modifier l'attribut $a_{j,k}$ qui est associé à cet ensemble.

Notons que le vecteur d'attributs A_j dépend totalement de l'objet o_j qu'il décrit, ces deux éléments ne pouvant pas être dissociés.

La combinaison des K_j ensembles pour un objet o_j donné définit toutes les possibilités de configuration liées à cet objet. Ainsi, la configuration d'un objet à un instant donné t est définie par les valeurs de tous ses attributs à cet instant $(a_{j,k,t}, \forall j, k)$. De même, la configuration globale du système est caractérisée par la configuration de tous ses objets. Elle prend sa valeur dans un ensemble Ω qui représente toutes les configurations possibles du système. Notons que, dans un problème de reconfiguration/adaptation typique, l'ensemble O des objets est fixe, tandis que dans une approche plus globale d'évolution, de nouveaux objets peuvent être créés et/ou supprimés dynamiquement.

Le diagramme de classe de la Figure 2.2 résume cette description. Pour plus de détails sur le formalisme UML, se référer par exemple à [Muller et Gaertner 1997].

À titre illustratif, si un système compte X machines et Y opérateurs, nous pouvons avoir les objets o_j où j représente les machines de 1 à X et les opérateurs de $X+1$ à $X+Y$. Rappelons que J représente le nombre total d'objets, donc ici $J = X + Y$. Considérons deux attributs pour chaque machine : la position $a_{j,1}$ et le taux de production $a_{j,2}$. Nous avons donc le nombre d'attributs variables $K_j = 2 \mid j = 1..X$. Pour chaque opérateur, considérons un seul attribut : l'affectation $a_{j,1}$. Ici nous avons $K_j = 1 \mid j = X+1..X+Y$. Enfin, l'ensemble $E_{j,1}$ pour un opérateur pourrait par exemple contenir les machines pour lesquelles il est habilité.

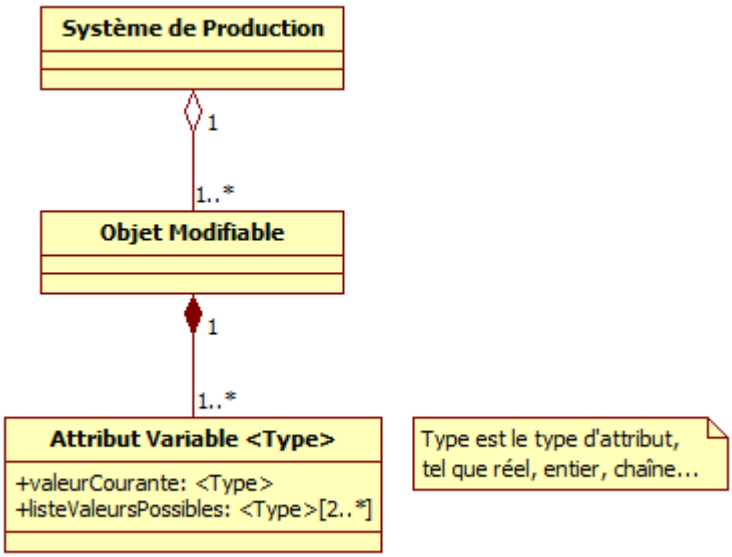

FIGURE 2.2 – Diagramme de classe modélisant les Objets modifiables d'un système de production

2.3.1.2 Types de changement

Une telle caractérisation instantanée (configuration courante) n'est cependant pas suffisante sans que l'on identifie comment la modifier pour évoluer dans Ω. Chaque objet identifié peut être modifié par des changements de ses attributs variables, ce qui nous mène aux types possibles de changement qui peuvent être mis en place. Ces changements appliqués aux attributs sont caractérisés par leur type et appartiennent à un ensemble $C = \{c_l \mid l = 1..L\}$ tel que : {déplacement, réaffectation, etc.}. Il faut alors définir chaque type de changement c_l en fonction de chaque attribut variable $a_{j,k}$ de chaque objet modifiable o_j. Ainsi, un changement de type *réaffectation* aura une incidence sur l'attribut correspondant à l'*affectation* de l'objet *opérateur*, c'est-à-dire sur l'ensemble actuel de tâches qu'il réalise. Les types de changement C peuvent affecter des quantités, des emplacements, des capacités, des compétences ou l'utilisation de modules/outils, entre autres. Certains de ces éléments sont identifiés dans la Figure 2.3 proposée par [ElMaraghy 2006]. Encore une fois, dans les systèmes dits évolutifs, des objets peuvent être créés ou supprimés par des types spécifiques de changement ; citons en exemple l'achat, la location ou la vente pour une machine, le recrutement, la sous-traitance, le licenciement ou la fin de contrat pour un employé.

Chaque c_l est une fonction modifiant la valeur actuelle $a_{j,k,t}$ d'un attribut en une autre valeur possible, tout en respectant les contraintes existantes. Par exemple, un déplacement change la position d'une machine tout en conservant des relations de précédence. De même, on ne peut pas réaffecter un opérateur à des tâches dont il n'est pas autorisé ni/ou habilité à effectuer : son affectation peut uniquement prendre une valeur dans un ensemble

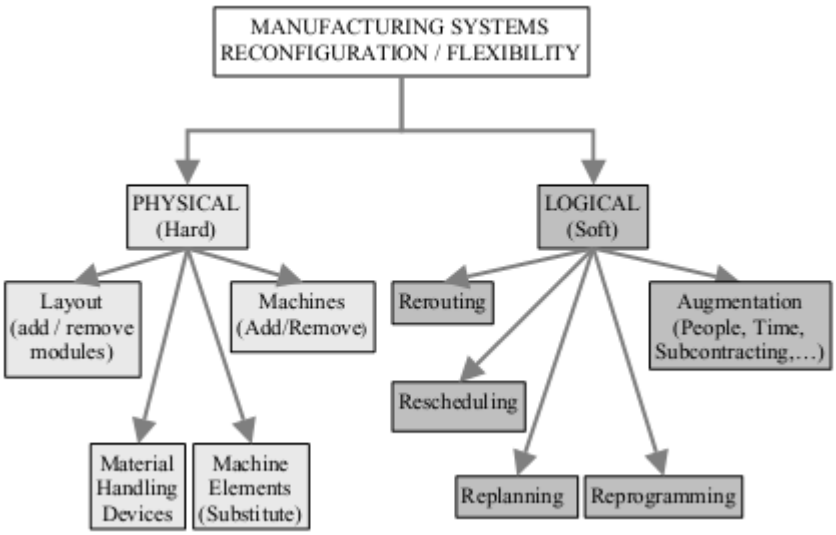

FIGURE 2.3 – Reconfiguration des systèmes de production selon [ElMaraghy 2006]

de tâches possibles selon ses compétences (et selon les ordres provenant de la hiérarchie).

Signalons une autre caractéristique importante liée aux différents types de changements : qu'ils concernent un seul objet ou plusieurs, ils peuvent être interdépendants. Par conséquent, les modifications doivent être réalisées dans une séquence cohérente, certaines séquences de changements pouvant même être imposées lorsqu'un changement en appelle un autre. Par exemple, la formation d'un opérateur permet de l'affecter à des tâches sur une nouvelle machine, ce qui implique des changements dans différents attributs d'un même objet avec un ordre cohérent. Cette formation ne serait cependant pas très pertinente si, en même temps, la machine correspondante est déplacée à un autre endroit non accessible par cet opérateur, ce qui correspondrait à des changements dans les attributs de différents objets mais cette fois-ci dans un ordre incohérent. Un tel changement dans la position d'une machine nécessiterait également des changements dans le circuit que les pièces utilisent pour accéder aux machines, une séquence de changements serait donc imposée. Un exemple concret d'interdépendance est adressé dans [Leitão *et al.* 2012, Pach *et al.* 2012] où les pièces sont réaffectées à des machines, impliquant la modification de leur acheminement en conséquence.

Ces interdépendances peuvent également être à l'origine de certaines contraintes. Citons le cas où deux opérateurs ont les mêmes compétences. Puisque de nouvelles affectations pourraient se chevaucher, la modification d'un même attribut (*affectation*) sur deux objets (*opérateurs*) différents doit être faite de façon à ce que la valeur attribuée au premier ne puisse pas l'être au deuxième, ce qui requiert une séquence cohérente de changements. À ce propos, [Mrabet *et al.* 2001] ont mis en avant la nécessité d'une compatibilité entre les états des différentes entités d'un système. De même, [Mittal et Frayman 1989] définissent la configuration (et par conséquent la reconfiguration) comme un type spécial d'activité de conception ayant une caractéristique-clé : ce qui est conçu est en fait assemblé à partir d'un ensemble de composants prédéfinis qui ne peuvent être reliés entre

eux que de certaines façons. Ces aspects sont alors à analyser avec attention mais une telle analyse sort de l'étendue de notre cadre conceptuel.

À titre illustratif, nous pouvons citer [Aldanondo *et al.* 2008, Vareilles *et al.* 2008]. Pour traiter les éventuelles contraintes et/ou interdépendances, les auteurs se sont basés sur le problème de satisfaction de contraintes (ou CSP pour *Constraint Satisfaction Problem*) [Montanari 1974, Tsang 1993] et plus précisément sur le concept des contraintes dites d'activation/inhibition et de compatibilité/incompatibilité [Mittal et Falkenheiner 1990]. Ils ont appliqué ces concepts pour coupler la configuration d'un produit à la planification de son processus de production, une phase d'optimisation ayant été ajoutée dans [Pitiot *et al.* 2013, Pitiot *et al.* 2014]. Le même type d'idée est aussi à la base des travaux de [Topçu 2014] décrits dans la chapitre précédent.

Enfin, soulignons la définition des horizons temporels auxquels on associe une décision. Nous avons d'une part l'horizon décisionnel $H_{D,l}$ qui correspond à l'un des niveaux stratégique, tactique ou opérationnel (soit au long, moyen ou court termes) [Lemaître 1981]. D'autre part, $H_{min,l}$ indique le temps pendant lequel une décision reste valable : il s'agit de son temps minimal d'application. $H_{D,l}$ et $H_{min,l}$ sont tous les deux dépendants de la décision prise, donc du type de changement à mettre en place. Ils font donc partie de sa caractérisation, c'est pourquoi ils sont indexés par l. De plus, $H_{min,l}$ peut parfois être défini en fonction de $H_{D,l}$. À cela s'ajoute encore le temps de mise en place d'un changement $H_{durée,l}$ qui n'est pas forcément négligeable et qui peut dépendre de plusieurs facteurs. Ainsi, $H_{D,l}$, $H_{min,l}$ et $H_{durée,l}$ peuvent être également indexés par j (indice pour les objets) mais nous n'irons pas jusqu'à ce niveau de détail dans cette thèse. À titre d'exemple, le temps de déplacement d'une machine dans [Barbosa *et al.* 2015] dépend de la distance à parcourir mais peut aussi varier selon la taille, le poids et la complexité des connexions (câbles pour installation/désinstallation) de la machine concernée. Souvent lié à des aspects techniques, $H_{durée,l}$ peut également influencer la définition de $H_{min,l}$ (Figure 2.4) et est souvent à l'origine de $H_{D,l}$. Le diagramme de classe de la Figure 2.5 illustre l'ensemble de cette description.

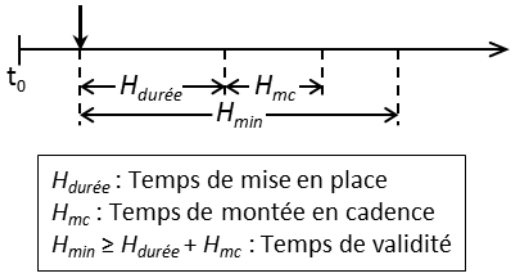

FIGURE 2.4 – Conséquences d'une décision dans le temps

En ce qui concerne $H_{min,l}$, même si sa valeur peut parfois être négligeable, il arrive qu'elle soit nécessaire pour certaines décisions. C'est par exemple le cas lors de la réaffectation d'opérateurs ou du déplacement de machines car il va de soi que l'on ne peut pas les remettre en cause à chaque instant. De plus, un temps est parfois nécessaire pour

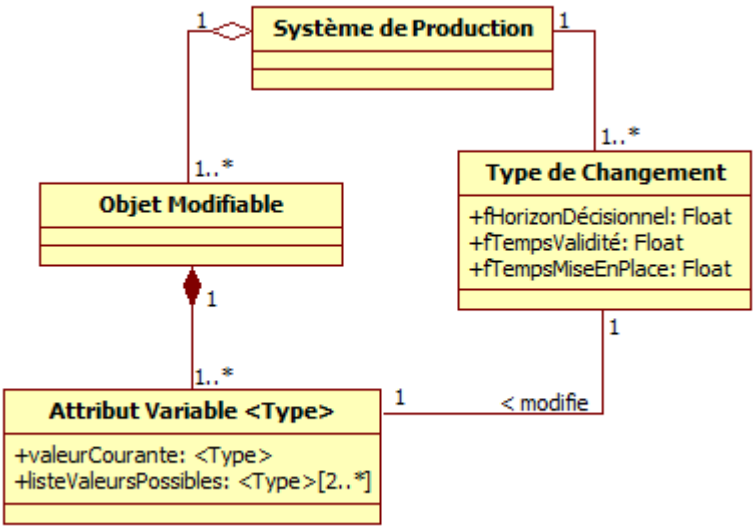

FIGURE 2.5 – Diagramme de classe modélisant les Types de changement pouvant affecter un système de production

qu'une décision prenne effet. Quelques considérations sont donc importantes à propos de la définition de $H_{min,l}$. Afin d'être très sensible (et donc réactif) aux changements, cet horizon peut être très court, mais cela peut induire des adaptations excessives. Ce phénomène peut être non souhaitable car, en plus des coûts induits, il peut conduire à une forte instabilité du système. On parle ici de nervosité du système, cette dernière faisant généralement référence à la fréquence des changements qui y sont apportés [Barbosa *et al.* 2012]. Quand un système réagit trop impulsivement et rapidement aux changements et/ou perturbations, il est dit trop nerveux [Hadeli *et al.* 2006]. Il faut cependant ajouter que, même si une logique induisant trop de changements ne serait pas acceptable d'un point de vue industriel en raison des coûts additionnels (pas forcément mesurables), la nervosité est souvent négligée lors de la conception et de la gestion des systèmes adaptables. Puisqu'adapter trop souvent le système peut être très coûteux pour sa performance, une période transitoire bloquant tout réajustement après chaque changement peut être une solution simple en ce qui concerne le contrôle de la nervosité, $H_{min,l}$ pouvant être utilisé dans ce but. Cela éviterait l'enchaînement de réajustements où le prix à payer serait l'incapacité du système à répondre à des changements soudains et importants s'enchaînant trop vite. D'autres approches typiques pour calmer la nervosité sont de restreindre le nombre de changements dans une fenêtre de temps ou alors de permettre leur application seulement à des intervalles prédéfinis [Barbosa *et al.* 2012]. Un mécanisme plus complexe pour sa stabilisation et basé sur la théorie du contrôle est proposé par [Barbosa *et al.* 2015]. Tous ces facteurs considérés, les effets et le contrôle de la nervosité implicite aux systèmes adaptables mériteraient une étude approfondie. Cela sort cependant de la portée du cadre conceptuel auquel ce chapitre est consacré.

2.3.1.3 Bilan

Pour répondre à la première question « *Qu'est-ce qui peut évoluer ?* », nous venons de définir un espace des possibles de façon à en tirer le meilleur profit. Même si les attributs variables sont un élément crucial dans cette caractérisation, ils n'existent que par les objets qu'ils décrivent et ne constituent donc pas un élément à part entière. De plus, malgré la pertinence d'y inclure les contraintes, celles-ci sont liées à chaque problème en particulier et peuvent donc conduire à une perte de généralité. Nous nous concentrons donc sur le couple <Objets modifiables, Types de changement>. La Figure 2.3 d'[ElMaraghy 2006] en présente quelques éléments. Le Tableau 2.1 fournit quant à lui une représentation plus large des objets modifiables dans les systèmes de production associés à leurs types de changement (l'espace des possibles), mais il n'en constitue pas pour autant une liste exhaustive. Pour les cas qui le dépassent, il reste tout à fait possible de suivre le formalisme proposé en l'adaptant en conséquence. Enfin, les références bibliographiques illustrent des cas publiés portant sur un attribut variable d'un objet et un type de changement en particulier.

2.3.2 Quand évoluer ?

2.3.2.1 Informations utilisées

Avec tout ce qui est susceptible à adaptation dans un système de production, la décision de quoi changer (quels attributs de quels objets) et à quel moment le faire dépend de la situation dans laquelle le système évolue. Il est donc important d'avoir des informations et/ou mesures pertinentes qui permettront la prise de ces décisions. Elles doivent notamment détecter, parmi les différents changements dans les conditions de fonctionnement, ceux qui justifient réellement d'une quelconque adaptation.

Généralement, dans les entreprises manufacturières, d'énormes quantités de données sont recueillies à différents niveaux. Elles peuvent être de différentes natures, qualitatives ou quantitatives, donc plus ou moins difficiles à mesurer [Barbosa 2015]. Afin d'avoir un système de production plus efficace, ces données disponibles doivent être transformées en informations et connaissances, c'est-à-dire qu'elles doivent être organisées, transférées et interprétées dans le but d'être utiles dans le processus décisionnel [Dudas *et al.* 2011]. Ces informations, regroupées dans l'ensemble d'informations I, comportent plusieurs aspects, parmi lesquels on trouvera typiquement :

- L'état du système (par exemple, les niveaux des stocks et des encours, la disponibilité des ressources, l'affectation des opérateurs) ;
- Des données historiques (par exemple, la fréquence des pannes et les temps effectifs pour leur correction par l'équipe de manutention, l'intervalle entre l'arrivée des demandes) ;
- Des prévisions ou des plans de production ;

Objet	Attribut variable	Type de changement	Exemples
Atelier	Nombre de machines	Ajout/Retrait de machines	[Deif et ElMaraghy 2006, Yildirim *et al.* 2006, Aryanezhad *et al.* 2009, Baqai 2010]
	Nombre d'opérateurs	Recrutement/Licenciement	[Aryanezhad *et al.* 2009]
	Calendrier	Changement des horaires d'ouverture-fermeture	
Machine	Position	Déplacement	[Aryanezhad *et al.* 2009, Baqai 2010, Barbosa *et al.* 2015]
	Taux de production	Augmentation/Réduction du taux de production	[Tamani *et al.* 2009]
	Fonctionnalité ou type d'opération (outils)	Ajout/Retrait/Déplacement de composants/outils	[Deif et ElMaraghy 2006, Baqai 2010, Chalfoun *et al.* 2012]
Opérateur	Affectation à un ensemble de tâches/machines	Réaffectation	[Aryanezhad *et al.* 2009, Salerno 2009]
	Compétences (niveau de flexibilité)	Formation	[Aryanezhad *et al.* 2009]
Pièce	Gamme de fabrication	Utilisation d'une gamme alternative	[Baqai 2010]
	Allocation	Réallocation	[Borangiu *et al.* 2010, Leitão *et al.* 2012, Pach *et al.* 2012, Barbosa *et al.* 2015]
Date de livraison	Coefficient	Augmentation/Réduction du coefficient	[Yildirim *et al.* 2006]
Règle de priorité	Règle d'ordonnancement (FIFO, SPT)	Changement de règle d'ordonnancement	[Yildirim *et al.* 2006, Mouelhi-Chibani et Pierreval 2010]
	Règle d'affectation (FAM, LBM, ECT)	Changement de règle d'affectation	
Politique de flux	Type (tiré, poussé)	Changement de type	
	Nombre de cartes	Ajout/Retrait de cartes	[Tardif et Maaseidvaag 2001, Takahashi *et al.* 2004, Belisário et Pierreval 2015]
Stock	Politique de stockage	Changement de mode de gestion	
	Seuil de réapprovisionnement	Augmentation/Réduction du seuil	
Ressource transport	Source	Changement de source	
	Destination	Changement de destination	[Leitão *et al.* 2012, Pach *et al.* 2012]
	Circuit	Changement de circuit	[Leitão *et al.* 2012, Pach *et al.* 2012]
	Affectation	Réaffectation	

Tableau 2.1 – Natures des changements dans différents systèmes de production

- Des mesures de performance (par exemple, la rotation des stocks, le nombre de tâches en retard, le surplus ou la rupture de stock, les taux de panne et d'utilisation des ressources, le nombre de réglages (*setup*) par machine, l'avancement du plan de production) ;
- Des informations externes, liées à l'environnement du système (par exemple, l'évolution des coûts de matières premières ou d'heures supplémentaires, les prévisions de demande du marché).

Les informations caractérisant l'état courant du système définissent ses conditions opératoires à un instant donné. Elles sont appelées variables d'état et peuvent changer dynamiquement au fil du temps. La valeur courante des attributs variables en fait partie $(a_{j,k,t}, \forall j, k)$. Pour certaines variables d'état, il peut être utile de connaître non seulement leur valeur courante mais aussi leur évolution. Une série d'observations peut être constituée avec les différentes valeurs qu'elles ont prises au cours du temps ($t' \leq t$) où la dernière observation correspond à celle de l'instant courant. Cette série peut également être restreinte aux données les plus récentes qui sont ciblées en définissant la durée de stockage ou le nombre d'observations à stocker. Les données brutes constituent l'historique. Pour en faciliter l'interprétation, celui-ci peut être présenté sous la forme d'indicateurs et, dans ce cas, faire partie des mesures de performance. Ce même raisonnement peut s'appliquer aux prévisions à la différence que nous aurons des informations sur le futur ($t' \geq t$). Bien évidemment, les informations externes peuvent également faire l'objet d'historiques et/ou de prévisions. Enfin, pour avoir une description plus générale du système, il peut aussi être intéressant de prendre en compte ses paramètres qui font référence à ce qui reste constant durant une période d'étude donnée.

Il est important de signaler que ces différentes informations peuvent être utiles avant et après une adaptation. Dans le premier cas, elles servent de tableau de bord pour « surveiller » le système et identifier les potentiels d'adaptation (par l'évaluation des éventuels avantages en adaptant le système à un moment donné). Dans le second cas, elles permettent un suivi du système pour évaluer l'adaptation faite. Il faut en revanche trouver les bonnes mesures indiquant l'intérêt d'une évolution ainsi que les bons critères d'évaluation d'un changement. La pertinence des informations utilisées dépend donc bien évidemment des objectifs (G) définis pour le système [Shahzad et Mebarki 2012].

Le choix de ces informations est en réalité encore plus délicat. Si des éléments importants sont omis, les décisions seront « pénalisées » par ce manque d'information. Inversement, trop de détails créent un bruitage inutile et ajoutent de la complexité au problème liée à l'identification de ce qui doit être vraiment retenu. Ainsi, la dépendance entre indicateurs tout comme le temps de calcul qu'ils engendrent sont aussi des considérations à prendre en compte lors de leur sélection [Shahzad et Mebarki 2012]. Si cette sélection n'est pas faite correctement, à savoir dans le manque ou l'excès d'information, les décisions seront très probablement biaisées et présenteront une forme de myopie sociale [Zambrano Rey 2014, Zambrano Rey *et al.* 2014b]. Il faut entendre par ce terme une limitation concernant la collecte de données (accès) et leur analyse à posteriori (capacités de

calcul). Plusieurs travaux se sont donc consacrés à la sélection des informations permettant la prise de décision, comme ceux de [Shiue et Guh 2006]. Ces derniers ont cherché à identifier les attributs les plus importants parmi un vaste ensemble disponible dans le cadre de l'ordonnancement adaptatif.

Bien que les informations effectivement utilisées dépendent de l'horizon décisionnel auquel on s'intéresse ($H_{D,l}, \forall l$), des exemples typiques souvent utilisées dans la littérature peuvent être identifiés. Citons le plan de production décomposé en périodes [Aryanezhad *et al.* 2009] ; le nombre de machines, leur fonctionnalité et leur position [Baqai 2010] ; le nombre de pièces dans chaque file d'attente et leur date de livraison [Mouelhi-Chibani et Pierreval 2010] ; l'existence de tâches prioritaires [Reaidy 2003] ; les niveaux d'encours et de capacité [Deif et ElMaraghy 2006] ainsi que le niveau des stocks [Deif et ElMaraghy 2007] ; l'état actuel des ressources, le moment où elles seront disponibles et le nombre actuel de places disponibles dans les files d'attente à capacité limitée [Pach *et al.* 2012] ; les taux de production des machines, les niveaux des stocks, les encours et l'erreur (déficit ou excédent) de production [Tamani *et al.* 2009] ; le nombre de machines et l'utilisation de la capacité disponible [Yildirim *et al.* 2006] ; ou encore le profit et l'utilisation pondérée des ressources (avec l'importance relative de chaque tâche en tant que facteur de pondération) [Shin *et al.* 2012]. Notons cependant que la difficulté à obtenir certaines données n'est pas à exclure [Edwards *et al.* 2004].

Les différentes adaptations d'un système de production impliquent généralement des coûts de production supplémentaires, qu'ils soient explicites ou implicites, influant sur les décisions de quand mettre en place quel changement. Ces coûts sont souvent difficiles à quantifier mais devraient être pris en compte parmi les informations dans I. Tout d'abord, une adaptation prend du temps et généralement, au moins un opérateur sera requis pour l'effectuer. Si aucun opérateur n'est disponible, il faudra ajouter une gestion adéquate des priorités entre les diverses tâches. Les effets indésirables peuvent aussi inclure des confusions dans l'atelier, des coûts élevés de réordonnancement et la fluctuation du taux d'utilisation des capacités [Campbell 1971]. Un autre point important est la montée en cadence de la production, analysée par [Surbier *et al.* 2013] dans le cadre de l'introduction d'un nouveau produit, mais qui peut aussi être liée à d'autres adaptations apportées au système. Dans des situations complexes, des interruptions de la production peuvent même se produire. [Barbosa *et al.* 2015] signalent notamment le risque de voir le système basculer vers un comportement chaotique suite à des changements nombreux. Ainsi, la nervosité (discutée précédemment) peut être particulièrement perturbatrice dans les processus de production, notamment lorsqu'un changement à un niveau peut en entraîner d'autres à d'autres niveaux (effet coup de fouet (ou *bullwhip*) par exemple).

On peut alors trouver différents freins à l'évolution selon ce que l'on envisage changer : des coûts de transport (par exemple, si les machines doivent être déplacées), des coûts de formation (pour permettre de réaffecter les opérateurs à un ensemble plus large d'activités), des coûts liés à des perturbations apportées à la production (qui peut avoir besoin d'être arrêtée) ou encore des difficultés humaines (par exemple, le stress induit,

la fatigue, la dégradation de la qualité du travail, la résistance au changement). Une réaffectation peut par exemple induire le mécontentement des opérateurs et par voie de conséquence une résistance au changement. L'attitude du décideur est également importante car un changement présente toujours un risque : on va donc être plus ou moins prêt à changer si le décideur est « neutre » ou « averse » au risque par exemple [Von Neumann et Morgenstern 1944]. Soulignons enfin que les systèmes de production ajoutent eux-mêmes des contraintes bien que souvent involontairement. Citons leurs limites au niveau du traitement des données et de leur capacité de détection et d'action en raison de leurs caractéristiques intrinsèques (plage de fonctionnement, précision, temps de réaction, entre autres) [Saenz de Ugarte 2009].

Tous ces obstacles sont souvent très complexes à prendre en charge et semblent être très peu discutés dans la littérature liée à l'adaptation des systèmes de production. Les auteurs ont plutôt tendance à en citer quelques-uns sans pour autant les explorer ou les mesurer. Voici quelques exemples à titre illustratif.

Dans [Aryanezhad *et al.* 2009], différents coûts sont utilisés, à savoir, pour l'achat, l'installation, le retrait et le déplacement des machines, et pour le recrutement, le licenciement et la formation des opérateurs. Le coût et/ou le temps de reconfiguration est mentionné par [Baqai 2010, Bensmaine *et al.* 2013, Borisovsky *et al.* 2013] tandis qu'[Essafi 2010] cite l'importance de pouvoir disposer d'indicateurs quantitatifs pour évaluer le niveau de reconfigurabilité d'un système donné. À ce propos, [Barbosa 2015] signale la difficulté de les obtenir malgré certains travaux où des métriques et méthodologies sont décrites [Brennan et Norrie 2003, Leitão 2004, Trentesaux *et al.* 2013]. [Takahashi *et al.* 2004] identifient quant à eux le temps de mise en œuvre de l'adaptation correspondant à un changement de situation détecté. Dans le même esprit, [Wiendahl et Breithaupt 1999, Deif et ElMaraghy 2006] mentionnent plus précisément le temps non négligeable nécessaire au changement de capacité. Cela est également abordé par [Deif et ElMaraghy 2007] qui ajoutent le coût d'augmentation de la capacité et celui de capacité non utilisée. Les erreurs humaines sont quant à elles évoquées par [Gunasekaran 1999]. Dans le même registre et tout en envisageant un système interactif d'aide multicritère à la décision, [Trentesaux 1996] identifie la surcharge des opérateurs qui compromet la qualité de leurs décisions. [Trentesaux *et al.* 1998] signalent également l'importance d'intégrer les ressources humaines comme partie du système de gestion pour sa réactivité.

2.3.2.2 Stratégies de déclenchement

En plus de l'identification des informations nécessaires au processus d'adaptation, il faut définir l'aspect temporel du changement qui symbolise quand étudier si oui ou non des adaptations doivent être faites. Le moment auquel il est souhaitable de vérifier les informations disponibles dans I pour prendre des décisions se repose sur une stratégie de déclenchement T. Les différentes possibilités décrites ci-après sont illustrées par la Figure 2.6.

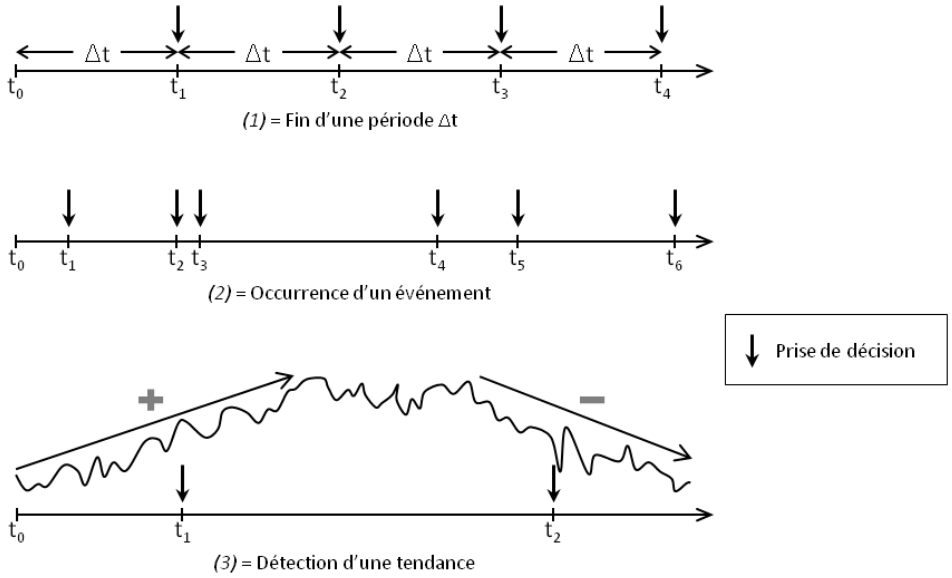

FIGURE 2.6 – Différentes stratégies de déclenchement pour la prise de décision

Des modifications peuvent être apportées à des intervalles fixes de temps, donc à l'aide d'une période fixe Δt : à la fin de chaque période, une décision est prise et indique si des modifications seront apportées pour la prochaine période (et si oui, lesquelles). Dans ce premier cas, la durée de la période doit être définie et c'est souvent un mécanisme de régulation qui lui sera associé. Les modifications peuvent également être basées sur l'occurrence de certains événements, appelés événements déclencheurs, tels que l'introduction d'un nouveau produit dans le catalogue de l'entreprise, des pannes ou le début d'une grève. Dans ce deuxième cas, ce sont les événements méritant attention qui doivent être identifiés au préalable. Ceci peut être une source de difficulté car, comme observé par [Mirdamadi 2009], il n'est pas forcément évident de détecter et identifier des événements qui ont un impact sur les objectifs (et de mesurer leurs conséquences). Si ces deux stratégies de déclenchement sont basées sur des « instants pré-identifiés » auxquels nous pouvons réagir, il est aussi souvent utile d'observer des éventuelles tendances dans les mesures importantes (internes ou externes) qui demanderaient une adaptation du système de production (par exemple, la satisfaction des clients qui diminue ou les encours qui s'accumulent). Ce troisième cas implique qu'un mécanisme adéquat de détection de tendances soit à disposition et que les informations pertinentes à surveiller aient été identifiées. À cet égard, le temps pour détecter un changement de situation est un aspect important [Takahashi *et al.* 2004].

Enfin, même si des prévisions adéquates ne sont pas toujours disponibles dans le contexte actuel des marchés incertains, elles sont, quand elles existent, extrêmement utiles pour décider quand et comment le système devrait être adapté. Ainsi, des changements peuvent également être déclenchés selon un plan donné. En effet, les plans de production fournissent à la fois des renseignements utiles sur ce que le système doit produire et sur le moment où des adaptations peuvent être nécessaires. S'ils sont disponibles, ils doivent donc être inclus dans I et T.

Il est important de signaler que ces stratégies peuvent également être combinées. Par exemple, les décisions périodiques n'empêchent pas celles déclenchées par la manifestation de certains événements considérés comme critiques. Dans une telle situation, il faudra établir la façon de considérer les périodes : soit indépendamment des événements, soit par rapport au dernier déclenchement (même si celui-ci correspond à un événement). Considérons en exemple qu'un processus décisionnel est déclenché chaque lundi et qu'un événement se produit un vendredi. Dans le premier cas, le processus sera déclenché le vendredi de l'événement mais par la suite, il continuera à être déclenché tous les lundis. Dans le deuxième cas en revanche, il sera déclenché chaque vendredi à partir de l'événement. La Figure 2.7 schématise ces deux fonctionnements.

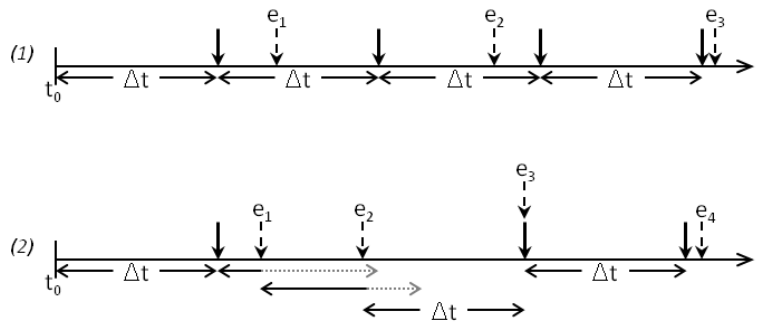

FIGURE 2.7 – Deux façons de combiner des déclenchements périodiques et événementiels

Ci-après, nous essayons d'illustrer et de « classer » des stratégies de déclenchement utilisées par plusieurs chercheurs dans divers types de problèmes d'adaptation. Lorsque l'on se base sur une stratégie de déclenchement à événements, nous pouvons identifier comme en étant à l'origine : l'arrivée d'un nouveau produit [Koren *et al.* 1999, Baqai 2010, Chalfoun *et al.* 2012] ; la substitution d'un produit [Essafi 2010, Bensmaine *et al.* 2013, Borisovsky *et al.* 2013] ; un changement de gamme opératoire [Essafi 2010] ; des changements des caractéristiques d'un produit [Koren *et al.* 1999, Essafi 2010] ; la fin d'une opération sur un produit [Reaidy 2003] ; des changements de statut des ressources [Leitão *et al.* 2012, Pach *et al.* 2012] ; ou encore des pannes [Trentesaux 1996, Mrabet *et al.* 2001, Borangiu *et al.* 2010]. Les plans de productions sont quant à eux plutôt liés à une stratégie périodique [Deif et ElMaraghy 2006, Deif et ElMaraghy 2007, Aryanezhad *et al.* 2009]. Il existe d'autres types de déclencheurs tels que la déviation du niveau de stock souhaité [Deif et ElMaraghy 2007], l'accumulation des encours [Deif et ElMaraghy 2006], la sous-utilisation de la capacité disponible [Yildirim *et al.* 2006] ou la dérive d'un temps de cycle [Mirdamadi 2009]. Selon la façon dont ils sont utilisés, ils peuvent être classés comme des événements, comme des détections de tendances ou tout simplement comme des éléments à vérifier lors de la mise en place d'une stratégie périodique.

La stratégie de déclenchement T détermine alors quand éventuellement adapter le système. Elle n'est pas sans lien avec les horizons $H_{D,l}$ et $H_{min,l}$ définis auparavant. Puisque ces derniers sont tous les deux associés à chaque type de changement, la stratégie T peut être choisie de façon appropriée en fonction des différents éléments considérés dans C.

2.3.2.3 Bilan

L'identification de ce que l'on peut modifier dans le système (O et C) est importante. Toutefois, le processus de remise en question, et par conséquent le moment de faire une adaptation, n'arrive pas seul : des changements doivent être pilotés et il faut donc spécifier quelles sont les informations disponibles I mais aussi quand le processus décisionnel est déclenché (T). La deuxième question « *Quand évoluer ?* » concerne donc l'espace temporel de la prise de décision. Une décision peut être prise en fonction de différents déclencheurs et les changements peuvent alors survenir : périodiquement, suite à un événement ou à une tendance ainsi que selon un plan de production ou des prévisions. Dans tous les cas, il faut disposer de différentes informations et/ou mesures. Il existe bien évidemment des freins à l'évolution mais aussi des indicateurs de potentiel et de suivi d'évolution qui aident à détecter les moments les plus propices à des adaptations.

2.3.3 Comment décider de l'évolution ?

Afin de décider quand et comment réadapter un système de production, il lui faut un composant dédié. C'est la raison pour laquelle nous avons introduit la notion de superviseur (Section 2.2). Celui-ci est responsable de l'analyse des informations disponibles pour mener à bien le processus d'adaptation. Le superviseur peut être plus ou moins complexe et implémenté de différentes manières dans les entreprises. Plusieurs considérations intéressantes sur sa conception peuvent être trouvées par exemple dans [Bousbia et Trentesaux 2002, Jimenez *et al.* 2013]. Ici, nous distinguons essentiellement trois dimensions importantes qui mettent en évidence ses principales caractéristiques, à savoir : sa structure organisationnelle S, son mécanisme de changement M et ses objectifs G. Tandis que S et M sont tous les deux uniques pour un système donné, les décisions de changement sont faites sur la base d'un ou plusieurs objectifs déclarés dans l'ensemble G.

2.3.3.1 Structure organisationnelle

La structure organisationnelle S du système superviseur concerne son architecture de contrôle, donc la coordination des flux d'information. Elle détermine la manière dont les rôles et les responsabilités sont attribués, contrôlés et/ou coordonnés et comment l'information circule. Ce composant dépend en quelque sorte des objectifs mais surtout de la stratégie de l'entreprise. Divers types de structures existent dans la littérature et y sont largement développés. Typiquement, nous trouvons les structures centralisée, hiérarchique, coordonnée, distribuée ou distribuée supervisée (voir par exemple [Trentesaux 1996, Berchet 2000, Habchi 2001, Reaidy 2003, Mirdamadi 2009]). Certaines sont représentées dans la Figure 2.8. Cependant, il peut y avoir un manque de consensus autour de leur définition. Notons par exemple que plusieurs auteurs utilisent les termes décentralisé et hétérarchique en faisant référence à une structure distribuée alors qu'une distinction est

faite par [Mirdamadi 2009] où un système distribué correspond à une distribution totale des capacités de décision tandis qu'un système décentralisé implique que chaque décideur soit situé au même niveau fonctionnel. [Bousbia et Trentesaux 2002] soulignent ce manque de clarté et la diversité des définitions concernant ces terminologies.

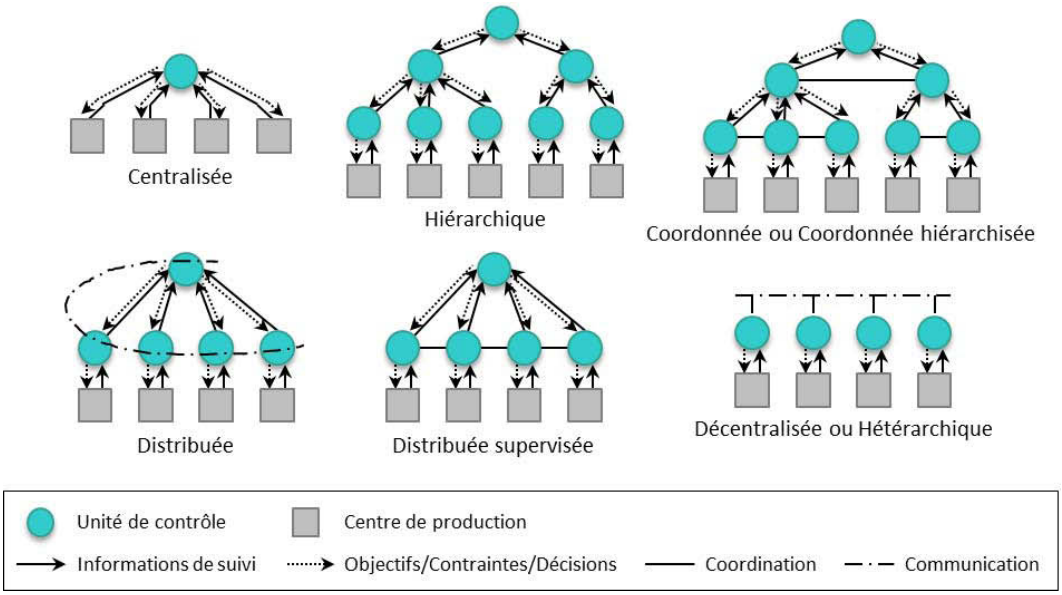

FIGURE 2.8 – Représentations de structures de pilotage trouvées dans la littérature

Toujours concernant S, il semble de plus exister une confusion sur ce qu'elle représente. D'une part, il y a la structure du système de production (ou, plus globalement, de l'entreprise). D'autre part, il y a la structure plus précisément liée au mécanisme de décision implémenté par le superviseur. Elles sont généralement fortement liées et beaucoup d'auteurs n'apportent pas de précision sur ce à quoi ils font référence et les utilisent souvent en tant que synonymes. Parfois cependant, une distinction mérite d'être faite. Cela est même nécessaire lorsque certaines difficultés liées à la mise en place des systèmes adaptables sont évoquées. Par exemple, [Tharumarajah 2003] signale une difficulté de conception, plus précisément au niveau de la découverte d'une structure appropriée qui supporte les changements. À ce propos et selon [Trentesaux 2002], les liens hiérarchiques peuvent entraver ces changements. Ceci est aussi abordé par [Reaidy 2003, Salerno 2009, Barbosa *et al.* 2015] qui parlent des systèmes traditionnels, souvent centralisés et hiérarchiques, donc rigides et peu adaptables. Il est important de remarquer que, lorsque ces structures sont dites incompatibles avec un environnement changeant, elles précisent l'organisation de l'entreprise et non du processus mis en place pour gérer les adaptations. C'est dans ce même esprit qu'[ElMaraghy 2006] souligne que l'existence des composants modulaires nécessaires à l'implantation des systèmes reconfigurables peut être une difficulté pour réaliser cette dernière.

Plusieurs formes concrètes de S existent dans la littérature : distribuée supervisée [Trentesaux 1996, Deif et ElMaraghy 2007, Tamani *et al.* 2009], coordonnée hiérarchisée [Berchet 2000, Habchi et Berchet 2003], holonique [Cardin et Castagna 2009, Borangiu

et al. 2010, Borangiu *et al.* 2015] ou encore hétérarchique holonique [Pach *et al.* 2012]. Dans [Shin *et al.* 2009], c'est une organisation fractale qui est adoptée comme architecture de contrôle prenant une forme dite quasi-hétérarchique. Les systèmes multi-agents sont largement utilisés en tant qu'approche de modélisation, et ce malgré leur manque de perspective globale [Leitão *et al.* 2012]. Ils sont surtout liés à des structures distribuées, comme c'est le cas dans [Reaidy 2003] et [Leitão *et al.* 2012]. D'autres travaux ne spécifient pas la structure utilisée mais il est parfois possible de la déduire. Ainsi, les méthodes d'optimisation comme dans [Essafi 2010, Borisovsky *et al.* 2013] induisent souvent une structure centralisée. Celles fournissant un ensemble de solutions non-dominées comme dans [Bensmaine *et al.* 2013] peuvent quant à elles être liées à une structure hiérarchique.

2.3.3.2 Mécanisme de changement

Le mécanisme de changement M est utilisé pour décider des adaptations à faire ou non sur le système. Il définit donc le fonctionnement du superviseur pour prendre des décisions appropriées et peut lui aussi être basé sur plusieurs approches telles que la négociation, l'apprentissage ou encore l'optimisation dynamique. Par ailleurs, il doit être capable de gérer l'aspect combinatoire du problème bien qu'il puisse exister des solutions pour limiter cet aspect [Mittal et Frayman 1989]. Il doit également gérer les décisions simultanées et/ou asynchrones [Angelidis *et al.* 2012]. Notons que dans les systèmes dits à auto-apprentissage, M contient une mémoire utilisée pour conduire le processus d'adaptation.

Le chapitre précédent visait à décrire plusieurs travaux en lien avec cette thèse qui illustrent en détail ce composant du cadre conceptuel et sur lesquels nous ne reviendrons pas. Cependant, nous avons identifié plusieurs difficultés qui méritent d'être évoquées. Elles ne sont pas liées à des adaptations proprement dites, mais plutôt à leur gestion. Commençons par l'utilisation de la simulation des systèmes de production. C'est une technique souvent coûteuse en temps de calcul [Monostori *et al.* 2000] et qui soulève le problème de la quasi-impossibilité d'intervenir de manière automatisée et structurée alors qu'elle est en cours [Habchi 2001]. De plus, elle n'est pas considérée comme adaptée à modéliser les processus de décision [Habchi et Berchet 2003]. Les méthodes d'apprentissage quant à elles ont un paramétrage difficile [Metan *et al.* 2010]. Le temps de calcul qu'elles impliquent peut être considérable [Sotskov *et al.* 2013] et il n'est pas aisé de déterminer la méthode la plus appropriée selon l'application [Markham *et al.* 2000].

D'un point de vue plus général, [Macías-Escrivá *et al.* 2013] s'intéressent à l'application pratique et signalent le manque de compréhension et donc de confiance des utilisateurs vis-à-vis des mécanismes implémentés pour l'adaptation. Par ailleurs, [Cámara *et al.* 2014] insistent sur un aspect souvent négligé bien qu'essentiel : l'importance de tester la robustesse de ces mécanismes afin de valider leur fiabilité. [Leitão *et al.* 2012] les rejoignent dans cette optique en précisant qu'il faut qu'une technologie soit mature pour être acceptée en entreprise. Des considérations sont aussi faite dans [Mrabet *et al.* 2001]. Elles portent sur les critères et contraintes supplémentaires à respecter (tels que l'intégrabilité

et l'ergonomie) lors du choix des outils de développement pour un déploiement industriel.

Enfin, même si M correspond à l'origine au mécanisme de changement à proprement dit, nous avons pu voir qu'il indique souvent l'approche ou les principes utilisés pour l'obtenir. Ceci permet de mieux différencier les travaux du domaine. Dans [Tardif et Maaseidvaag 2001] par exemple, M est un ensemble de règles basé sur des seuils pour l'ajout ou le retrait de cartes dans un système de production à flux tiré, seuils qui sont obtenus par une méthode de recherche locale gloutonne. L'optimisation pourrait donc être utilisée pour caractériser M. C'est également le cas pour toute méthode d'apprentissage : un tel type de méthode représente un moyen pour obtenir un résultat pouvant prendre différentes formes mais qui sera ensuite utilisé à part entière. Une application en ligne fonctionnant en permanence constitue une exception puisqu'elle fait effectivement partie du mécanisme de changement à proprement dit.

2.3.3.3 Objectifs

Dans tous les cas, pour que le mécanisme M choisi soit efficace, il doit décider des adaptations à réaliser ou non tout en ciblant un ou plusieurs objectifs dans G. Ses éléments peuvent être par exemple {produire au moindre coût, avoir de courts délais de livraison, maintenir les encours aussi bas que possible, etc.}. Quelques éléments dans I peuvent aussi être utiles pour mesurer les objectifs puisqu'ils fournissent parfois des critères d'évaluation des changements et que le suivi du système permet d'évaluer les adaptations faites. Le superviseur agit alors en accord avec G de sorte que ces objectifs doivent être clairement mis en évidence [Burlat et Campagne 2001]. Cependant et comme observé dans [Saenz de Ugarte 2009], il est parfois difficile d'identifier les bonnes mesures de performance à utiliser. De plus, l'aspect bien souvent contradictoire des objectifs est à considérer avec attention [Wiendahl et Breithaupt 1999, Tamani *et al.* 2009].

Notons également que ces objectifs peuvent être de différentes natures. [Barbosa 2015] identifie par exemple l'existence de deux types de KPIs (pour *Key Performance Indicators*) : qualitatifs et quantitatifs, donc plus ou moins difficiles à mesurer. De même, [Brennan et Norrie 2003] proposent deux classes de métriques, une pour le système de contrôle et l'autre pour le système contrôlé. Dans cette thèse, nous nous limiterons à celle concernant le système contrôlé. Étant donnée notre problématique d'adaptation, nous considérons qu'elle évalue par elle-même la performance du système de contrôle : c'est ce dernier qui mène le système contrôlé à la performance atteinte. L'évaluation exclusive du système de contrôle, par exemple par le flux de messages tel qu'illustré dans [Brennan et Norrie 2003], sort du cadre de cette thèse.

2.3.3.4 Bilan

La troisième question se résume donc à « *Comment décider de l'évolution ?* ». Elle concerne la conception du « système superviseur » et plus précisément la définition de quand et où appliquer les différents changements possibles, de sorte que le système soit toujours adapté à la situation du moment. Elle constitue par ailleurs et très probablement le principal défi pour la conception (et la réussite) de ces systèmes adaptables auxquels nous nous intéressons.

2.4 Conclusion

Dans ce chapitre, nous avons proposé un cadre conceptuel pour le contexte actuel caractérisé par un fort besoin d'adaptation de la part des systèmes de production. Il est basé sur une représentation formelle prenant la forme d'un 7-uplet. Par la même occasion, de nombreuses considérations ont été faites sur les composants de ce cadre conceptuel et nous avons notamment mis en avant plusieurs difficultés liées à la conception et à la gestion des systèmes adaptables.

Inspiré d'approches systémiques, notre cadre conceptuel s'est avéré suffisamment générique par sa confrontation avec les contributions dans la littérature. Il peut donc être appliqué à plusieurs types de systèmes. Il a pour but d'aider à concevoir, décrire, spécifier, analyser, comprendre et par conséquent exploiter les systèmes de production adaptables et les problématiques qui en découlent. Il peut par ailleurs apporter un support à l'analyse et au classement de la littérature du domaine. Cette proposition représente donc une tentative afin de mieux formaliser les caractéristiques de ces systèmes adaptables.

Le chapitre suivant sera consacré à l'introduction d'un possible « mécanisme de changement ». Il s'agit de celui sur lequel cette thèse se repose, à savoir l'apprentissage par simulation.

Chapitre 3

Apprentissage à partir de la Simulation

3.1 Introduction

Depuis bientôt un demi-siècle, les chercheurs en intelligence artificielle (IA) travaillent à programmer des machines capables d'effectuer des tâches qui requièrent de l'intelligence (aide à la décision, reconnaissance de formes, prédiction, etc.) [Cornuéjols et Miclet 2003]. Cette intelligence est généralement issue de connaissances « expertes » et est éventuellement capable de prendre en compte de l'imprécision comme dans les approches floues ou possibilistes [Mouelhi et Pierreval 2007]. Cependant, il est difficile de programmer des machines capables de s'adapter à toutes les situations voire d'évoluer en fonction des nouvelles contraintes [Cornuéjols et Miclet 2003]. Cela est principalement dû à l'élicitation des connaissances (pertinentes) dont doivent être dotées ces machines, ce qui constitue un écueil majeur pour leur mise en œuvre. Dans la littérature, il a été suggéré que cette connaissance puisse être acquise par apprentissage [Pierreval et Ralambondrainy 1990, Huyet 2004, Metan *et al.* 2010]. L'enjeu est ainsi de contourner cette difficulté en dotant une machine de capacités d'apprentissage lui permettant de tirer parti de son expérience [Cornuéjols et Miclet 2003].

Dans la plupart des approches d'apprentissage, l'objectif est généralement d'extraire des connaissances à partir d'exemples et/ou d'observations. Dans le contexte des systèmes de production adaptables, il est malheureusement difficile de trouver de bons exemples indiquant comment piloter efficacement leur adaptation. [Mouelhi-Chibani 2009] insiste sur la difficulté d'acquérir des exemples d'apprentissage qui soient adaptés aux problèmes liés au comportement des systèmes de production. Leur état changeant dynamiquement au cours du temps rend difficile la détermination des « bonnes pratiques » dans le domaine. De plus, puisque les décisions dans un tel contexte s'enchaînent, c'est un ensemble de décisions qui s'avère être performant ou non et non pas chaque décision individuellement. Ceci souligne donc l'aspect dynamique du problème dû à la nature dynamique des systèmes de production eux-mêmes. Mais alors comment peut-on apprendre ?

C'est ici que la simulation intervient étant donné son fort potentiel pour la modélisation du comportement de nombreux systèmes complexes et notamment des systèmes de production. Nous nous intéressons donc à comment l'utiliser pour apprendre à adapter ces systèmes, voire pour les rendre capables eux-mêmes de s'auto-adapter.

Nous présenterons tout d'abord diverses approches d'apprentissage. L'objectif n'est

pas de les détailler mais d'en fournir les notions de base. Cela fera ressortir l'importance des exemples d'apprentissage pour leur fonctionnement. Ensuite, nous nous concentrerons sur l'intérêt de la simulation pour l'application de ces approches dans le domaine de la production, ainsi que sur les différents travaux dans la littérature. Cela permettra de mettre en évidence les façons dont ces techniques sont couplées et utilisées (qui ne se montrent pas toujours suffisantes).

3.2 Notions générales sur l'apprentissage

Les deux paragraphes suivants sont principalement constitués d'extraits de [Cornuéjols et Miclet 2003].

La définition classique d'apprentissage en sciences cognitive est « la capacité à améliorer les performances au fur et à mesure de l'exercice d'une activité ». L'assimilation de l'expérience et la puissance de raisonnement se combinent au fil du temps. On sait alors, consciemment ou non, en abstraire des règles ou les faire évoluer afin de mieux effectuer une tâche. À cela s'ajoute la faculté à généraliser rationnellement, autrement dit à appliquer ces règles à des situations non encore rencontrées. Les modalités d'apprentissage naturel sont donc multiples : par cœur, par instruction, par généralisation ou par découverte, entre autres. Mais qu'en est-il pour les machines ?

La notion d'apprentissage artificiel, plus couramment appelé apprentissage automatique, englobe toute méthode permettant de construire un modèle de la réalité à partir de données, soit en améliorant un modèle partiel ou moins général, soit en créant entièrement ledit modèle. L'apprentissage artificiel peut être définit comme la science qui cherche et établit des liens entre les principes généraux d'apprenabilité et les méthodes et outils permettant de réaliser un apprentissage dans un contexte particulier. Il s'agit de rechercher les conditions de réalisabilité des modèles d'apprentissage, c'est-à-dire les lois qui règlent la possibilité d'apprendre.

L'apprentissage naturel, surtout humain, est à l'apprentissage automatique ce que l'intelligence humaine est à l'intelligence artificielle, essentiellement une source de modèles ; d'où son importance [Hammoud 2011].

En effet, même si l'apprentissage est un des champs d'étude de l'IA et donc avant tout un sous-domaine de l'informatique, il s'intéresse à la façon dont les systèmes améliorent leur propre performance en se basant sur leur expérience : il est donc tentant de s'inspirer des êtres vivants pour concevoir des machines capables d'apprendre. Ainsi, l'apprentissage se trouve intimement lié aux sciences cognitives, aux neurosciences, à la biologie et à la psychologie. Il pourrait, à la croisée de ces domaines, aboutir à des systèmes d'intelligence artificielle plus performants. Par la suite, le terme « apprentissage » désignera l'apprentissage artificiel (ou automatique).

L'apprentissage fait référence au développement, à l'analyse et à l'implémentation de méthodes qui permettent à une machine (au sens large) d'évoluer grâce à un processus automatique, et ainsi de remplir des tâches difficiles voire impossibles à réaliser avec des moyens algorithmiques plus classiques [Michalski *et al.* 1993]. C'est un processus qui s'effectue progressivement à partir d'une série d'expériences. Selon [Frawley 1989], le processus d'apprentissage comprend :

- L'acquisition de nouvelles connaissances ;
- Le développement de compétences par l'instruction et la pratique ;
- L'organisation (ou réorganisation) des connaissances par des représentations effectives (taxonomies) ;
- La découverte de nouveaux faits et théories par l'observation et l'expérimentation.

Du point de vue de la recherche et toujours selon le même auteur, les objectifs typiques de l'apprentissage sont :

- Développer et analyser des systèmes pour améliorer la performance d'une tâche prédéterminée ;
- Investiguer et simuler le processus d'apprentissage humain ;
- Explorer des possibilités théoriques (méthodes et algorithmes) indépendamment du domaine d'application.

Il existe plusieurs types ou problèmes d'apprentissage qui sont définis par un certain nombre de caractéristiques parmi lesquelles l'espace des données, l'espace des hypothèses et le protocole régissant les interactions de l'apprenant avec son environnement [Cornuéjols et Miclet 2003]. Le terme apprenant désigne le système « qui doit apprendre la tâche ». Ainsi, les méthodes (d'apprentissage) peuvent être classifiées selon de nombreux points de vue différents tels que les informations sur les exemples d'apprentissage, le type de représentation des connaissances et le type de stratégie ou technique d'inférence utilisée [Michalski *et al.* 1993, Mitchell 1997, Cornuéjols et Miclet 2003, Hammoud 2011].

À cet égard, les informations disponibles concernant les exemples d'apprentissage définissent le mode d'apprentissage que les algorithmes emploient. La littérature distingue en particulier les modes supervisé, non supervisé et par renforcement [Cornuéjols et Miclet 2003]. Si l'on considère le type de représentation des connaissances, il existe deux tendances principales en apprentissage : celle issue de l'intelligence artificielle qualifiée de symbolique et celle issue des statistiques qualifiée de numérique ou connexionniste [Michalski *et al.* 1993, Kayser 1997, Mitchell 1997, Cornuéjols et Miclet 2003]. Les arbres de décision et les « règles de production » sont des exemples symboliques tandis que les réseaux de neurones sont numériques. Enfin et concernant les stratégies d'inférence, il y a notamment l'induction et la déduction qui ont des démarches opposées [Nilsson 1980, Hammoud 2011].

Par la suite, nous nous concentrerons sur ce qui est le plus couramment trouvé dans la littérature dans le but de dégager des notions globales d'apprentissage : les modes et

les algorithmes. Nous donnerons un aperçu des notions et méthodes connues et largement revisitées dans la littérature, sans pour autant être exhaustifs. Cela nous permettra de cibler les méthodes qui correspondent à notre problématique et de mieux positionner notre travail par rapport à ce domaine spécifique de recherche.

3.2.1 Différents modes d'apprentissage : une classification par les informations disponibles

3.2.1.1 Apprentissage supervisé

Dans l'apprentissage supervisé, les exemples d'apprentissage sont préalablement étiquetés (par un expert ou oracle) afin d'identifier la classe à laquelle ils appartiennent. Les classes sont alors prédéterminées et les exemples connus : on cherche un modèle (ou fonction) pour prédire l'étiquette de nouvelles données. L'apprenant doit alors trouver ou approximer la fonction (ou produire des règles) qui permet de généraliser pour des entrées inconnues ce qu'il a pu « apprendre » grâce aux données déjà traitées (soit un ensemble de couples entrée-sortie). Ceci peut être schématisé par la Figure 3.1.

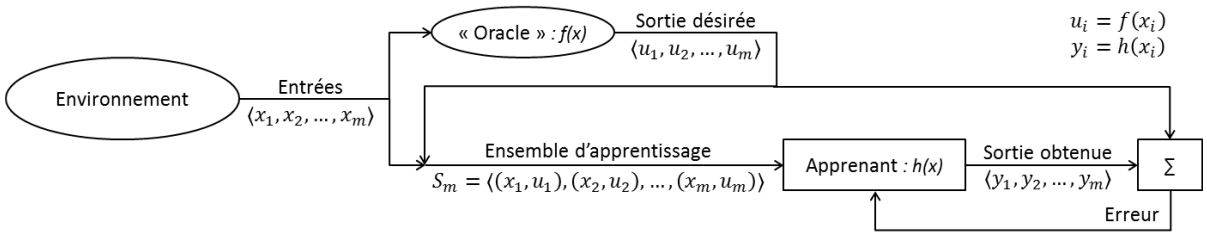

FIGURE 3.1 – Processus d'apprentissage supervisé (inspiré de [Cornuéjols et Miclet 2003, Benmammar 2009])

Parfois il est préférable d'associer une donnée non pas à une classe unique mais à une probabilité d'appartenance à chacune des classes prédéterminées. On parle alors d'apprentissage supervisé probabiliste. Indépendamment et selon la nature de l'ensemble de sortie, on distingue généralement deux types de problèmes pouvant être traités : lorsque la sortie est une valeur dans un ensemble continu de réels, on parle d'un problème de régression ; et lorsque l'ensemble des valeurs de sortie est de cardinal fini, on parle d'un problème de classification. Un troisième type apparaît lorsque la sortie est complexe (fonctions, chaînes de caractères, arbres, graphes) : c'est un problème dit de prédiction structurée [Geurts *et al.* 2006, Maes 2009].

Une comparaison empirique de dix algorithmes d'apprentissage supervisé est proposée par [Caruana et Niculescu-Mizil 2006]. L'analyse discriminante linéaire et les SVM (machines à vecteurs de support) en sont des exemples typiques, ainsi que le *boosting*, la régression logistique (ou modèle logit), les réseaux de neurones, la méthode des k plus proches voisins, les arbres de décision ou encore la classification naïve bayésienne.

3.2.1.2 Apprentissage non supervisé

L'apprentissage non supervisé (ou classification automatique) se distingue du supervisé par le fait qu'il n'y a pas de sortie à priori : on dispose d'exemples (d'apprentissage) mais non d'étiquettes. Autrement dit, il y a en entrée un ensemble de données collectées dont on ne connaît pas la classe d'appartenance, le nombre de classes et leur nature n'étant pas prédéterminés. L'apprenant doit découvrir par lui-même la structure plus ou moins cachée des données (Figure 3.2). Il procède alors par regroupement en cherchant les données similaires (selon une certaine métrique), d'où l'appellation « *clustering* ». Ici encore, si l'approche est probabiliste (chaque exemple, au lieu d'être associé à une seule classe, est caractérisé par un jeu de probabilités d'appartenance à chacune des classes), on parle alors de *soft clustering* ou regroupement doux (par opposition au *hard clustering* ou regroupement strict).

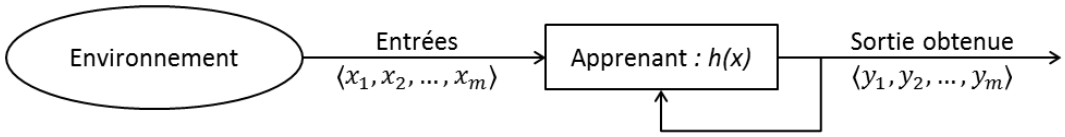

FIGURE 3.2 – Processus d'apprentissage non supervisé (inspiré de [Benmammar 2009])

Pour trouver des régularités/similitudes dans une collection d'exemples afin de les classer en groupe homogènes, leurs attributs disponibles sont ciblés. Une technique employée consiste à implémenter des algorithmes pour rapprocher les exemples les plus similaires et éloigner ceux qui ont le moins de caractéristiques communes, la similarité étant généralement calculée selon une fonction de distance entre les paires d'exemples. Ces groupes d'exemples similaires sont parfois appelés des prototypes.

Le partitionnement de données (ou *data clustering*) est un exemple d'algorithme. Nous pouvons aussi citer l'algorithme espérance-maximisation et, ici encore, les réseaux de neurones.

3.2.1.3 Apprentissage par renforcement

L'apprentissage par renforcement [Sutton et Barto 1998] vise quant à lui à résoudre des problèmes complexes lorsqu'il n'est pas possible d'avoir un ensemble d'apprentissage à disposition, la seule information étant le but à atteindre. L'apprenant doit ainsi apprendre, à partir de l'environnement dans lequel il évolue, comment atteindre le but fixé par une succession d'essais. Autrement dit, il va découvrir par tentatives et erreurs ce qu'il convient de faire en différentes situations. Le comportement décisionnel (appelé aussi stratégie ou politique) est ainsi une fonction associant à l'état courant l'action à exécuter (ou décision à prendre). Pour que le comportement conçu soit le plus optimisé possible, ce type d'apprentissage fonctionne avec des récompenses que renvoie l'environnement. Elles peuvent être encourageantes ou punitives : l'action de l'apprenant sur l'environnement

produit une valeur de retour qui le guide de façon à optimiser au cours du temps une récompense (numérique), dite aussi le signal de renforcement. La Figure 3.3 résume ces échanges.

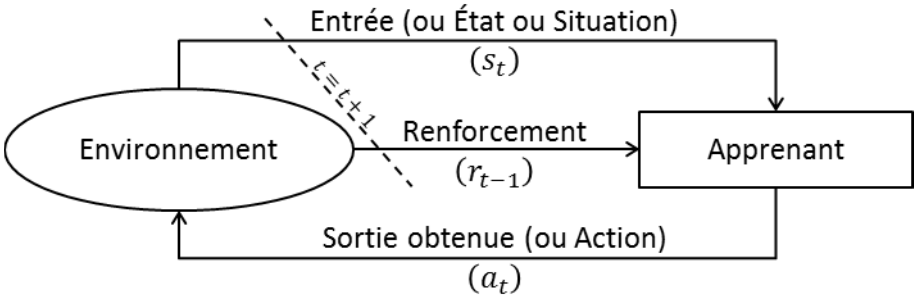

FIGURE 3.3 – Processus d'apprentissage par renforcement (inspiré de [Mitchell 1997, Sutton et Barto 1998])

L'approche dérive de formalisations théoriques de méthodes de contrôle, et plus précisément du processus de décision markovien. Son usage rencontre en revanche quelques limitations quand les espaces d'états et d'actions sont très vastes [Benmammar 2009]. Par ailleurs et d'après [Cornuéjols et Miclet 2003], le signal de renforcement fourni en retour, souvent très pauvre, apporte peu d'information sur l'environnement et les décisions à prendre. Selon eux, ce problème s'accentue par le délai qui sépare ce signal des décisions qui y ont conduit rendant ardue l'attribution d'une bonne ou mauvaise performance à chacune des décisions prises dans le passé.

Une autre difficulté est liée au dilemme exploitation/exploration [Meuleau 1996]. L'exploitation sert à se rappeler quelles actions ont bien fonctionné par le passé. La stratégie extrême (ou pure) associée consiste alors à refaire les actions dont on connaît déjà la récompense afin d'obtenir la plus grande. Ceci n'est pas sans risque d'être sur un optimum local ou sous l'influence du hasard, donc de laisser passer l'occasion de découvrir des meilleures actions. L'exploration quant à elle a pour but de trouver les bonnes actions. La stratégie extrême inverse consiste à parcourir de nouvelles actions pour un état donné à la recherche de la récompense cumulée la plus grande, au risque d'adopter parfois un comportement sous-optimal. Il faut alors trouver un compromis entre ces deux composantes de sorte que le système exploite ce qu'il connaît, tout en explorant plusieurs possibilités pour découvrir l'action qui conduit au meilleur rendement.

Pour ce qui est des algorithmes, le Q-*learning* est un exemple classique tout comme le TD-*learning* et le SARSA.

3.2.2 Différentes méthodes ou algorithmes d'apprentissage

De nombreuses méthodes d'apprentissage ont été développées dans la littérature et peuvent être combinées pour obtenir des variantes. Le spectre d'applications de l'appren-

tissage est très large : moteur de recherche, détection de données aberrantes ou manquantes, aide au diagnostic, détection de fraudes, analyse des marchés financiers, reconnaissance de la parole ou de l'écriture manuscrite, analyse et indexation d'images et de vidéo, robotique ou encore bio-informatique, entre autres. Ainsi, l'utilisation d'un algorithme dépend fortement de la tâche à résoudre (classification, estimation de valeurs, etc.) mais également du désir d'interprétabilité du modèle obtenu.

3.2.2.1 Apprentissage par induction

L'apprentissage par induction est l'une des principales méthodes étudiées dans le domaine de l'apprentissage automatique [Osório 1998]. L'idée est d'acquérir des règles générales qui représentent les connaissances obtenues à partir d'exemples, en passant du particulier (des faits) au général (la loi) [Nilsson 1980]. Ce type d'apprentissage est alors synthétique dans la mesure où il consiste à synthétiser et construire de nouvelles connaissances [Hammoud 2011]. Pour ce faire, [Mitchell 1997] explique qu'une méthode de généralisation de connaissances est utilisée, aussi appelée apprentissage ou acquisition de concepts. Parmi les approches empiriques les plus connues, nous avons les réseaux de neurones et les arbres de décision [Mouelhi-Chibani 2009], ce qui montre que les règles obtenues peuvent être représentées d'une façon explicite (et donc facilement interprétable) ou implicite avec un codage qui n'est pas toujours facile à interpréter.

3.2.2.1.1 Arbres de décision

L'induction par arbres de décision est l'une des formes d'apprentissage les plus simples et efficaces, souvent utilisée en pratique [Murthy 1998, Hammoud 2011]. Les termes méthode d'induction descendante ou TDIDT (pour *Top Down Induction of Decision Trees*) sont utilisés dans la littérature pour les qualifier.

À l'origine du paradigme de la représentation par arbres se trouvent très probablement [Morgan et Sonquist 1963], les premiers à les utiliser dans un processus de prédiction et d'explication. À son apogée, citons [Breiman *et al.* 1984] avec la méthode CART (pour *Classification And Regression Tree*). En apprentissage, la plupart des travaux s'appuient sur la théorie de l'information et il est d'usage de citer [Quinlan 1979] avec sa méthode ID3 (*Induction of Decision Tree*) rattachée aux travaux d'[Hunt 1962]. Des variantes existent dont notamment le concept de graphes latticiels [Terrenoire 1970] popularisé par les graphes d'induction [Rakotomalala 1997, Zighed et Rakotomalala 2000] qui constituent une généralisation des arbres de décision au cas non arborescent, modifiant la forme du concept lui-même en sortant du cadre classique [Rakotomalala 2005].

Le principe général s'exprime comme suit : chercher à discriminer les exemples selon leur classe en fonction d'attributs considérés comme « les meilleurs » parmi tous les autres au sens d'un critère donné. Une succession de partitions de plus en plus homogènes

(au sens du critère) est construite. Pour cela, on recherche l'attribut qui discrimine « le mieux » les exemples sur l'ensemble d'apprentissage et l'on fixe autant de sous-ensembles que l'attribut possède de valeurs. Chaque sous-ensemble contiendra les exemples ayant la valeur d'attribut qu'il représente. Ce processus se répète sur chaque sous-ensemble jusqu'à ce que tous les exemples d'un même sous-ensemble possèdent la même classe. Le résultat est donc la représentation graphique d'un processus de classification récursif où les attributs étiquettent les nœuds internes (ou branches) et à toute description complète est associée une seule feuille.

Dans les processus arborescents classiques, la seule opération possible lors de la construction de l'arbre de décision est celle de segmentation des exemples selon une variable (ou attribut). Dans les graphes, une opération de fusion permet de regrouper des individus de mêmes caractéristiques appartenant à deux feuilles différentes de l'arbre. Ceci confère à l'algorithme la capacité de minimiser le nombre de feuilles, assurant à chacune d'elles un effectif plus important, ainsi qu'une meilleure résistance à la fragmentation des données [Oliver 1993, Rakotomalala 1997]. Dans tous les cas, chaque chemin (qui part de la racine pour aboutir à une feuille) correspond à une règle exprimée sous la forme « *si condition alors conclusion* » dans laquelle « condition » désigne une ou plusieurs propositions logiques de type <attribut, valeur> [Zighed et Rakotomalala 2000]. L'ensemble des règles constitue ainsi le modèle de prédiction.

Un arbre de décision, aussi appelé arbre d'induction (de l'anglais *induction decision tree*), est ainsi composé d'une structure hiérarchique en forme d'arbre. Traditionnellement, il se construit par raffinements de sa structure, en commençant par la racine. C'est un outil d'aide à la décision et à l'exploration de données discrètes et continues. La lisibilité est un point fort auprès des praticiens puisqu'un arbre peut non seulement être lu et interprété directement mais aussi traduit en base de règles sans perte d'informations. Sa lisibilité et sa rapidité d'exécution expliquent donc sa popularité [Utgoff 1989]. Il reste cependant des difficultés liées par exemple à la manipulation des variables continues [Osório 1998].

Un exemple d'arbre de décision largement connu de la littérature est illustré dans la Figure 3.4 ci-dessous. Il prédit le comportement d'un sportif (jouer ou non) selon les conditions météorologiques à partir d'un ensemble d'exemples recueillis au préalable (Tableau 3.1 extrait de [Quinlan 1986]).

Dans la Figure 3.5 nous retrouvons le même exemple représenté sous forme de graphe d'induction.

3.2.2.1.2 Réseaux de neurones

Un réseau de neurones artificiel est un modèle de calcul dont la conception est très schématiquement inspirée du fonctionnement des neurones biologiques. Il a vocation à imiter certains mécanismes du cerveau humain. Il est constitué d'un graphe orienté et pondéré dont les nœuds sont les neurones dits formels (ou artificiels) généralement or-

Numéro	Ciel	Température	Humidité	Vent	Classe
1	Ensoleillé	Élevée	Forte	Non	NON
2	Ensoleillé	Élevée	Forte	Oui	NON
3	Couvert	Élevée	Forte	Non	OUI
4	Pluvieux	Moyenne	Forte	Non	OUI
5	Pluvieux	Basse	Normale	Non	OUI
6	Pluvieux	Basse	Normale	Oui	NON
7	Couvert	Basse	Normale	Oui	OUI
8	Ensoleillé	Moyenne	Forte	Non	NON
9	Ensoleillé	Basse	Normale	Non	OUI
10	Pluvieux	Moyenne	Normale	Non	OUI
11	Ensoleillé	Moyenne	Normale	Oui	OUI
12	Couvert	Moyenne	Forte	Oui	OUI
13	Couvert	Élevée	Normale	Non	OUI
14	Pluvieux	Moyenne	Forte	Oui	NON

Tableau 3.1 – Données de l'historique constituant l'ensemble d'apprentissage

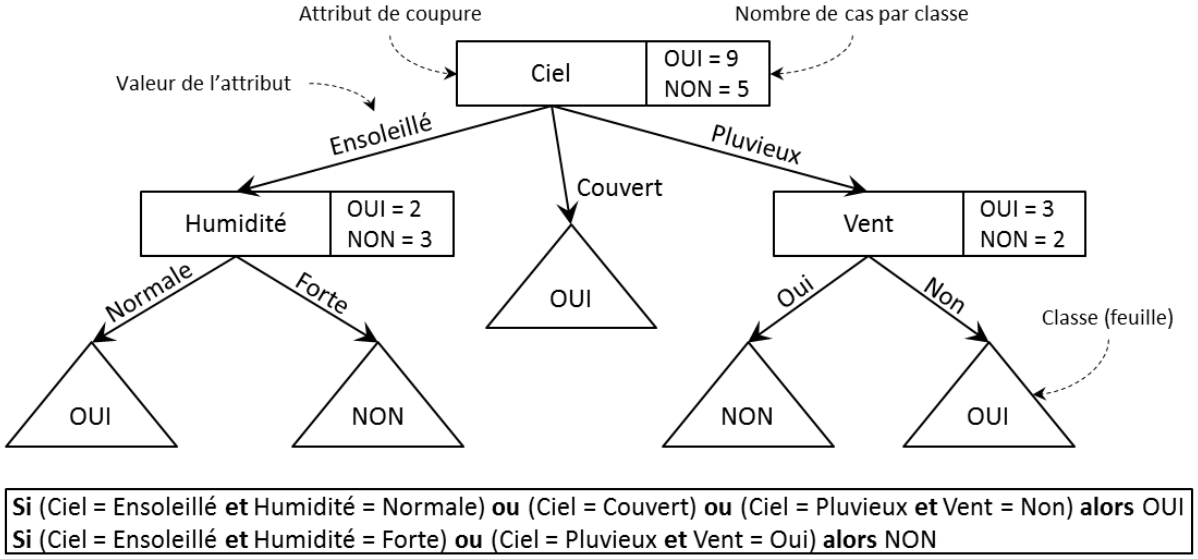

FIGURE 3.4 – Exemple d'arbre de décision

ganisés en une succession de couches. Ils sont dotés d'un état interne dit d'activation pouvant être propagé aux autres en passant par des arcs pondérés appelés connexions, liens, coefficients ou poids synaptiques. Chaque neurone est ainsi une fonction algébrique (un opérateur mathématique) non linéaire et bornée dont la valeur (numérique) dépend de ces paramètres [Dreyfus *et al.* 2004]. Habituellement, les variables de cette fonction et sa valeur de retour sont respectivement appelées les « entrées » et la « sortie » du neurone, sachant que chaque couche (de neurones) prend ses entrées sur les sorties de la précédente. La règle qui détermine l'activation d'un neurone en fonction de ses entrées et de leurs poids respectifs s'appelle règle ou fonction d'activation. La constitution d'un neurone peut alors être schématisée par la Figure 3.6 qui suit.

Les réseaux de neurones peuvent par exemple être classifiés selon leur architecture.

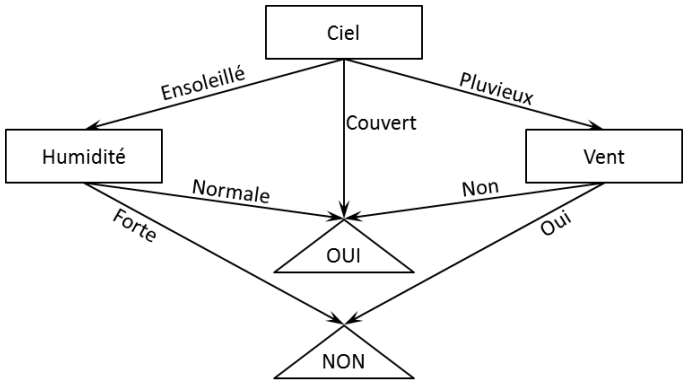

FIGURE 3.5 – Exemple de graphe d'induction

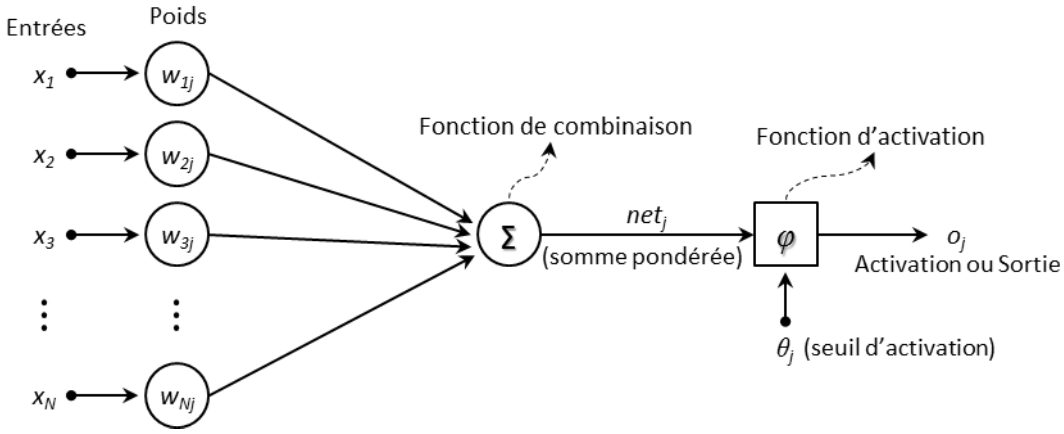

FIGURE 3.6 – Structure générique d'un neurone formel

Dans la plupart des cas, ils sont multicouches : une couche d'entrée, une ou plusieurs cachées et une de sortie. Toutefois, il en existe aussi à une seule couche ainsi que ceux, plus rares, qui ne sont pas organisés en couches. Ils peuvent être complètement ou partiellement connectés, et ce pour une organisation ou non en couches. En ce qui concerne la façon dont l'information circule, les plus utilisés dans la littérature sont les réseaux dits non bouclés ou unidirectionnels (et alors statiques), où un graphe acyclique ne permet pas de retours en arrière. Au contraire, les réseaux bouclés ou récurrents contiennent des interconnexions depuis la sortie d'un neurone vers lui-même, un autre neurone de la même couche ou d'une couche inférieure. Ceci permet de modéliser des aspects temporels et des comportements dynamiques où la sortie d'un neurone peut dépendre de son état antérieur. Les boucles internes rendent cependant ce type de réseau instable et son utilisation plus complexe [Osório 1998]. Différents exemples d'architecture sont illustrés dans la Figure 3.7 ci-dessous, et la Figure 3.8 fait ressortir la structure la plus utilisée dans la littérature.

Indépendamment de son type, pour qu'un réseau de neurones puisse être utilisé en tant que fonction de prédiction selon les valeurs qui lui sont données en entrée, il doit d'abord passer par une phase d'apprentissage. Les changements apportés aux valeurs des poids ou dans la structure d'interconnexion des neurones sont responsables des changements de

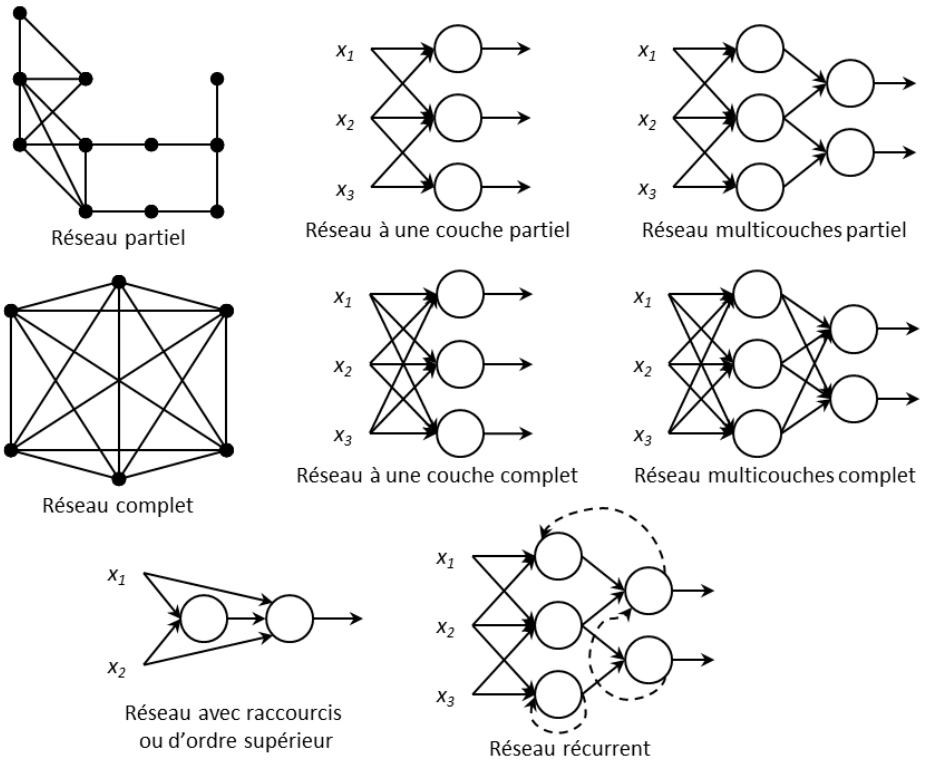

FIGURE 3.7 – Exemples de réseaux de neurones

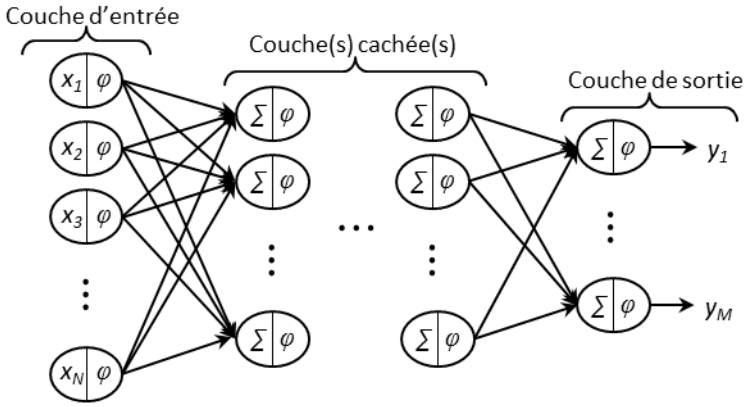

FIGURE 3.8 – Vue simplifiée d'un réseau de neurones classique

comportement d'activation du réseau, ce qui lui permet d'apprendre un nouveau comportement. Ainsi, le réseau est capable d'établir des associations entrée-sortie (stimulus et réponse) afin de bien résoudre un problème.

L'apprentissage est en général un processus graduel, itératif, où les poids du réseau sont modifiés plusieurs fois selon une règle dite d'apprentissage avant d'atteindre leurs valeurs finales. Il nécessite en général une grande quantité de données (ensemble d'apprentissage). Ici encore une classification est possible. Cela se fait le plus souvent par rapport à un comportement de référence en cherchant à minimiser l'erreur entre les sorties réelles et

désirées (et donc de façon supervisée) ; mais aussi en fonction de critères internes tels que la coactivation des neurones (non supervisée).

3.2.2.2 Réseaux bayésiens

Issus du domaine de la statistique, les réseaux (d'inférence) bayésiens sont des modèles (graphiques) qui permettent de représenter des situations de raisonnement probabiliste (entre des faits) à partir de connaissances incertaines. Cette représentation est faite sous la forme d'un graphe orienté acyclique.

D'une part, leur aspect qualitatif en fait des modèles de représentation de connaissances. Les relations causales (ou liens de causalité) entre variables (aléatoires) d'intérêt sont décrites par un graphe dont les nœuds correspondent aux variables et les arcs aux dépendances directes entre les variables qu'ils relient. Les liens manquant indiquent ainsi une certaine indépendance entre les variables non reliées.

D'autre part, leur aspect quantitatif fait qu'ils servent aussi à calculer des distributions de probabilités. Chaque nœud possède une table de probabilités conditionnelles de la variable qu'il représente, et ce suivant ses parents dans le graphe.

Les relations de cause à effet entre les variables ne sont alors pas déterministes mais probabilistes (Figure 3.9). Ainsi, l'observation d'une (ou plusieurs) cause(s) n'entraîne pas systématiquement l'effet ou les effets qui en dépendent mais modifie seulement la probabilité de les observer.

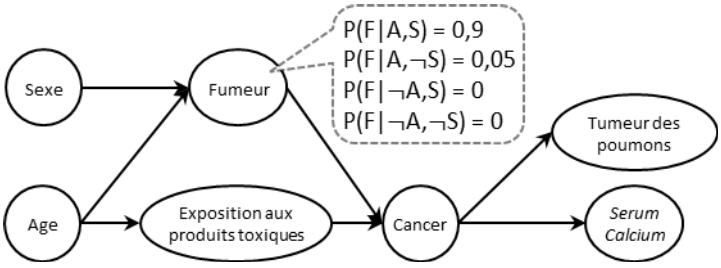

FIGURE 3.9 – Exemple de réseau bayésien extrait de [Cornuéjols et Miclet 2003]

Il est important de signaler qu'un réseau bayésien comprend à la fois une structure et des paramètres (les probabilités conditionnelles) qui y sont associés. L'apprentissage peut ainsi être utilisé soit pour estimer ses tables de probabilités soit pour estimer aussi sa structure, dans les deux cas à partir de données.

3.2.2.3 Machines à vecteurs de support

Les machines à vecteurs de support ou séparateurs à vaste marge (SVM pour *Support Vector Machine*) sont un ensemble de techniques d'apprentissage destinées à résoudre des

problèmes de discrimination (prédiction d'appartenance à des groupes prédéfinis) et de régression (analyse de la relation d'une variable par rapport à d'autres). Nous pouvons citer par exemple [Boser *et al.* 1992] et [Cortes et Vapnik 1995] comme étant à leur origine (en termes de popularité).

Étant donné un ensemble d'exemples d'apprentissage, chacun étiqueté comme appartenant à une parmi deux catégories possibles, l'idée derrière un SVM est de construire un modèle pour attribuer de nouveaux exemples dans une catégorie ou dans l'autre. Ceci en fait un classificateur linéaire binaire non probabiliste. Le modèle obtenu est une représentation des exemples comme points dans l'espace, placés de sorte que les exemples de catégories distinctes soient séparés par un hyperplan dont la marge de séparation est aussi large que possible (Figure 3.10). De nouveaux exemples peuvent alors être placés dans ce même espace qui prédit leur appartenance à une catégorie selon le côté où ils se trouvent.

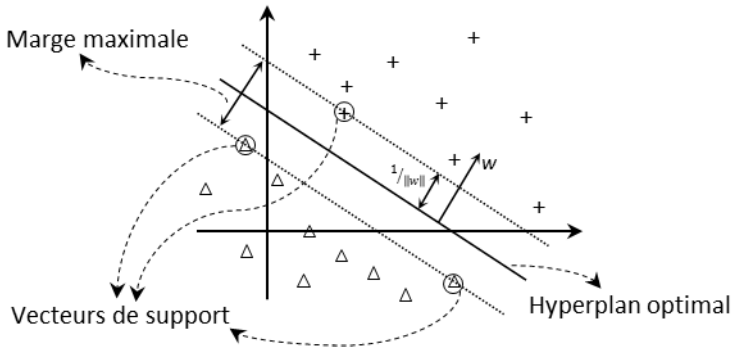

FIGURE 3.10 – Principe de fonctionnement d'un SVM

La technique peut aussi être utilisée efficacement pour effectuer une classification non linéaire. Pour cela, une fonction noyau transforme l'espace de représentation des données d'entrée en un espace de plus grande dimension dans lequel il est probable qu'il existe une séparatrice linéaire. Les SVM sont donc une généralisation des classificateurs linéaires. Conçus pour une séparation des données en deux ensembles, ce sont des classificateurs binaires.

3.2.2.4 Programmation génétique

La programmation génétique (PG) est une sous-catégorie des algorithmes évolutionnistes. Issue, à l'origine, de la programmation évolutionniste, c'est une méthode d'apprentissage utilisant les concepts fondamentaux des algorithmes génétiques appliqués à des programmes [Mahboub 2011]. Contrairement aux méthodes classiques d'algorithmes évolutionnistes où les solutions sont généralement représentées par des structures de taille fixe, la PG utilise des structures de longueur variable et peut donc être utilisée pour la création automatique de programmes informatiques pouvant représenter des formules ou des algorithmes. Elle imite les programmeurs (humains) qui construisent des programmes

en les réécrivant progressivement. La technique est connue pour pouvoir générer des heuristiques puissantes bien qu'il n'y ait pas de garantie d'atteindre l'optimum [Kleinau et Thonemann 2004]. Traditionnellement, elle est en mesure de résoudre les mêmes types de problèmes que d'autres techniques d'apprentissage telles que les réseaux de neurones [Brameier et Banzhaf 2007, Mahboub 2011].

Malgré les différentes variantes parues ces dernières décennies, la représentation la plus répandue et emblématique de la PG reste sa déclinaison originelle sous forme d'arbres proposée par [Koza 1992]. Par ailleurs, c'est une représentation très connue dans le domaine de l'apprentissage [Murthy 1998]. En PG, un arbre est composé de fonctions (opérateurs et fonctions relationnels, arithmétiques et conditionnels) manipulant un nombre d'entrées pertinentes à un domaine de problèmes particulier pour produire les résultats requis. Chaque entrée, appelée terminal, est composée d'une variable spécifique au problème ou d'une constante. Les fonctions sont situées dans les nœuds internes de l'arbre tandis que les feuilles contiennent les constantes et variables d'entrée (Figure 3.11). L'espace de recherche est alors constitué de tous les arbres de longueurs diverses qui peuvent être construits en utilisant des ensembles prédéfinis de fonctions et de terminaux. Cela signifie que leur choix est essentiel en ce qui concerne l'efficacité de l'algorithme. La composition de ces ensembles détermine leur expressivité : ils doivent être assez puissants pour représenter la solution optimale (ou au moins un « bon » optimum local), mais trouver une solution devient plus difficile si l'espace de recherche est augmenté inutilement par un trop grand nombre d'éléments. Pour résoudre cette situation de compromis (*trade-off*), des connaissances de l'utilisateur sur le domaine du problème sont nécessaires et, dans la pratique, il est recommandé de définir ces ensembles aussi réduits que possible [Brameier et Banzhaf 2007].

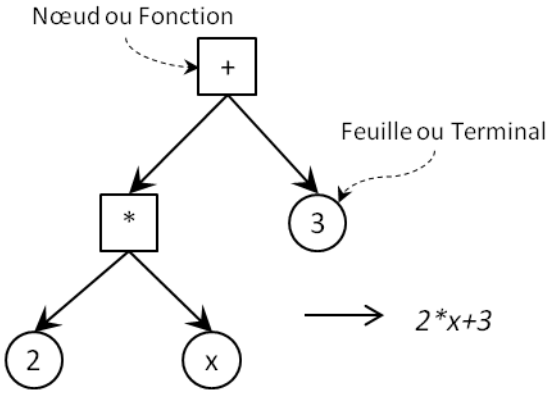

FIGURE 3.11 – Exemple d'expression sous forme d'arbre

Pour représenter des programmes répondant à un problème donné, le principe de la PG est le suivant : générer (le plus souvent aléatoirement) une population de programmes dits individus ; évaluer leur adéquation en leur attribuant ce que l'on appelle leur *fitness* ; appliquer des opérateurs génétiques (croisement, mutation, reproduction) sur la population courante afin d'en créer une nouvelle ; sélectionner les individus les mieux adaptés

à l'environnement (les programmes répondant le mieux au problème posé) ; revenir à la phase d'évaluation et répéter les étapes qui suivent jusqu'à ce qu'un critère d'arrêt soit atteint.

Il est important de s'assurer de la validité syntaxique des individus crées au long de l'évolution. L'évaluation de leur *fitness* reste l'étape la plus critique en temps de calcul alors que leur génération est beaucoup moins coûteuse voire négligeable [Brameier et Banzhaf 2007].

Pour plus de détails sur la PG, se référer à [Poli *et al.* 2008] ainsi qu'aux travaux de [Koza 1992, Koza 1994, Koza *et al.* 1999, Koza *et al.* 2003] qui en sont à l'origine et qui ont fait émerger le potentiel de cette approche.

3.2.2.5 Méthodes hybrides

Dans les sous-sections précédentes, nous avons présenté différentes méthodes d'apprentissage. Étant donnés les problèmes inhérents à chacune d'entre elles, les chercheurs essaient de trouver de nouvelles solutions plus performantes. Dans ce contexte, les solutions hybrides se présentent comme une tendance dans les recherches développées sur les systèmes intelligents [Kandel et Langholz 1992, Michalski et Tecuci 1994, Sahin *et al.* 2012]. Le concept de système (ou méthode) hybride est cependant très large et inclut toute méthode qui intègre deux approches différentes pour la solution d'un problème. Ceci permet certainement de tirer profit de chacune des approches qui la constituent (tout en palliant les déficiences de chacune d'entre elles) mais une méthode hybride reste difficile à mettre en œuvre [Shavlik et Towell 1989, Hammoud 2011].

Dans le but de donner un aperçu des possibilités de combinaison, voici quelques exemples connus d'hybridation : système symbolique-flou, symbolique-génétique, génético-flou, neuro-génétique ou encore neuro-symbolique. Ce dernier peut être de type neuro-flou, neuro-IDT (avec un arbre de décision), neuro-CBR (avec un système de raisonnement fondé sur des cas) et neuro-KBS ou SHNS (avec un système à base de connaissances). Pour plus de détails, se référer par exemple à [Osório 1998].

3.3 Notions générales sur la simulation

La modélisation et la simulation de flux sont le processus de conception d'un modèle pour un système réel et celui de conduite d'expérimentations sur ce modèle dans le but d'en comprendre le fonctionnement et d'évaluer différentes stratégies de pilotage [Pegden *et al.* 1995]. La simulation est alors l'évolution du modèle étudié dans le temps, à partir d'hypothèses à tester, afin de connaître son comportement dynamique et de pouvoir prédire l'effet de tel paramètre ou scénario sur le système [Hill 1996, Claver *et al.* 1997]. Elle permet donc de répondre à des questions du type « *Que se passerait-il si... ?* », ce qui

en fait un outil d'aide à la décision qui permet l'analyse des différentes solutions possibles sans pour autant déterminer laquelle est la meilleure. Cela rend intéressant son couplage à un algorithme d'optimisation. Pour plus de détails sur l'optimisation via simulation, se référer à [Fu 1994, Fu 2002, Tekin et Sabuncuoglu 2004, Lee *et al.* 2013, Wang et Shi 2013].

La simulation est une technique généralement efficace pour rendre des problèmes difficiles davantage traitables [Monostori *et al.* 2000]. Elle est largement utilisée pour étudier des systèmes complexes [Fishwick 1991]. Un modèle (de simulation) décrit des entités (éléments basiques d'un système physique), leurs attributs (caractéristiques principales et relations) et comment elles interagissent dans le but de voir comment le système se comporte sur une période de temps [Doukidis et Angelides 1994]. La notion de temps est ici extrêmement importante et distingue deux types de modèles. Suivant la spécification des états du système pouvant être dénombrable ou non, le temps est vu comme discret ou continu et l'on parle alors de simulation discrète ou continue [Law 2014]. Dans le premier cas, l'intérêt se porte sur les événements qui sont à l'origine des changements d'état tandis que dans le second, des équations différentielles sont souvent utilisées. Dans la littérature, la simulation à événements discrets est plus usitée que la simulation continue pour les systèmes de production [Pierreval 2006], c'est d'ailleurs l'un des outils choisis pour cette thèse. À noter cependant qu'en simulation, une gestion discrète du temps dans le modèle n'est pas forcément liée à la nature discrète du système modélisé et des flux qu'il comporte : ce qui est continu peut parfois être discrétisé.

Les systèmes de production réels sont sujets à de nombreux phénomènes aléatoires qui peuvent perturber leur fonctionnement. Leurs gestionnaires cherchent à les gérer de façon à que ces phénomènes n'aient qu'un impact limité sur la production. La simulation est un outil capable non seulement de vérifier une solution proposée (empiriquement ou par la résolution analytique par exemple), mais aussi de fournir les informations manquantes sur la dynamique des flux en prenant en compte des données stochastiques et les événements aléatoires. Cela la rend potentiellement puissante pour conduire des expérimentations tout en sachant que les données de sortie, influencées par l'aléa, ne sont qu'une estimation [Doukidis et Angelides 1994]. De plus, [Castagna *et al.* 2001] ont montré que la simulation n'est plus seulement un outil d'aide à la conception des systèmes de production mais aussi un puissant outil d'aide au pilotage lorsqu'elle est couplée à d'autres applications du type ERP ou MES par exemple.

Non négligeable mais bien meilleur que celui d'un système réel, le temps de réponse de la simulation permet de tester un grand nombre de combinaisons de paramètres [Mirdamadi 2009]. Cela s'ajoute au fait que la simulation rend l'expérimentation possible avant même l'existence du système réel, ce qui élimine le besoin de prototypes. Il est ainsi possible d'observer le comportement d'un modèle dans des conditions bien définies, y compris des situations inhabituelles ou anormales [Ören 1994]. La simulation peut contribuer à l'élaboration de nouveaux algorithmes, au soutien des décideurs, à la diminution des risques d'investissement et au fonctionnement plus efficace des systèmes exposés aux

changements et perturbations [Monostori *et al.* 2000]. Elle peut également aider au choix de solutions favorisant le développement durable [Jaegler et Burlat 2012]. La technique de simulation fournit non seulement un moyen pour énoncer une théorie mais aussi un critère très fort pour tester si la déclaration est appropriée [Ören 1994]. Cela s'avère utile : un expert, poussé par des situations simulées spécifiques, peut énoncer son expertise et la tester avec le modèle de simulation pour s'assurer qu'une représentation adéquate a été obtenue [Doukidis et Angelides 1994].

Les vastes potentiels de la simulation se doivent aussi à sa capacité de représenter et donc de couvrir tous les flux de l'entreprise, qu'ils soient physiques, logiques ou décisionnels. Elle peut également couvrir toutes les phases du cycle de vie d'un système. De plus, cela peut être fait selon différents niveaux de granularité (degré de détail et de précision), donc pour résoudre des problèmes à différents niveaux hiérarchiques. La simulation peut donc tenir compte des différents horizons temporels : long, moyen et court termes ainsi que le temps réel. La Figure 3.12 extraite de [Berchet 2000] résume ces différents champs de représentation couverts par la simulation.

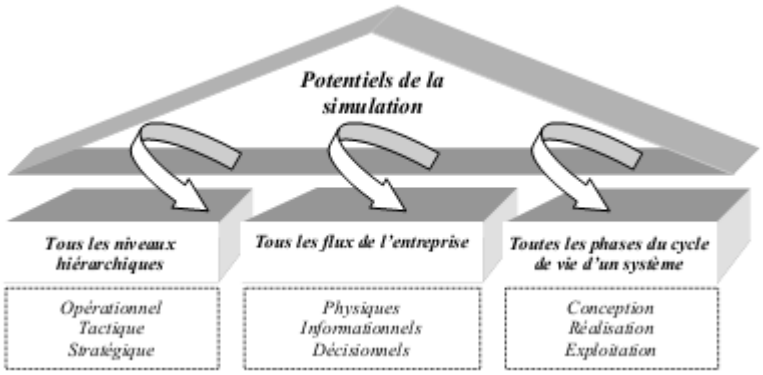

FIGURE 3.12 – Potentiels de la simulation selon [Berchet 2000]

Tous ces facteurs expliquent la popularité de la simulation en tant qu'outil pour évaluer la performance des systèmes de production ainsi que d'autres systèmes complexes. Cependant, d'autres potentiels ressortent si l'on considère son utilisation conjointe avec des méthodes d'apprentissage. Un tel couplage apparaît comme une approche très complémentaire pour la génération de connaissances [Pierreval et Ralambondrainy 1990, Fishwick 1991, Pierreval 1992a, Monostori *et al.* 2000, Huyet 2006]. D'après [Doukidis et Angelides 1994], malgré les progrès dans les technologies de construction des systèmes experts, le processus d'acquisition des connaissances des experts humains et de leur codage dans un système informatique reste difficile. Ici, les modèles de simulation peuvent jouer un rôle positif car ils peuvent être considérés comme des générateurs de données. Par la suite, nous nous concentrerons sur ce qui a été réalisé dans la littérature sur les différentes possibilités de couplage de ces techniques.

3.4 Usage conjoint de l'apprentissage et la simulation

3.4.1 Rôle de la simulation

Dans le cadre de cette thèse, nous nous sommes intéressés aux méthodes d'apprentissage (donc issues de l'intelligence artificielle) permettant d'extraire des règles à partir du comportement des systèmes de production en vue de leur adaptation. Nous avons pu constater qu'il y a un fort besoin d'exemples pour le bon fonctionnement de la plupart de ces méthodes (Section 3.2.2). Dans ce contexte, [Mouelhi-Chibani 2009] souligne que plusieurs approches ont été proposées dans le domaine du pilotage des systèmes de production pour constituer des ensembles d'apprentissage à partir desquels il serait possible apprendre, à savoir : des méthodes d'échantillonnage [Pierreval et Ralambondrainy 1990, Li *et al.* 2007], une connexion avec un algorithme d'optimisation [Huyet et Paris 2004] ou encore des exemples provenant de pratiques des opérateurs de systèmes réels [Zhao et De Souza 2001]. C'est aussi dans ce contexte que la simulation est apparue comme une solution intéressante à l'application des méthodes d'apprentissage lorsque les exemples viennent à manquer.

D'après [Doukidis et Angelides 1994], le couplage de l'intelligence artificielle et de la simulation peut être mutuellement utile, ce qui est renforcé plus précisément dans le cas des systèmes experts par [Fishwick 1991]. Citons que les techniques d'IA peuvent aider à modéliser le processus décisionnel dans la simulation [Egdorf et Roberts 1988]. De même, l'IA peut tirer profit d'une telle intégration, notamment vis-à-vis des systèmes experts : la conception de ces derniers peut être améliorée par l'utilisation de la simulation pour l'acquisition de connaissances et l'évaluation de bases de connaissances [Doukidis et Angelides 1994].

De fortes similarités méthodologiques entre les domaines de l'IA et de la simulation ont inspiré des chercheurs à proposer des taxonomies permettant de mieux exploiter leur couplage. Ces propositions étant trop génériques et/ou éloignées du sujet de cette thèse, nous n'en avons gardé qu'un aperçu dans l'Annexe A.

En revanche, d'autres s'intéressent plus précisément à la combinaison de l'apprentissage et de la simulation. [Monostori *et al.* 2000] signalent par exemple que l'apprentissage peut améliorer la performance de différentes formes de couplage entre un système basé sur la connaissance et la simulation. En effet, l'apprentissage donne à ce type de systèmes la capacité de mettre à jour leur base de connaissances [Ören 1994]. Les auteurs signalent d'autre part que la simulation peut servir à générer des exemples pour l'apprentissage. À ce propos et d'après [Frawley 1989], l'apprentissage est basé sur l'expérience et, malgré les différentes classes d'algorithme, il dépend crucialement de la simulation pour lui fournir cette expérience.

Ce même auteur explique plus précisément que l'apprentissage est basé sur l'expérience

fournie par l'interaction entre le « monde » et l'apprenant qui cherche à améliorer sa performance lors de l'exécution d'une tâche. Cela requiert un environnement comprenant des situations à partir desquelles il pourra apprendre et d'autres avec lesquelles il pourra tester l'état de l'amélioration basée sur l'apprentissage. Cet environnement doit être à la fois actif, passif, purement observable, contrôlable, etc., ce qui est impossible pour un environnement réel. Les modèles de simulation pourraient ainsi y jouer un rôle positif. Toujours selon [Frawley 1989], ils sont même nécessaires et les travaux en apprentissage font usage d'environnements et de situations simulés (spécifiques), la simulation étant ainsi au cœur de la recherche en apprentissage. Nous le verrons dans ce qui suit, le simulateur joue le rôle du monde, ses états correspondant à des situations rencontrées. Par ailleurs, les actions de l'apprenant changent des aspects du simulateur. L'évaluation de ces interactions est à l'origine du changement de comportement de l'apprenant de façon à ce qu'il fasse mieux lorsqu'il sera confronté à une situation identique ou similaire.

Citons encore [Ören 1994] qui identifie l'apprentissage par simulation comme étant un domaine de recherche prometteur où la simulation est utilisée en tant que technique de génération de connaissances empiriques à base de modèles pour l'apprentissage. Pour cela, des modules peuvent concevoir, exécuter et surveiller les expérimentations de simulation alors que d'autres vont interpréter les résultats pour en apprendre davantage sur la structure ou le comportement du système étudié. Cette voie de recherche constitue par ailleurs l'essence même de cette thèse.

Nous présenterons par la suite les travaux qui suivent cette perspective où l'apprentissage est combiné à la simulation. Nous nous concentrerons sur ce qui a été fait dans le domaine des systèmes de production.

3.4.2 Travaux de la littérature appliqués aux systèmes de production

L'utilisation de l'IA dans la gestion de la production est largement rapportée [Monostori 2003, Liao 2005, Sahin *et al.* 2012] et des approches la combinant avec la simulation sont soulevées [Pierreval et Ralambondrainy 1990, Fishwick 1991, Pierreval 1992a, Pierreval 1992b, Monostori *et al.* 2000, Huyet 2006]. Comme identifié par [Mouelhi-Chibani 2009], il existe plusieurs travaux sur l'apprentissage à partir de la simulation appliqués aux systèmes de production puisque celle-ci est capable de fournir des informations extrêmement utiles sur leur comportement.

3.4.2.1 Méta-modèles

Il est reconnu que les temps de calcul des approches basées sur la simulation ne sont en général pas négligeables [Law 2014]. À ce propos, de nombreux chercheurs ont utilisé l'apprentissage pour construire des méta-modèles qui représentent les systèmes de pro-

duction dans le but de remplacer le modèle de simulation. Un modèle de simulation est une représentation simplifiée (ou abstraction) de la réalité (dans notre cas un système de production) développée selon des objectifs précis. Ce processus d'abstraction peut être appliqué au modèle en lui-même, de sorte qu'un modèle simplifié du modèle de simulation soit obtenu. C'est ce deuxième niveau d'abstraction de la réalité qui est appelé méta-modèle [Kleijnen 2015]. Ainsi, un méta-modèle est capable de prédire les sorties du modèle de simulation d'un système de production à partir de ses entrées.

Malgré l'étude comparative réalisée par [Can et Heavey 2012] montrant la supériorité de la programmation génétique face aux réseaux de neurones pour la construction de méta-modèles, la méthode la plus couramment utilisée est basée sur ces derniers puisque ce sont des approximateurs universels parcimonieux [Kilmer *et al.* 1994, Dreyfus *et al.* 2004]. Ce potentiel est par exemple exploité par [Pierreval et Huntsinger 1992] dans le cadre d'un atelier *job-shop*, les auteurs montrant également par le modèle d'un percolateur à café la faisabilité de ce type de méthode pour des systèmes continus. Citons aussi [Fonseca et Navaresse 2002] qui en font usage pour évaluer les temps moyens de passage des commandes traitées simultanément dans un *job-shop* à quatre machines.

[LeCroy *et al.* 1996] et [Mollaghasemi *et al.* 1998] s'intéressent quant à eux à une application réelle impliquant des opérations de test d'une usine de fabrication de semi-conducteurs. Un méta-modèle est développé de façon à répondre à des questions dites inversées, c'est-à-dire pour estimer les entrées nécessaires à l'obtention d'un résultat spécifique. Leur but est de constituer un système d'aide à la décision pour la conception de systèmes de production. C'est un méta-modèle inversé où la sortie de la simulation (le temps de cycle et les encours) est l'entrée du méta-modèle, et l'entrée (la règle de priorité et les nombres nécessaires pour trois types de testeurs) correspond à la sortie désirée.

Intéressés par le modèle de stocks d'un atelier, [Kilmer *et al.* 1994, Kilmer *et al.* 1999] utilisent un méta-modèle pour estimer le coût moyen total de stockage et un autre pour estimer sa variance. Les systèmes à flux tiré ont aussi retenu l'attention des chercheurs. Nous pouvons citer les travaux de [Guneri *et al.* 2009] pour étudier les avantages d'un système kanban flexible par rapport à un système traditionnel (soit d'un nombre de cartes variable plutôt que fixe). Citons également un autre type d'application où des méta-modèles sont utilisés pour la réduction de modèles. C'est notamment le cas dans [Thomas *et al.* 2011, Thomas *et al.* 2014] qui exploitent pour cela des réseaux de neurones mais aussi des arbres de régression.

Toujours dans cet esprit, quelques travaux proposent des approches hybrides où ces méta-modèles basés sur les réseaux de neurones sont couplés à d'autres méthodes. Dans [Chan et Spedding 2001], le couplage est fait avec une approche de surfaces de réponse pour prédire la performance d'un système d'assemblage de produits optoélectroniques et ainsi trouver la configuration optimale du processus. [Wang *et al.* 2005] développent quant à eux un modèle hybride de découverte de connaissances qui combine un arbre de décision et un réseau de neurones. Ce dernier est utilisé pour prédire la performance de la règle

de priorité appropriée à différentes situations et choisie par le premier en utilisant des données de production bruitées.

Le gain en temps de calcul est peut-être encore plus recherché lorsque des processus d'optimisation via simulation sont mis en place. Les méta-modèles peuvent ici encore être utiles et nous pouvons par exemple citer [Hurrion 1997] qui en tire profit dans un problème de recherche combinatoire. Le méta-modèle développé pour un système de production à flux tiré est utilisé conjointement à une méthode de recherche exhaustive pour trouver le nombre optimum de cartes kanban. De même, [Araz *et al.* 2006, Araz *et al.* 2008] en font usage pour assister le choix du nombre cartes. Pour cela, ils énumèrent les mesures de performance pour toutes les combinaisons possibles de nombre de cartes de sorte à fournir des estimations rapides pour une technique de décision multicritère.

Ce type d'approche, très utile lorsque les caractéristiques d'un système restent stables, aide à mieux maîtriser les aspects liés à sa conception. Cependant, son utilisation est limitée lorsque les conditions dans lesquelles le système opère sont susceptibles de changer, et ce de façon imprévisible. Il faut alors être capable d'adapter le système aux nouvelles conditions.

3.4.2.2 Outils d'aide à la décision

D'autres combinaisons à exploiter pourraient se montrer très utiles. Par ailleurs, l'état de l'art réalisé par [Mouelhi *et al.* 2007] sur l'utilisation des réseaux de neurones en simulation en donne un aperçu puisqu'il pourrait être élargi à d'autres méthodes d'apprentissage. Il démontre qu'il reste d'autres possibilités même si la construction de méta-modèles reste la finalité la plus récurrente. Ainsi, les exemples de bonnes ou de mauvaises décisions étant difficiles à trouver de façon suffisamment variée dans le monde réel, plusieurs auteurs ont montré que la simulation était capable d'en fournir. Dans ces situations, nous sommes sur une utilisation des méthodes d'apprentissage non pas pour remplacer la simulation mais pour piloter le système (et donc ses adaptations). C'est dans cet esprit que [Pierreval et Ralambondrainy 1990] et [Pierreval 1992a] mettent en évidence l'intérêt de disposer de connaissances qui soient présentées notamment sous la forme de règles de production. Ces dernières, dominantes en intelligence artificielle et souvent appelées règles situation-action ou condition-action, sont des règles du type « si-alors » : lorsqu'une certaine situation/condition (détaillée dans le « si ») est rencontrée/satisfaite, les actions associées (dans l'« alors ») sont exécutées [Doukidis et Angelides 1994, Wilson 1998, Frécon et Kazar 2009]. Les connaissances sont ainsi directement exploitables par un système expert ou par un décideur.

Les réseaux de neurones ou autres méthodes d'apprentissage ont été peu utilisés pour le pilotage d'atelier en tant que « décideur » jouant un rôle actif dans un système d'aide à la décision. Cependant, certains chercheurs ont décidé dans ce but d'exploiter différemment les données obtenues par simulation pour la constitution d'un ensemble d'apprentissage.

La Figure 3.13 montre la différence entre une utilisation comme méta-modèle ou décideur. Dans le deuxième cas qui nous intéresse plus particulièrement, le résultat de l'apprentissage est un module pour la prise de décisions dans le modèle de simulation. Ceci est matérialisé soit par la proposition de décisions possibles aux décideurs afin de les assister au mieux dans pilotage du système, soit directement par le choix de la décision censée optimiser la performance visée dans le système de production.

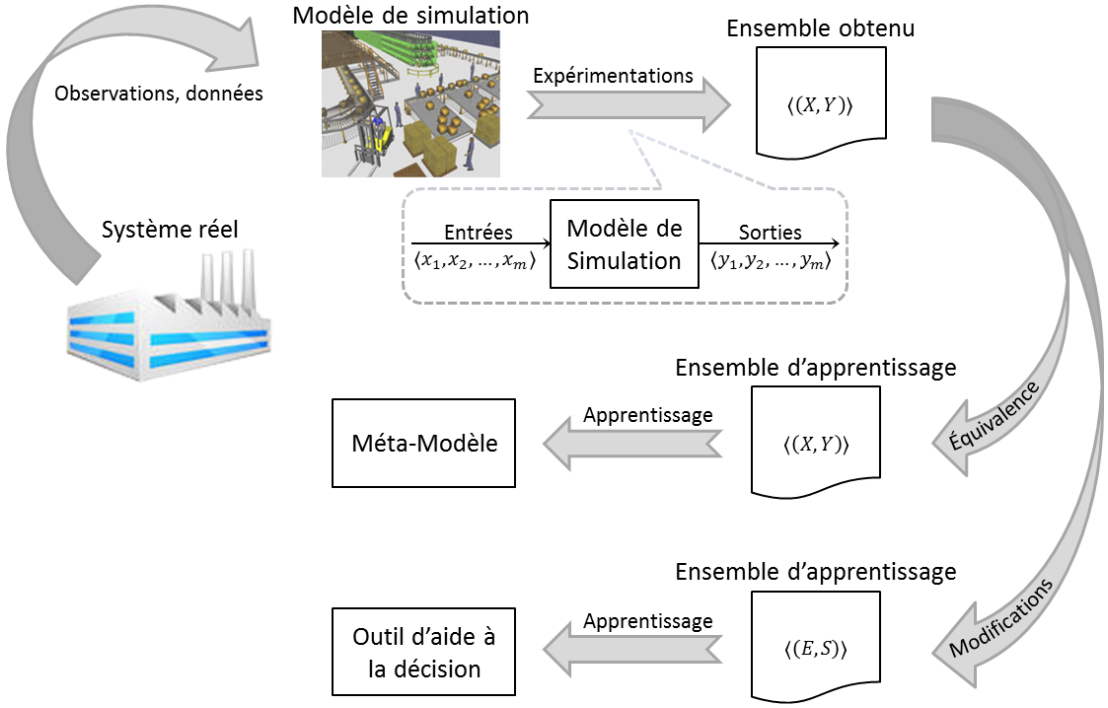

FIGURE 3.13 – Rôle de l'apprentissage selon l'usage des expérimentations de simulation

Dans le premier cas, l'apprenant va reproduire le comportement du modèle de simulation et nous indiquer par exemple la performance du système selon la configuration et la règle de priorité qui lui sont fournies en entrée. Dans le deuxième cas, l'ensemble d'apprentissage va être réorganisé de sorte qu'il contienne la règle de priorité qui a fourni des meilleures performances pour une configuration donnée. L'apprenant ne sera donc pas en mesure de reproduire le comportement du système simulé mais plutôt de suggérer la règle de priorité à mettre en place selon la configuration qui lui a été fournie en entrée.

Les propositions de [Pierreval 1992b, Piramuthu *et al.* 1993, Caskey 2001, Yildirim *et al.* 2006] décrites dans la Section 1.3.2.6 se placent dans ce contexte. Reprenons-en quelques-unes de façon succincte. Dans [Yildirim *et al.* 2006] par exemple, les données de la simulation sont transformées de sorte que le réseau de neurones indique le nombre de machines sur chaque station suivant ce qui lui est fourni en entrée, à savoir : la règle de priorité, le coefficient pour le calcul des dates de livraison ainsi que la performance souhaitée. Pour pouvoir fixer la valeur des deux premiers paramètres, la simulation est de nouveau utilisée mais cette fois pour un nombre bien plus limité de solutions.

[Pierreval 1992b] crée un ensemble d'apprentissage modifié à partir des données simulées pour entraîner un réseau de neurones à déterminer la règle de priorité à prendre en compte pour chaque station de travail selon l'état du système, résultat fourni par un classement. Il ne montre pas son utilisation pour un pilotage dynamique en temps réel mais signale que le réseau de neurones trouvé peut être utilisé soit comme un outil d'aide à la planification, soit intégré dans un logiciel de gestion de la production.

Nous pouvons encore citer les travaux d'[El-Bouri et Shah 2006], très proches de ceux de [Pierreval 1992b]. Ils considèrent que le réseau de neurones est construit selon un critère de performance donné. Ils en entraînent donc un pour minimiser le temps total d'exécution (*makespan*) et un autre pour minimiser le temps moyen de passage des pièces dans l'atelier (*flow time*). La façon dont ils structurent leurs réseaux de neurones permet de traiter des problèmes de plus grande taille puisque pour P règles de priorité et M machines, ils ont $P \cdot M$ plutôt que les P^M sorties de [Pierreval 1992b]. Les résultats restent sous la forme d'un classement, mais il s'agit d'un classement de règles à appliquer pour chaque machine plutôt que de la combinaison à appliquer directement. Cela permet d'évaluer la préférence relative de chaque règle en dépit d'une interprétation plus complexe.

Le méta-modèle inversé de [LeCroy *et al.* 1996, Mollaghasemi *et al.* 1998] présenté auparavant peut aussi être placé dans cette catégorie puisqu'il aide à la décision, bien que cela soit fait de façon statique. C'est un cas très particulier où la transformation de l'ensemble d'apprentissage consiste tout simplement à inverser les entrées et les sorties.

Dans ce même esprit d'aide à la conception, plusieurs chercheurs s'intéressent non pas à assister dans le choix d'une règle de priorité parmi les différentes existantes mais plutôt à la génération d'une nouvelle règle. Cela est fait à partir de différents attributs des produits ainsi qu'à partir de variables et paramètres du système, tous considérés comme significatifs vis-à-vis du problème. Citons [Geiger *et al.* 2006], [Geiger et Uzsoy 2008], [Pickardt *et al.* 2010] ou encore [Pickardt *et al.* 2013], tous utilisant la programmation génétique.

Ces méthodes où les décisions sont considérées statiquement se basent le plus souvent sur le comportement du système à l'état stable. À cet égard et pour mieux prendre en compte des modifications dans l'état du système, certains prennent des décisions périodiquement. À la fin de chaque période de temps Δt (déterminée au préalable), les valeurs des variables de décision à appliquer dans l'atelier sont réévaluées : toutes les possibilités pour la période à venir sont testées et la meilleure suivant le critère de performance considéré est choisie. Ces réévaluations sont le plus souvent faites en re-simulant périodiquement le système tel que dans [Wu et Wysk 1989]. Cette approche présente deux inconvénients selon [Pierreval 1992a] :

- Son coût en temps de calcul en raison du nombre de simulations qui doivent être effectuées périodiquement ;
- Sa sensibilité à la période utilisée car la performance dépend de la période de temps entre deux simulations.

Ce principe est cependant retenu par plusieurs chercheurs qui utilisent l'apprentissage pour contourner le premier inconvénient. Par exemple, [Metan *et al.* 2010] utilisent une décomposition en périodes d'ordonnancement, chaque période correspondant à une donnée dans l'ensemble d'apprentissage à partir duquel un arbre de décision est construit. Ensuite les règles de priorité sont sélectionnées en ligne à partir de l'arbre. De même, dans le contexte des systèmes à flux tiré, [Wray *et al.* 1997] et [Markham *et al.* 1998] utilisent respectivement des réseaux de neurones et l'induction de règles pour déterminer périodiquement le nombre de cartes à utiliser. Pour ce faire, ils considèrent que des informations sont disponibles concernant la période à venir, à savoir des prévisions sur la demande, les temps de traitement des machines et l'approvisionnement. Ils les résument par des valeurs qualitatives : un niveau de variabilité haut ou bas. Le nombre de cartes appartient à un intervalle prédéfini et toutes les possibilités sont simulées pour trouver celle qui fournit un coût minimum, et ce pour divers scénarios d'atelier. L'ensemble d'apprentissage peut être décrit par les données de la période précédente et de celle à venir en entrée ainsi que le nombre de cartes correspondant en sortie. L'approche est alors sensible à la période utilisée mais aussi à l'usage de valeurs quantitatives d'une façon qualitative, ce qui limite les possibilités d'apprentissage.

[Manupati *et al.* 2013] développent une approche basée sur l'apprentissage par SVM pour déterminer périodiquement la règle de priorité à appliquer dans une cellule de production flexible. La simulation est utilisée d'une part pour la génération des ensembles d'apprentissage et d'autre part pour évaluer le contrôle par SVM. Pour la constitution des exemples, la performance obtenue pour une période de simulation est associée à une paire <règle de priorité, état du système> qui correspond à la règle utilisée tout au long de la simulation et à l'état initial du système. Les différents états intermédiaires au cours de la période ne sont donc pas pris en compte.

Des changements et/ou perturbations survenant en temps réel ne peuvent toujours pas être convenablement pris en compte par la décomposition du temps en périodes. Pour remédier à ce problème, des chercheurs visent la prise de décision dynamique dans l'atelier lorsque c'est nécessaire. Ceci peut être fait en fixant des seuils de contrôle pour différentes variables à surveiller. Une autre possibilité est la détermination d'événements déclencheurs tels que l'arrivée d'une nouvelle commande ou encore la libération ou la panne d'une machine. Une fois ces seuils atteints ou lors de l'occurrence de ces événements, un nouveau choix est fait concernant les variables de décision.

[Pierreval 1992a] propose alors l'utilisation d'un système expert comme une alternative aux approches décomposant le temps en périodes. La simulation est utilisée pour tester et améliorer les bases de connaissances au sein d'un système expert. Ce dernier a pour but de changer dynamiquement les règles de priorité à appliquer selon l'état du système, les éventuelles contraintes et les critères de performance ciblés. Les décisions sont prises à chaque fois qu'une machine finit une tâche, donc lors de l'occurrence d'un événement déclencheur. Nous avons ici un usage de la simulation non pas pour concevoir un ensemble d'apprentissage mais pour assister un expert. Ce dernier doit interpréter les résultats de

simulation et modifier la base de connaissances en conséquence. Malgré cette intervention humaine, ceci reste une contribution possible de la simulation pour concevoir des systèmes experts plus performants étant donnée la possibilité d'en évaluer des différents et de les comparer.

[Lee 2008] développe une approche par apprentissage pour résoudre un problème d'ordonnancement dit adaptatif : la règle de priorité la plus adéquate est choisie et appliquée dynamiquement selon l'état courant de l'environnement d'ordonnancement. Selon lui, ce problème est généralement considéré comme une tâche de classification : la performance du système qui le résout dépend de l'effectivité de la connaissance utilisée pour lier les états du système aux meilleures règles pour ces états. Une base de règles floues est construite à partir d'un ensemble d'apprentissage issu de la simulation. Des règles floues sont préférées aux règles de production car la robustesse de la solution sera ainsi moins vulnérable aux changements de situation et donc non liée à la résolution d'un problème isolé. Pour construire l'ensemble d'apprentissage, une même règle donnée est appliquée tout au long d'une simulation lorsque la décision coïncide avec un état flou donné. Cependant, d'autres états différents peuvent également apparaître lors des décisions qui sont dans ces cas prises selon une règle par défaut définie par l'utilisateur. La performance associée à l'application d'une règle pour un état n'est alors pas complètement maîtrisée étant donné que les règles par défaut influencent également cette performance.

Une autre approche dynamique est proposée par [Takahashi et Nakamura 1999a] avec des réseaux de neurones. L'idée est de détecter des changements dans une série chronologique caractérisant la demande afin d'ajuster la taille du stock tampon dans un système à flux tiré. L'ensemble d'apprentissage est généré par des simulations faites sur des conditions stables. Par conséquent, leur approche se base sur l'échantillon de séries chronologiques utilisé. De plus, le comportement en temps réel dans un environnement fortement changeant ne peut pas véritablement être déduit à partir d'informations recueillies dans des conditions stables.

Une nouvelle idée d'apprendre sans ensemble d'apprentissage a depuis émergé. Elle vise à tirer profit de la synergie entre simulation et apprentissage pour le pilotage en temps réel tout en palliant les déficiences que nous avons pu apercevoir jusque-là. Cette idée est exploitée par [Mouelhi-Chibani 2009, Mouelhi-Chibani et Pierreval 2010] qui y associent le concept d'apprentissage autonome [Mouelhi et Pierreval 2007]. Leur approche consiste à entraîner un réseau de neurones directement à partir d'un modèle de simulation. La mise à jour (apprentissage) du réseau de neurones est supportée par un module d'optimisation.

Cela est aussi fait par [Kapanoglu et Alikalfa 2011] mais cette fois-ci en utilisant un algorithme génétique. Ils exploitent l'idée d'associer des règles de priorité connues à différents états du système. Pour caractériser un état, ils se concentrent uniquement sur les longueurs des files d'attente, réparties en intervalles consécutifs et sans différentiation par machine. Chaque machine composant un *job-shop* contribue à la construction d'une base de règles. Cette dernière est commune à toutes les machines de ce *job-shop* mais

utilisée indépendamment par chacune d'entre elles. À chaque libération d'une machine, celle-ci vérifie la règle de priorité à utiliser pour sélectionner la prochaine pièce à traiter, et ce selon le nombre de pièces dans sa propre file d'attente. L'approche impose une règle commune à toutes les machines plutôt que des listes individuelles et indépendantes pour chacune d'entre elles qui pourraient être plus spécifiques. De même, elle néglige toutes les autres variables caractérisant l'état d'un système. Leur prise en compte semble difficile avec la représentation utilisée de par leur aspect combinatoire.

Cette idée est également exploitée dans quelques travaux cités auparavant dans le cadre de décisions à caractère statique. La programmation génétique est utilisée par [Geiger *et al.* 2006] pour découvrir des nouvelles règles de priorité pour un environnement donné, et ce dans le cas de problèmes d'ordonnancement à une seule machine. L'idée est de trouver, hors ligne, une expression mathématique qui indique la priorité des tâches. Pour l'apprentissage, les auteurs utilisent la performance moyenne de chaque règle sur différentes instances représentant un même scénario, cette performance étant obtenue par des simulations. Puis, des nouvelles instances sont utilisées pour valider le résultat. Ils montrent comment adapter la méthode aux problèmes avec plusieurs machines, chacune ayant sa propre règle de priorité.

Le même type d'approche, encore une fois pour le problème à une seule machine, est appliqué par [Dimopoulos et Zalzala 2001]. Ils se concentrent sur l'ordonnancement statique lorsque l'objectif est de minimiser le retard total. Ils affirment cependant que l'approche peut être utilisée pour d'autres critères de performance. [Pickardt *et al.* 2010] s'intéressent quant à eux à des règles de priorité et à la formation de lots dans un environnement complexe. Ils récupèrent un certain nombre de règles (les meilleures pour chaque génération) qui sont simulées de nouveau pendant une période de temps plus longue pour le choix final. Leur approche est par la suite complétée par un algorithme génétique dans [Pickardt *et al.* 2013] afin d'affecter les règles aux différentes machines. Tous ces travaux sont réalisés du point de vue de la conception : ils ne font rien évoluer au cours du temps mais découvrent la règle à utiliser selon les caractéristiques de l'atelier et ce, de façon « définitive ». En revanche, [Pickardt *et al.* 2010] signalent une voie de recherche où l'apprentissage pourrait s'exécuter en continu pendant la production en temps réel, adaptant constamment les règles à l'environnement.

3.4.2.3 Autres

D'autres types de travaux exploitant le lien entre la simulation et l'apprentissage ont aussi été menés. Par exemple, [Huyet 2006] propose une méthodologie basée sur la synergie entre une approche d'optimisation évolutionniste et une approche d'apprentissage par des graphes d'induction. Elle vise à extraire des connaissances sur le comportement du système à partir des meilleures solutions proposées. L'idée est de fournir des explications sur les solutions obtenues qui soient utiles à leur conception et exploitation. L'interaction entre différents facteurs et leurs impacts dans la performance pourraient mener à

l'identification de facteurs critiques et à une possible caractérisation des solutions ayant de bonnes performances. Appliquée à un *job-shop* à cinq machines, la méthodologie reste une aide à la conception puisque les paramètres sont définis statiquement et non pas ajustés dynamiquement selon les changements dans les conditions d'opération.

[Dudas *et al.* 2011] analysent aussi des systèmes de production pour avoir un aperçu des paramètres qui influent sur les mesures de performance et obtenir une meilleure compréhension du problème. Pour ce faire, ils combinent un algorithme de type NSGA-II et la technique de fouille de données, tous les deux appliqués à des modèles de simulation. La simulation sert alors à évaluer les solutions pendant le processus d'optimisation et ainsi à constituer les données devant être exploitées par l'apprentissage. La combinaison entre simulation, optimisation et fouille de données est aussi exploitée par [Shahzad 2011, Shahzad et Mebarki 2012] qui se concentrent sur l'ordonnancement dans un *job-shop*, la performance étant liée au retard. Ils utilisent la recherche tabou pour l'optimisation et l'arbre de décision comme forme de représentation de connaissance (résultat de l'apprentissage). [Otto et Otto 2014] critiquent cependant ces méthodes utilisant la fouille de données car elles sont coûteuses en temps de calcul. Ils proposent à la place de formuler empiriquement des principes génériques de conception de règles de priorité testées par des expérimentations et validées par des tests statistiques. Il n'est cependant pas toujours possible d'appliquer une telle méthode.

D'autres auteurs dans la littérature s'intéressent aussi au couplage de la simulation et de l'apprentissage. [Bonneau et Proth 1985] proposent deux méthodes de regroupement pour classer les résultats obtenus par simulation et ainsi aider à choisir les règles de priorité pour un système de production. [Heritier-Pingeon 1991] exploite des données de simulation par l'analyse en composantes principales et la classification pour aider à la conception de systèmes de production. Citons enfin [Lereno *et al.* 2001, Chebel-Morello *et al.* 2006] qui construisent des arbres de décision (et même un réseau de neurones pour les premiers auteurs) à partir des résultats de simulation afin d'aider indirectement à la décision d'ordonnancement en assistant au paramétrage du logiciel de support de cette activité.

3.5 Conclusion

Nous avons commencé ce chapitre par une brève présentation des méthodes d'apprentissage les plus connues. Nous avons ensuite vu que la simulation est une solution largement utilisée lors de l'application de ces méthodes, étant donnée l'importance des exemples qui ne sont malheureusement pas toujours disponibles. Nous avons donc mis l'accent sur les travaux de la littérature utilisant à la fois les méthodes d'apprentissage et la simulation dans le domaine des systèmes de production.

Sur le plan théorique, différentes façons possibles de mettre en place une telle combi-

naison des modules d'apprentissage à la simulation pourraient être envisagées. Sur le plan pratique en revanche, nous pouvons tout d'abord constater que ces différentes possibilités ne sont pas forcément exploitées. Il s'avère que le rôle de la simulation reste « passif » dans la plupart des cas et mériterait donc plus d'attention. Les recherches se sont surtout concentrées sur l'usage de la simulation pour la génération d'exemples pour l'apprentissage. En effet, des exemples de l'effet des décisions en production étant difficiles à trouver de façon suffisamment variée dans le monde réel, plusieurs auteurs ont cherché à montrer que la simulation pourrait s'y substituer. Ils ont ainsi montré que cet outil est capable de fournir de tels exemples, nécessaires à la construction d'ensembles d'apprentissage. Pour cela, ils ont lié des stratégies de décision aux performances afin que des experts ou des outils d'apprentissage puissent en extraire les connaissances utiles et donc l'expertise recherchée.

La plupart des travaux se concentre sur des problèmes de conception où les décisions sont prises « une fois pour toutes », et ce en se basant sur des analyses du système dans son état stable. Ceci est risqué de nos jours compte tenu des conditions industrielles de plus en plus instables (marché, concurrence). D'autres travaux adoptent la stratégie de décomposition du temps en périodes. Cela permet déjà de prendre en compte l'apparition de quelques changements mais en restant très sensible à la durée établie pour une période. Les changements au cours d'une même période risquent d'être pris en compte tardivement et de se cumuler, voire de s'annuler entre eux. Avec de tels types d'approche, on peut trouver des logiques bonnes mais locales et instantanées : elles ne considèrent pas les objectifs dans le long terme et présentent donc très probablement une forme de myopie dite temporelle. Cette dernière est définie par [Zambrano Rey 2014, Zambrano Rey *et al.* 2014b] comme une limitation concernant l'estimation des conséquences à long terme des décisions courantes.

Une décision efficace est une décision qui est toujours valable au présent et que l'on réitère au besoin, donc dynamiquement. Les travaux dans cette optique ne sont pas nombreux et ont pour la plupart été accomplis à l'aide d'hypothèses et de simplifications pouvant nuire à la qualité des données pour l'apprentissage et, par conséquent, à la qualité du résultat.

En effet, les méthodes d'apprentissage sont très sensibles à la qualité de l'ensemble d'apprentissage : les résultats obtenus restent très dépendants des données à partir desquelles les connaissances sont extraites. Ainsi, leur usage nécessite que l'on soit capable non seulement de générer des exemples (ce qui peut effectivement être fait par simulation), mais surtout d'en générer des bons. Autrement dit, il faut suffisamment de maîtrise pour pouvoir distinguer des exemples de bonnes ou de mauvaises décisions au regard d'un objectif de performance. Or, lorsqu'il s'agit du pilotage des systèmes de production, l'impact sur la performance des décisions prises est difficile à évaluer immédiatement ou dans le très court terme. L'état de ces systèmes change dynamiquement au cours du temps ainsi que l'environnement dans lequel ils opèrent, ce qui doit être convenablement pris en compte lors des prises de décisions afin que celles-ci soient efficaces. Ceci devient encore

plus flagrant lorsque l'on parle de leur adaptation aux changements, ce qui nous intéresse plus particulièrement.

De plus, puisque plusieurs décisions sont prises et donc s'enchaînent au cours du temps, c'est cet enchaînement de décisions qui s'avère être performant ou non, et non pas chaque décision individuellement. C'est cet aspect dynamique du problème et des systèmes de production eux-mêmes qui rend difficile la tâche d'évaluation intrinsèque à la génération d'exemples pour l'apprentissage. En même temps, si la possibilité d'apprendre sans exemples d'apprentissage est attractive pour générer des connaissances relatives au comportement d'un atelier, la littérature ne rapporte à notre connaissance que très peu d'applications de telles approches dans le contexte des systèmes de production. Par ailleurs, les travaux existants se concentrent surtout sur des décisions dans le court terme, notamment sur les règles de priorité et les cartes kanban. Ce type d'approche devrait cependant pouvoir être appliqué à des décisions beaucoup plus vastes.

Bien que peu d'études aient été réalisées, nous avons pu voir que l'usage de la simulation peut aussi être une source de génération de nouvelles connaissances [Geiger *et al.* 2006, Mouelhi-Chibani 2009, Pickardt *et al.* 2010]. Les voies principales étant identifiées, nous sommes conduits à étudier l'apport de l'usage conjoint de la simulation et de l'apprentissage, évitant par ailleurs les nombreux inconvénients que nous avons soulignés. Autrement dit, notre travail vise à développer un principe d'apprentissage par simulation et non pas à partir d'un ensemble de données générées par simulation. Nos propositions ont vocation à être appliquées à des problèmes d'adaptation, donc à un spectre beaucoup plus large que le pilotage d'atelier. Par la suite, nous proposerons une formalisation du problème d'adaptation qui fait l'objet de cette thèse. Nous proposerons également une approche pour le résoudre, et ce dans le but de permettre une étude sur les contributions de ce type d'approche.

Chapitre 4

Apprentissage par Programmation Génétique Linéaire basée sur la Simulation pour l'Adaptation des Systèmes de Production

---

4.1 Introduction

De nos jours, la complexité et l'incertitude caractérisent l'environnement de nos systèmes de production. Ils ont donc besoin d'accomplir une large variété de tâches tout en étant adaptatifs. De ce fait, résoudre le problème sur la façon dont ils peuvent répondre à des changements continus dans un environnement dynamique est devenu un enjeu aussi bien pour les industriels que pour les membres de la communauté scientifique.

En effet, nous avons pu constater dans le Chapitre 1 le besoin croissant en termes d'aide à la décision dans les problématiques liées aux systèmes de production adaptables. Nous nous intéressons donc à la contribution que pourrait y apporter l'apprentissage par simulation. Ainsi, ce chapitre propose une approche qui permet de traiter des problèmes d'adaptation de natures diverses.

Différentes décisions interviennent tout au long du cycle de vie d'un système de production. Pour les traiter, nous pouvons tout d'abord distinguer les cas où l'on peut considérer qu'il existe une certaine visibilité sur le futur ainsi que la disponibilité des nombreuses informations nécessaires à la prise de décision. Ces données permettent de faire appel à des méthodes d'optimisation ou des logiciels de gestion de production par exemple. Ceci est même souhaitable, l'objectif étant de déterminer la meilleure décision possible au vu des critères de performance visés (coûts, délais) et en accord avec certaines contraintes (capacités, priorités). Nous avons pu identifier dans le Chapitre 1 des travaux qui déterminent « quand changer quoi », lesquels supposent la plupart du temps un horizon temporel, un plan de production et même des changements connus à l'avance. Or, ce n'est pas toujours le cas.

Nous pouvons ainsi distinguer les décisions qui sont davantage sujettes à la complexité et l'imprévisibilité de l'évolution dynamique du système [Coffman 1976, Pujo et Brun-

Picard 2002, Pujo et Kieffer 2002]. C'est sur ce contexte précis que nous allons nous intéresser plus particulièrement. Ces décisions doivent souvent être prises de façon empirique ou par des heuristiques qui peuvent être plus ou moins élaborées et dépendantes de l'état du système [Coffman 1976].

Il nous faut ici être capables de déterminer quand un système de production doit s'adapter et comment. Qu'elles soient de l'ordre du pilotage ou d'une adaptation plus profonde du système, ces décisions sont complexes et demandent de l'expertise. Or, cette dernière est très souvent insuffisante, voire même inexistante. Est-il alors possible de doter le système d'une forme de capacité à générer lui-même les connaissances nécessaires (et éventuellement les utiliser tout seul) ?

Nous avons pu identifier, notamment par le Chapitre 3, que l'apprentissage se présente comme un moyen possible pour aider à déterminer quand mettre en œuvre quelle(s) action(s) pour l'adaptation d'un système. Pour cela, il nous faut pouvoir associer les décisions que l'on souhaite assister à des mesures de performance. Or, dans le contexte de la production, les conséquences d'une décision ne sont pas facilement visibles et mesurables aussitôt celle-ci mise en place. Comment savoir sur-le-champ si augmenter le nombre de cartes circulant dans un système à flux tiré est une bonne décision ? De même, comment savoir si l'affectation d'un opérateur à une machine donnée parmi plusieurs possibles est la meilleure ? Ou encore si le moment est vraiment propice à des investissements (augmentation de capacité, nouvelles technologies) ?

Un système de production évolue avec le temps tout comme l'environnement qui l'entoure, mais nous ne pouvons prévoir comment. À ce jour, il n'existe que très peu d'exemples sur la façon dont une entreprise doit s'adapter selon les différents changements qui peuvent survenir. Bien que l'acquisition de connaissances soit possible en créant artificiellement (par simulation) des ensembles d'apprentissage, nous avons montré dans le chapitre précédant que ceci implique de nombreuses simplifications. Dans le cas contraire, il faudrait connaître le futur. De plus, signalons que les décisions sont dépendantes de l'évolution. Autrement dit, elles ne sont pas uniques mais liées entre elles selon ce qui va se passer sans que l'on puisse le savoir à l'avance : encore une fois, il faudrait connaître le futur. Il nous faut donc être capables de générer des connaissances relatives au fonctionnement d'un système de production où aucun ensemble d'apprentissage ne peut être conçu. Mais comment peut-on le faire ?

L'idée est d'apprendre à partir d'un modèle et plus précisément d'un modèle de simulation. Quelques travaux ont été identifiés dans cette direction (Chapitre 3). Toutefois, c'est une approche « alternative » qui n'a pas été très approfondie dans la littérature et qui n'a jusqu'alors ciblé que des problèmes précis de règles de priorité. Elle nous semble pourtant prometteuse. Mais une fois son potentiel étendu, quelle peut être sa contribution en tant qu'aide « intelligente » à l'adaptation des systèmes de production ?

Pour répondre aux questions évoquées précédemment, ce chapitre s'organise comme suit. Dans un premier temps, nous fournirons les principes généraux permettant de forma-

liser le problème d'adaptation aboutissant à un problème d'extraction de connaissances. L'approche de résolution sera donc introduite dans sa forme la plus générale. Une fois ces bases posées, nous pourrons choisir une méthode d'apprentissage adéquate. Puis, nous présenterons le codage d'une solution en accord avec la méthode choisie. Enfin, nous détaillerons le fonctionnement spécifique de notre approche vis-à-vis de la méthode ainsi que nos choix d'implémentation.

Pour faciliter la présentation, les deux sections qui suivent s'inspireront parfois du cadre conceptuel développé dans le Chapitre 2.

4.2 Principes généraux

Nous nous intéressons aux connaissances permettant d'adapter un système de production. Elles doivent se baser sur un certain nombre d'informations, lesquelles sont regroupées dans l'ensemble I défini lors du cadre conceptuel (Section 2.3.2.1). L'objectif est de satisfaire des critères de performance pré-établis dans l'ensemble G comme le temps de cycle, le taux d'utilisation de ressources ou le retard (Section 2.3.3.3). Contrairement à la plupart des travaux dans la littérature, nous ne nous restreignons pas au pilotage en temps réel : les adaptations peuvent se faire sur différents horizons temporels y compris le long terme. Cependant, nous ne le savons pas à l'avance puisqu'il n'y a pas de prévisions suffisantes. Nous ne pouvons donc pas nous baser sur des données futures. À chaque moment, il peut potentiellement y avoir des changements. Ainsi, pour identifier le moment où le système doit être remis en cause, nous devons scruter son évolution régulièrement (voire à tout instant), ce qui est fait selon une stratégie T de déclenchement du processus décisionnel (Section 2.3.2.2).

Pour ce faire, nous abordons le problème d'un point de vue événements discrets : l'état du système change seulement à certains instants précis, à savoir lors de l'occurrence d'événements particuliers appelés événements déclencheurs. Ils peuvent se produire plus ou moins souvent et être de différentes natures : l'arrivée d'un nouvel ordre de fabrication, la fin d'une opération sur une machine, le franchissement d'un seuil d'alerte, le début d'une grève, l'apparition d'une tendance, etc. Puisque l'état du système reste stable entre deux événements, les décisions peuvent être prises à ces instants, ce qui évite de décomposer le temps en périodes comme le font de nombreux travaux. Cela définit la stratégie de déclenchement T par une liste d'événements pertinents à surveiller. Cependant, rien n'empêche d'en utiliser une autre.

Il faut ici signaler que les états du système ne sont pas forcément dénombrables et connus à l'avance. Aucune hypothèse n'est faite là-dessus, c'est pourquoi l'espace d'états est un ensemble discret fini ou non. On ne se place donc pas dans un contexte de processus de Markov. Ce dernier, ayant des probabilités de transition d'un état vers l'autre, aurait permis l'utilisation d'approches du type optimisation dynamique par exemple.

Notre approche a comme principe de ne pas déterminer directement l'état souhaité (la finalité) pour le système à un instant t donné mais le moyen d'y arriver : les décisions à prendre représentent plutôt un choix parmi différentes actions possibles. Celles-ci sont déterminées en se basant sur les types de changements dans l'ensemble C et les valeurs dans $E_{j,k}$ pouvant prendre chaque attribut $a_{j,k}$ de chaque objet $o_j \in O$ (Section 2.3.1) : une action est un changement précis qui, appliqué à la valeur courante, en détermine la nouvelle. Autrement dit, une décision n'indique pas la nouvelle valeur de chaque attribut considéré mais l'adaptation spécifique à leur apporter, bien que ces deux notions puissent parfois se confondre. Si l'on s'intéresse au nombre de machines dans le système, les actions peuvent être par exemple un achat, une mise au rebut ou même ne rien faire. Elles peuvent être représentées par les entiers +1, -1 et 0 respectivement et l'état résultant (souhaité) du système dépendra de celui en cours. Si l'on s'intéresse par contre à l'affectation d'un opérateur, les actions « Réaffectation vers la Machine 1 » et « Réaffectation vers la Machine 2 » indiquent malgré tout très clairement la nouvelle valeur que prendra l'attribut, et ce indépendamment de l'état actuel. Dans tous les cas, il faut s'assurer que les actions soient valables (donc pertinentes) vis-à-vis de l'état actuel, ce qui peut être fait par des restrictions à priori et/ou des corrections à posteriori.

Nous nous concentrons donc sur les actions mais pas dans le but de choisir une action précise pour un moment précis. Nous cherchons plutôt à établir d'une façon plus générale comment choisir quelle action appliquer à quel moment. Ceci consiste à définir ce que nous appelons une logique décisionnelle sur la façon d'adapter un système, logique qui pourra ensuite être appliquée à tout instant t (la précision « décisionnelle » sera parfois omise par la suite).

Pour la plupart des actions préconisées, leur pertinence ne peut pas être connue dans l'immédiat. Il faut attendre leurs effets pour en évaluer les conséquences. Les performances doivent donc être considérées sur une période de temps suffisamment longue pendant laquelle il est rare qu'il n'y ait qu'une seule décision à évaluer. La performance n'est donc pas le résultat d'une action unique mais d'une séquence d'actions qui est quant à elle le résultat direct de l'application d'une logique décisionnelle.

Paradoxalement, cela accentue le problème d'une part mais le « résout » d'autre part : il est certes plus difficile de trouver des décisions performantes lorsqu'elles doivent être cohérentes avec celles passées et futures, mais l'évaluation ne serait pas adéquate autrement. Une logique décisionnelle combine la pertinence de chaque action individuellement et collectivement. Notons que les dernières décisions peuvent ne pas avoir eu le temps de voir leurs effets escomptés.

L'interaction entre le système de production et la logique décisionnelle (Figure 4.1) nous mène à un autre principe important de notre approche : celui de la décomposition d'un système en sous-systèmes complémentaires [Gourgand et Kellert 1991, Le Moigne 1994].

Nous optons ici pour que le système soit décomposé plus précisément en deux sous-

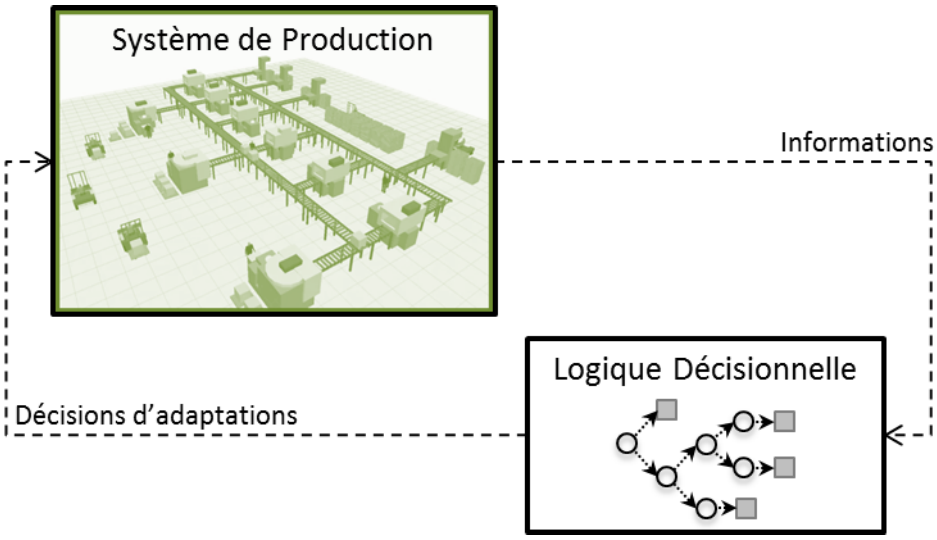

FIGURE 4.1 – Sous-systèmes physique et décisionnel

systèmes. D'une part, la composante physique représente le processus de production avec les ressources, les stocks, les produits ainsi que les flux logiques. D'autre part, la composante décisionnelle contrôle la précédente en lui communicant les décisions prises relativement à son pilotage et/ou son adaptation. Notons que ce contrôle n'est pas forcément direct et automatisé et peut correspondre tout simplement à des alertes envoyés aux gestionnaires qui décideront eux-mêmes quoi faire. Dans tous les cas, c'est cette dernière composante qui permet d'être réactif puisqu'elle correspond à la logique décisionnelle qui nous intéresse, donc au mécanisme de changement M (Section 2.3.3.2).

À cet égard, nous nous intéressons à trouver hors ligne une logique décisionnelle qui soit pertinente pour un système de production adaptable donné en utilisant une approche d'apprentissage. Nous utiliserons plus précisément la programmation génétique linéaire, un choix qui sera expliqué par la suite (Section 4.5). Pour faire face aux difficultés évoquées dans le chapitre précédent, nous proposons une approche qui ne nécessite pas d'ensemble d'apprentissage : elle apprend directement à partir d'un modèle de simulation à événements discrets, lui-même décomposé en deux sous-modèles complémentaires. Nous avons ici le principe utilisé pour trouver M. Une telle approche n'implique pas d'hypothèses restrictives particulières : les spécificités du système de production sont bien prises en compte grâce à la capacité des modèles de simulation à gérer des comportements stochastiques et complexes de manière assez réaliste.

Citons que la simulation permet de prendre en compte des fluctuations comme celles de la demande des clients, ce qui est important dans le contexte des systèmes de production adaptables. Par exemple, le taux moyen de la demande peut évoluer dans le temps selon différents types de *patterns* ou même aléatoirement. Cette évolution est matérialisée par un signal et la Figure 4.2 en donne quelques exemples. Chaque type de signal peut être fourni en entrée au modèle pour indiquer comment la demande moyenne évoluera durant toute la simulation. Cela sert à apprendre les comportements d'un système en cas de

changements et nous permet ainsi de traiter certains problèmes où les prévisions s'avèrent insuffisantes ou inexistantes.

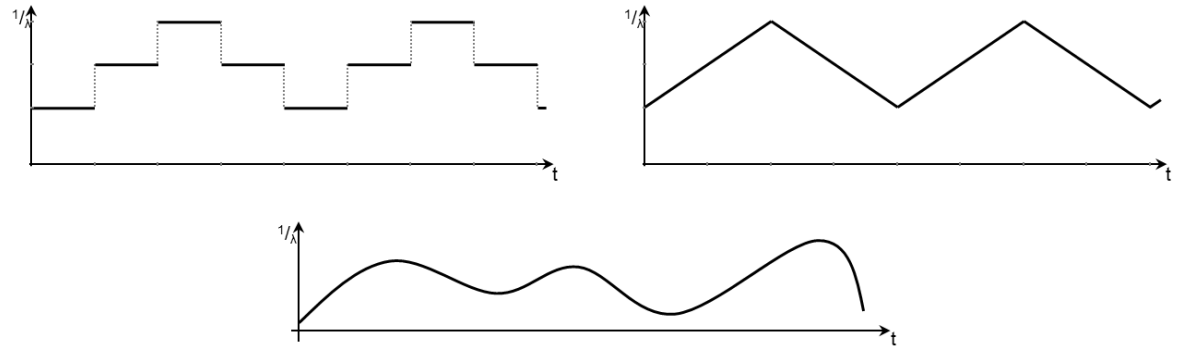

FIGURE 4.2 – Exemples de variation de la demande au cours du temps

Si l'on revient à l'idée d'apprendre, comment peut-on tirer profit de la simulation dans ce contexte ? Comment l'utiliser au mieux pour arriver à en extraire des connaissances « dynamiques » permettant de déterminer quand et quoi adapter selon un critère de performance donné ? Car l'objectif est soit de rendre ces connaissances accessibles en ligne aux gestionnaires de production pour le contrôle du système réel, soit de les utiliser dans un système de contrôle auto-adaptatif [Bousbia et Trentesaux 2002, Macías-Escrivá *et al.* 2013, Zambrano Rey *et al.* 2013].

Nous n'avons pas pour autant la prétention de pouvoir résoudre tous les problèmes d'adaptation avec une telle méthode d'apprentissage. D'une part, elle ne s'applique par partout car les aspects stochastique et incertain font qu'il existe des situations non « apprenables ». D'autre part, l'espace de recherche induit peut être considérable et donc représenter une barrière à l'obtention de bons résultats.

4.3 Formalisation mathématique du problème

Un certain degré de formalisation est nécessaire pour être capable de caractériser convenablement le problème d'adaptation traité ici. Celui-ci n'a à notre connaissance jamais fait l'objet d'une formalisation mathématique dans la littérature, c'est pourquoi nous allons maintenant en proposer une. Nous ferons encore référence aux composants du cadre conceptuel du Chapitre 2 bien que cette formalisation en soit détachée. Cette analogie sera aussi reprise de façon plus concrète lors des exemples d'application traités dans le chapitre suivant.

Nous nous intéressons à trouver une logique décisionnelle qui pourra être utilisée lors des instants t de prise de décision et qui concerne les N variables de décision d'un système. Pour mieux détecter les changements et/ou perturbations, ces instants sont définis selon l'occurrence d'événements déclencheurs. Puisqu'elle convertit des informations pertinentes

du système en décisions d'adaptation à mettre en place, cette logique décisionnelle peut être exprimée comme une fonction $\varphi \in \mathcal{L}$. Nous avons donc :

$$\begin{array}{rccl} \varphi : & \mathcal{I} = \mathcal{S} \cup \mathcal{P} & \longrightarrow & \mathcal{D} = D_1 \times D_2 \times \ldots \times D_N \\ & I = S_t \cup \mathcal{P} & \longmapsto & d_t = \langle d_{1,t}; d_{2,t}; \ldots; d_{N,t} \rangle \end{array} \tag{4.1}$$

où la notation utilisée peut être résumée comme suit :

- $t \in [0; H]$: Indice pour les dates des événements déclencheurs dont les valeurs peuvent être plus ou moins espacées les unes des autres, H étant la durée totale de la période d'étude ;
- $n \in \{1; \ldots; N\}$: Indice pour les variables de décision, N représentant le total de variables considérées ;
- $\mathcal{L}$: Ensemble de toutes les logiques décisionnelles possibles et applicables au problème à résoudre ;
- $\varphi \in \mathcal{L}$: Logique décisionnelle courante ;
- $\mathcal{S}$: Ensemble de toutes les informations utilisées pouvant changer dynamiquement (variables d'état, informations externes, indicateurs temporaires, historiques et prévisions) ;
- $S_t \in \mathcal{S}$: Vecteur d'informations caractérisant l'état du système et de son environnement à l'instant t ;
- $\mathcal{P}$: Ensemble de paramètres du système et de son environnement ;
- $\mathcal{I}$: Ensemble de toutes les descriptions possibles du système et de son environnement ;
- I : Description du système et de l'environnement où il opère à l'instant t, ce qui correspond à l'ensemble d'informations du cadre conceptuel (d'où la notation reprise) ;
- D_n : Ensemble de toutes les actions pouvant être appliquées à la variable n ;
- $d_{n,t} \in D_n$: Action choisie et donc à appliquer pour la variable n à l'instant t ;
- $\mathcal{D}$: Ensemble de toutes les actions possibles pouvant être appliquées à chaque variable du système ;
- $d_t \in \mathcal{D}$: Vecteur indiquant les actions sélectionnées pour les N variables du système à l'instant t.

Ainsi, φ est une fonction de l'ensemble $\mathcal{I}$ de toutes les valeurs possibles pour les informations utilisées dans le processus décisionnel vers l'ensemble $\mathcal{D}$ de toutes les actions pouvant avoir lieu dans le système. Les données d'entrée de cette fonction φ comprennent certaines composantes pouvant évoluer dans le temps (ensemble $\mathcal{S}$ instancié par le vecteur S_t) et d'autres non (ensemble $\mathcal{P}$). La sortie est quant à elle représentée par un vecteur d_t regroupant les éventuelles décisions d'adaptation devant être appliquées au système, comme la réaffectation d'un opérateur à une nouvelle tâche. Ces décisions peuvent être de différents types ; chaque composante $d_{n,t}$ concerne une variable de décision n et peut

prendre des valeurs dans un ensemble D_n fini. En somme et à partir des informations dans I, φ fournit les actions correspondantes à mettre en place.

Nous avons deux types de variables : les variables d'état et les variables de décision. Dans ce qui suit, nous apportons quelques précisions concernant ces variables et permettant de clarifier leurs similarités et différences.

Notons tout d'abord que les variables de décision définies ici font à elles seules référence aux types de changement des attributs variables des objets modifiables constituant le système de production en question. Leur indice n correspond à la fois aux indices $<j, k>$ du cadre conceptuel ($k^{ième}$ attribut variable du $j^{ième}$ objet modifiable). Chaque n peut être obtenu par l'Équation 4.2 ci-dessous :

$$n = \sum_{j'=0}^{j-1} K_{j'-1} + k \tag{4.2}$$

où K_0 est une donnée fictive valant 0. Cette équation est toujours valable car chaque attribut variable est « atomique », c'est-à-dire défini de façon à ce qu'un seul type de changement puisse le concerner.

Rappelons que la valeur courante d'un attribut variable fait partie des variables d'état et est donc potentiellement dans l'ensemble I. Ainsi, il y a bien un attribut associé à chaque variable de décision mais ils ne sont pas tout à fait les mêmes puisque nos variables de décision définissent des actions à mettre en place et non pas les valeurs finales que prendront les attributs associés. C'est la raison pour laquelle on ne retrouve pas directement $E_{j,k}$ dans D_n mais plutôt le type de changement $c_l \in C$ pour arriver de la valeur courante $a_{j,k,t}$ à une nouvelle valeur selon les possibilités offertes par $E_{j,k}$.

La définition du domaine d'appartenance des différents éléments dont nous nous servons pour notre formalisation n'est pas forcément connue puisqu'elle dépend directement du problème une fois concrétisé. En effet, nous pouvons facilement déduire que $t, H \in \mathbb{R}^+$ et $n, N \in \mathbb{N}^*$. Notons que pour l'indice t, une autre possibilité aurait été de le définir comme la $t^{ième}$ occurrence d'un événement déclencheur. Cela aurait discrétisé sa valeur ($t \in \mathbb{N}^*$), rendant alors impossible la définition d'une borne supérieure H puisque l'on ne connaît pas à l'avance le nombre d'événements que comportera une simulation.

Toujours à propos de t, notons également que cette formalisation peut facilement être adaptée à une autre stratégie de déclenchement T. Ainsi, l'indice t aurait pu faire référence à une stratégie périodique, auquel cas ses valeurs auraient été également espacées les unes des autres, la durée Δt définissant T. Dans ce cas, même une discrétisation permettrait de garder une borne supérieure qui serait équivalente à la durée de la simulation divisée par celle d'une période : $t \in \{1; \ldots; \lfloor \frac{H}{\Delta t} \rfloor\}$. Une autre possibilité aurait été de combiner les stratégies événementielle et périodique pour définir T où le traitement continu ($t \in \mathbb{R}^+$) s'avérerait plus pertinent. La liste d'événements et la durée Δt auraient été toutes les deux nécessaires pour définir T ainsi que la façon de considérer les périodes, indépendamment ou non des événements (revoir Figure 2.7, Section 2.3.2.2).

Revenons à la définition des domaines d'appartenance. Pour $d_{n,t}$, d_t, S_t, $\mathcal{P}$ et I, la tâche n'est pas aussi triviale. C'est pour cette raison que les domaines $D_n(\forall n)$, $\mathcal{D}$, $\mathcal{S}$, $\mathcal{P}$ et $\mathcal{I}$ sont « génériques ». Leur nature et leur dimension sont spécifiques à chacun des éléments du problème. Ils peuvent contenir, selon l'élément qu'ils concernent, des entiers et/ou réels mais aussi des éléments qualitatifs. De plus, chaque élément peut être un vecteur à une ou plusieurs dimensions.

Nous pouvons cependant faire la constatation suivante : la dimension d'un ensemble étant donnée par celle de l'élément qu'il représente, celle de $\mathcal{D}$ est donc connue et égale au nombre N de variables de décision. Ce dernier est quant à lui donné par la somme du nombre d'attributs variables K_j pour tous les objets $o_j \in O$, ce qui est résumé par l'Équation 4.3 qui découle de l'Équation 4.2 pour $j = J$ et $k = K_J$:

$$\dim \mathcal{D} = |d_t| = N = \sum_{j=1}^{J} K_j \tag{4.3}$$

Puisque nous avons une variable d'état associée à chaque variable de décision, nous pouvons nous attendre à avoir $|S_t| \geq N$. Cependant, ces variables d'état-ci ne font pas toujours partie du vecteur S_t qui regroupe uniquement celles effectivement considérées pour la prise de décision. De plus, bien qu'un historique ou des prévisions puissent être utilisés directement parmi les données d'entrée de la fonction φ, nous les considérons ici uniquement sous la forme d'indicateurs résumant chacune des informations dont l'évolution nous intéresse. La création de ces indicateurs temporaires facilite l'interprétation des données par φ qui n'aura pas à traiter les données brutes. Enfin, malgré le caractère fixe des paramètres du système, ils ne sont pas à négliger. Ils peuvent servir par exemple de seuils minimum ou maximum pour certaines variables d'état. Les paramètres permettent aussi d'utiliser une logique décisionnelle commune à des systèmes qui ne diffèrent que par la valeur de ceux-ci.

L'entrée de la fonction φ est ainsi un facteur-clé de sa performance. Les bonnes informations doivent être utilisées si l'on veut favoriser des décisions d'adaptation pertinentes. Sachant qu'il existe d'innombrables informations dans un système de production et l'environnement qui l'entoure, celles-ci peuvent être utiles comme « inutiles » vis-à-vis des décisions à prendre. Leur choix est donc étroitement lié aux types de décisions considérés et aux objectifs fixés. Que ce soit par le manque d'information ou par une complexification inutile induite par des éléments trop nombreux, les décisions puis les résultats en découlant s'en ressentiront.

Malheureusement, une fois les informations identifiées, la définition de φ n'est pas triviale. Nous ne nous intéressons pas à trouver une fonction φ quelconque mais celle qui optimise des critères de performance donnés. Même si nos décisions sont prises en surveillant des événements déclencheurs, le but est d'optimiser la performance sur le long terme. Cela réduit en partie la myopie temporelle associée à chaque décision individuelle [Zambrano Rey 2014, Zambrano Rey *et al.* 2014b].

Plusieurs critères de performance peuvent être utilisés et sont disponibles dans la littérature (se référer par exemple à [Burlat et Campagne 2001, Tahon 2003, Ducq 2007, Berrah 2013]). Soit f celui choisi pour évaluer le système ou $\vec{f}$ si l'on en considère plusieurs (cas multi-objectif). Dans ce qui suit, nous utiliserons cette dernière notation pour rester généralistes. La valeur de $\vec{f}$ dépend de la séquence de toutes les actions d_t choisies pour contrôler le système durant la période de temps considérée. Cette séquence est une conséquence directe de la logique décisionnelle φ en place puisque c'est φ qui définit comment convertir les informations en actions. Notons qu'en raison de l'existence d'aléas, l'évaluation doit en tenir compte et peut par exemple être faite en termes d'espérance $\mathbb{E}[\vec{f}]$. Ainsi, $\vec{f}(\varphi, \xi)$ dénote la performance du système utilisant la logique décisionnelle φ tout en étant soumis à des effets stochastiques ξ. Puisque nous ne disposons pas d'une fonction explicite pouvant être calculée analytiquement, cette performance est évaluée en utilisant une simulation de durée H. Après R réplications, la fonction objectif z est donnée par $z = \mathbb{E}[\vec{f}(\varphi, \xi)]$ qui dans notre cas est considérée comme la valeur *fitness* devant être minimisée (ou maximisée). Le problème d'optimisation (via simulation) correspondant peut alors être énoncé comme suit :

$$Minimiser \text{ ou } Maximiser \; z = \mathbb{E}[\vec{f}(\varphi, \xi)] \tag{4.4}$$

où les critères de performance $\vec{f}$ désignent les objectifs dans G, z précisant comment les évaluer.

Notons que nous venons d'identifier qu'il nous est possible de poser notre problème d'apprentissage par simulation sous forme d'un problème d'optimisation via simulation. En effet, la simulation possède ici plusieurs fonctions. Elle sert tout d'abord à mettre en pratique et évaluer une logique décisionnelle φ donnée. De par cette évaluation, elle sert également à guider le processus d'apprentissage, d'où l'analogie avec l'optimisation via simulation.

La performance atteinte est en fait le résultat du raisonnement qui induit les adaptations. Les N variables de décision sont alors considérées indirectement puisque nous ne cherchons pas à les définir à chaque instant t mais bien automatiquement pour l'ensemble des instants t. C'est la raison pour laquelle nous souhaitons optimiser notre sous-système décisionnel en utilisant la structure et les composantes de φ comme véritables variables de décision du problème. Comme introduit dans la section précédente, la logique décisionnelle φ sera déterminée en utilisant la programmation génétique linéaire basée sur la simulation dont les principes seront présentés en détails dans les sections suivantes. Celles-ci traiteront en premier lieu de notre approche d'une façon générale et s'étendront ensuite sur notre choix de méthode puis d'implémentation.

4.4 Résolution par hybridation de l'apprentissage et de la simulation

Une fois le problème formulé, intéressons-nous à comment le résoudre. Nous avons montré que nous cherchons une fonction φ qui nous donne des solutions d'adaptation optimisant des objectifs de production à partir des informations disponibles sur le système et son environnement. Plus précisément, nous cherchons φ optimisant leur valeur espérée puisque nous sommes souvent face à des facteurs stochastiques dans un système de production. Il nous faut donc une approche permettant de générer φ et/ou de la faire évoluer. Dans cette thèse, il s'agit du résultat de l'application d'une technique d'apprentissage.

D'après le Chapitre 3, apprendre correspond le plus souvent à trouver une « fonction » qui exprime au mieux un phénomène décrit par des exemples en mémorisant l'écart entre ce qui est prédit par la fonction et ce qui est attendu. C'est ainsi une classe de techniques qui par nature fonctionnent avec des exemples : habituellement elles font évoluer des fonctions afin de déterminer celle qui coïncide le mieux avec les exemples disponibles. Nous allons cependant l'utiliser autrement.

Nous avons vu dans la Section 3.3 que la simulation est un outil largement utilisé pour représenter la dynamique des systèmes de production et étudier des choix de configuration. Nous souhaitons aller au-delà de ces analyses statiques ou sporadiques en acquérant des connaissances utiles à l'auto-adaptation de ces systèmes.

Dans le cadre des systèmes de production adaptables, nous cherchons à définir une logique décisionnelle φ pouvant générer des alertes avec des propositions d'adaptation ou même être directement utilisée en tant que décideur. Pour ce faire, nous ne nous intéressons pas à une adaptation en particulier mais à ce qu'elle soit cohérente avec les décisions passées et futures. Mais comment peut-on générer les connaissances permettant de déterminer quelles actions sont nécessaires pour qu'un système maintienne de bonnes performances ou, encore mieux, qu'il cherche constamment à les améliorer ?

L'idée est de tirer profit du potentiel de la simulation et plus précisément d'apprendre itérativement à partir d'expérimentations menées par un modèle de simulation, et ce sans exemples. Avant de détailler le fonctionnement de notre approche, quelques considérations supplémentaires doivent être faites concernant les caractéristiques à remplir par la technique de simulation telle que nous souhaitons l'utiliser.

4.4.1 Simulation agile

Puisque nous nous intéressons aux systèmes de production adaptables, nous avons besoin d'une simulation dite « agile », capable d'appliquer des adaptations proposées sur le système simulé. Mais comment la faire changer de configuration ? Les logiciels de simula-

tion ne sont pas faits pour un tel usage. La structure d'un modèle est souvent considérée comme fixe puisqu'elle doit définir les relations entre les éléments le constituant lors de son développement [Hagendorf 2009]. Seuls quelques paramètres peuvent être modifiés automatiquement pour des fins d'optimisation. En revanche, lorsque la modification concerne la structure même du modèle, nous nous retrouvons souvent face à une tâche manuelle. Des travaux ont été faits pour résoudre en partie cette barrière. Citons [Hagendorf 2009] qui propose une approche pour la reconfiguration automatique de la structure des modèles de simulation, permettant ainsi qu'elle soit elle aussi optimisée. [El Haouzi *et al.* 2008] proposent quant à eux une approche pour construire des modèles de simulation modulaires et réutilisables à partir de composants génériques.

Nous verrons par la suite que dans notre implémentation, nous utilisons un logiciel de simulation disponible dans le marché. Il nous a fallu trouver la solution au problème d'une structure fixe parmi ses fonctionnalités. Ainsi, nous exploiterons la modularité d'un modèle plus précisément par la définition de sous-parties qui sont activées ou non selon les décisions prises. Cette activation peut être faite de deux façons :

- Ponctuellement, donc uniquement pour la mise en place de la décision. Le fonctionnement l'intègre ensuite automatiquement ;
- Jusqu'à la prochaine décision la remettant en cause, lorsque c'est la logique de fonctionnement qui change.

Cette solution est cependant coûteuse en temps de développement et en mémoire. Elle implique donc une limite aux adaptations pouvant être prises en compte.

4.4.2 Fonctionnement général

Notre approche se base sur deux techniques complémentaires : la simulation et l'apprentissage. Différents schémas seront utilisés pour mieux expliquer comment elles interagissent. Notons que nous donnerons d'abord les explications de façon assez générique sans faire référence à la technique d'apprentissage utilisée ici. Cela montre que rien n'empêche d'en utiliser une autre en suivant les mêmes principes.

Un système de production adaptable doit pouvoir gérer son processus intrinsèque d'adaptation. Pour ce faire, il nécessite ce que l'on appelle une logique décisionnelle qui regroupe des règles lui indiquant quand et comment s'adapter. Pour la mettre au point, il n'y a pas d'autre solution que de la tester. Puisqu'il serait trop coûteux de le faire sur le système réel, le recours à la simulation se présente comme une bonne alternative.

Nous nous servons ici du principe de décomposition du système en sous-système physique d'une part et sous-système décisionnel d'autre part (Section 4.2). Nous avons une logique décisionnelle φ à évaluer pour un système de production donné. Le système de production correspond à la composante physique du modèle de simulation tandis que la logique décisionnelle constitue, comme son nom l'indique, sa composante décisionnelle.

Elle sera alors utilisée tout au long d'une simulation et ce à chaque fois que le processus décisionnel est déclenché (selon la stratégie de déclenchement T).

Lorsqu'une décision doit être prise lors d'une simulation, le système de production fait appel à φ en lui communicant les informations I disponibles à l'instant t de la décision. Avec ces informations, φ détermine les éventuelles adaptations à faire sur le système, décisions qui sont retournées au système sous la forme d'un vecteur spécifique d'actions d_t. Les modifications correspondantes, s'il y en a, sont ensuite appliquées sur le système simulé. Puis la simulation suit son cours jusqu'au besoin d'une nouvelle décision relançant ce processus, et ainsi de suite jusqu'à ce qu'elle touche à sa fin. Nous pouvons alors connaître la performance $\vec{f}(\varphi, \xi)$ obtenue. Ceci est résumé par la Figure 4.3. Notons que cette interaction est similaire à celle de la Figure 4.1 à la différence qu'elle se passe au sein d'un environnement de simulation et pour une durée prédéterminée.

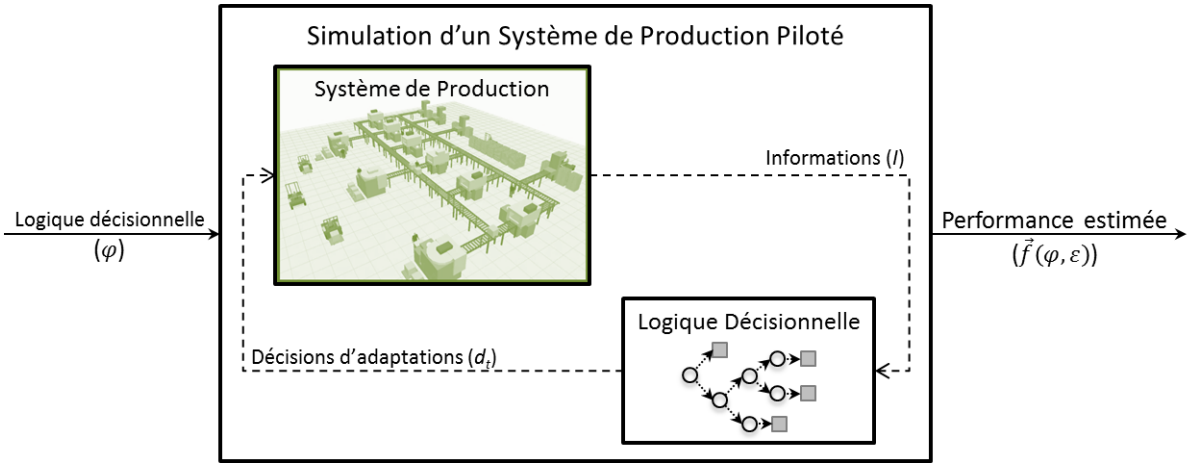

FIGURE 4.3 – Système simulé piloté par une logique décisionnelle

En raison de la nature stochastique du système, plusieurs réplications et/ou des longues périodes doivent être utilisées selon le type de simulation (terminant ou non) pour estimer $\mathbb{E}[\vec{f}(\varphi, \xi)]$ [Law 2014, Kleijnen 2015]. Ici, cette valeur correspond à la moyenne des performances obtenues sur chaque réplication d'une durée suffisamment longue. Elle est stockée comme le *fitness* de φ : si nous avons une bonne logique décisionnelle, alors les performances sur la période étudiée seront bonnes, sinon, les performances seront mauvaises. C'est la raison pour laquelle nous cherchons à optimiser φ. En ajoutant la notion de réplication au schéma précédent, nous obtenons la Figure 4.4.

Ce schéma suffirait si nous étions en possession des différentes logiques décisionnelles pouvant être utilisées pour piloter le système : nous l'appliquerions pour chacune d'entre elles pour en retenir que la meilleure. Ceci n'est généralement pas le cas. Alors comment construire et éventuellement améliorer une logique décisionnelle ?

C'est pour résoudre cette problématique que l'apprentissage intervient. Le moteur d'apprentissage propose une logique décisionnelle qui peut être sous une forme similaire à un programme informatique. Pour lui associer une performance, il a cependant besoin

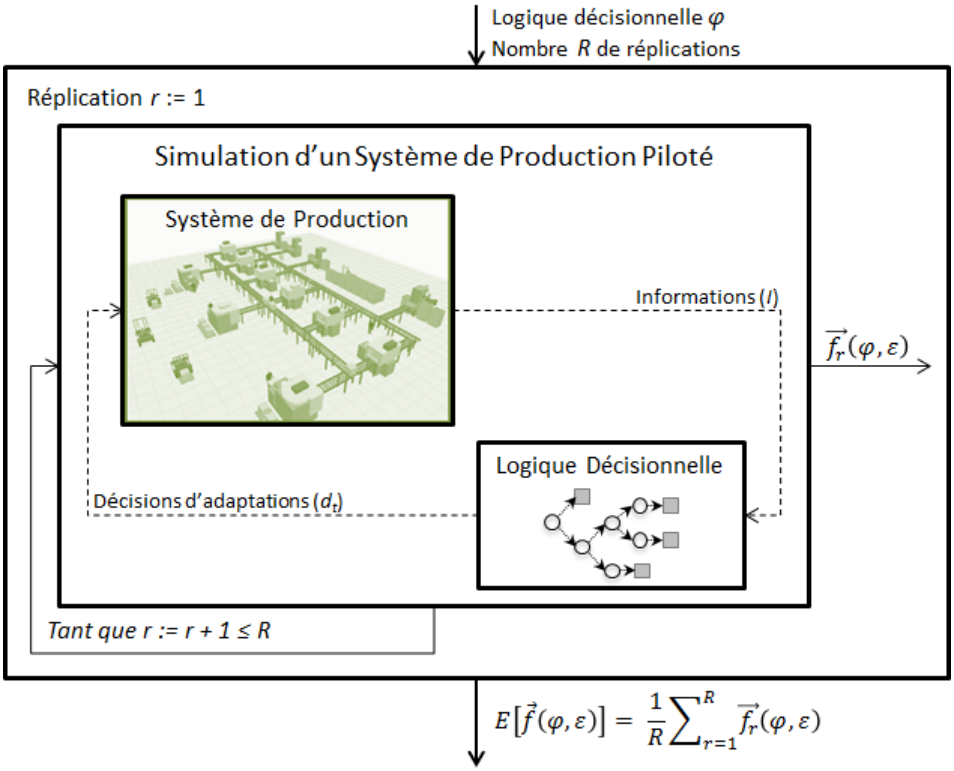

FIGURE 4.4 – Évaluation avec plusieurs réplications d'un système simulé piloté

d'un « évaluateur », fonction remplie par le module de simulation. La performance obtenue lui sert d'indicateur pour guider son processus d'apprentissage car c'est également lui qui essaie d'améliorer progressivement cette logique à chaque nouvelle itération jusqu'à ce qu'un critère d'arrêt soit atteint. La Figure 4.5 nous permet de visualiser ces interactions : dans un sens par la logique décisionnelle transmise à la simulation et dans l'autre par la performance obtenue communiquée à l'apprentissage.

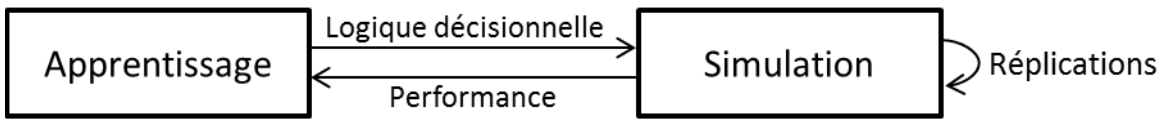

FIGURE 4.5 – Interactions entre simulation et apprentissage

Notre approche peut ainsi être vue comme un système à deux boucles, l'une dite « externe » et l'autre « interne ». Un moteur d'apprentissage propose des solutions dont les performances sont évaluées par simulation : c'est la boucle externe. Elle est répétée jusqu'à ce qu'un critère d'arrêt soit atteint. Pour que cette boucle puisse être réalisée, la simulation se sert en interne du résultat intermédiaire du moteur d'apprentissage, c'est-à-dire, de la solution (logique décisionnelle) courante. Cette dernière accède aux données de simulation pour ensuite lui faire connaître ses décisions d'adaptation : c'est la boucle interne. La Figure 4.6 résume les principes de base de l'approche que nous suggérons.

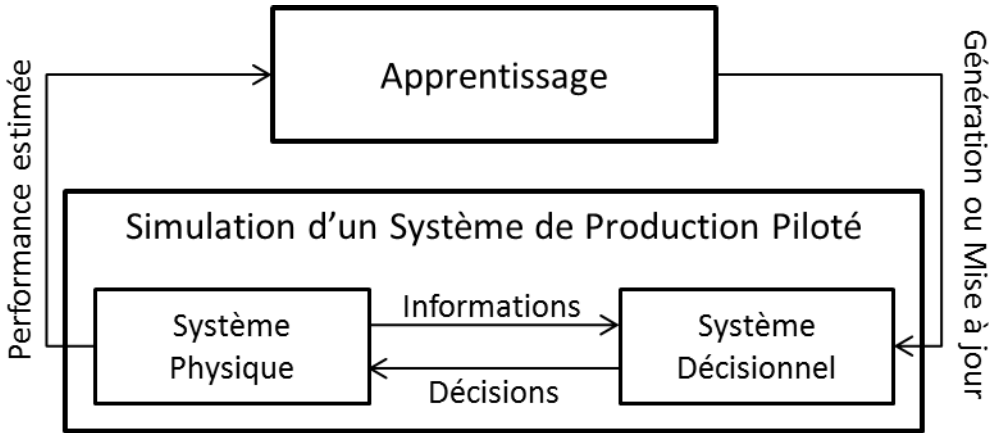

FIGURE 4.6 – Fonctionnement d'une approche d'apprentissage par simulation

Notons que la notion de réplications n'est volontairement pas explicitée dans ce schéma puisqu'elle est considérée comme intrinsèque à la simulation. Dorénavant, nous ne ferons référence à cette notion que lorsque des précisions la concernant seront nécessaires.

Puisque nous avons besoin d'une technique concrète d'apprentissage pour poursuivre notre étude, quelques précisions seront données sur le fonctionnement de notre approche en accord avec la technique utilisée.

4.5 Choix d'une méthode d'apprentissage

Nous avons identifié dans la Section 4.3 que notre problème d'adaptation aboutit à un problème d'extraction de connaissances. Ce dernier peut être traduit par l'optimisation de la logique décisionnelle φ en utilisant sa structure et ses composantes comme variables de décision. Il nous faut donc une approche de résolution admettant une représentation qui soit capable d'exprimer une « fonction ». L'approche doit de plus être en mesure de modifier les paramètres de la représentation impactant sur la fonction effectivement générée.

Plusieurs techniques existent. Les réseaux de neurones par exemple possèdent des propriétés déjà évoquées leur permettant d'approximer des fonctions non linéaires où le nombre de neurones et leurs poids font office de paramètres à exploiter. Dans le cadre de cette thèse, nous avons regardé plus précisément la programmation génétique (PG) : la fonction sera donc un arbre et nous interviendrons sur les éléments qui y sont associés pour le modifier.

Le principe de fonctionnement et les notions de base de cette méthode ont été énoncés dans la Section 3.2.2.4. Nous nous concentrerons ici sur les motivations à l'origine de ce choix.

4.5.1 Intérêt de la programmation génétique pour le problème posé

Tout d'abord, la programmation génétique est une technique qui bénéficie de l'approche évolutionniste : elle fait évoluer non pas une solution unique mais toute une population de solutions à la fois. Les opérateurs génétiques (mutations et croisements) dont elle se sert sont par ailleurs largement développés et discutés dans la littérature. La PG prend ainsi les avantages de cette « famille d'approches » tout en palliant un potentiel inconvénient : elle admet des solutions de tailles variées. Ceci permet de représenter des solutions (dans notre cas des logiques décisionnelles) des plus simples aux plus complexes. Elles sont donc potentiellement très différentes, augmentant la possibilité d'en trouver une qui soit efficace.

D'autre part, la PG produit des fonctions explicites contrairement à d'autres approches non paramétriques telles que les réseaux de neurones et peut ainsi fournir un moyen de comprendre l'impact des variables et leurs relations [Can et Heavey 2012]. Bien que cet aspect « boîte noire » des réseaux de neurones souvent identifié comme un inconvénient [Doukidis et Angelides 1994, Metan *et al.* 2010] puisse de nos jours être contourné par des méthodes qui en déduisent les règles, la PG reste une approche plus directe. En effet, c'est une technique qui fait de l'apprentissage en manipulant directement des arbres. Cette représentation est tout à fait compatible avec ce que l'on cherche : une logique décisionnelle telle que nous l'avons définie peut être représentée sous la forme de règles de production ou d'arbres de décision, nous le verrons par la suite (Section 4.6). Elle constitue même un avantage pour les méthodes d'apprentissage qui sont basées dessus : leur pouvoir explicatif rend l'interprétation des résultats beaucoup plus simple car l'utilisation de modèles dits « boîte blanche » rend une décision particulière aussi transparente que possible [Markham *et al.* 2000, Dudas *et al.* 2011, Shahzad et Mebarki 2012]. Le résultat est donc lisible et compréhensible. Ceci est d'un plus grand intérêt dans le contexte de l'adaptation des systèmes de production où s'inscrit cette thèse : nous pouvons disposer de connaissances directement exploitables par un système expert ou par un décideur [Pierreval et Ralambondrainy 1990]. C'est donc une représentation utile autant pour communiquer que pour décider automatiquement [Markham *et al.* 2000]. De plus, la modularité offerte par ce type de formalisme facilite sa mise à jour, par exemple lors d'un changement des objectifs de production ou des caractéristiques de l'atelier [Pierreval 1992a]. De même, elle augmente sa capacité à être « éduqué », ce qui peut facilement être fait par un opérateur habilité qui pourra ajouter des nouvelles règles ou en modifier des existantes au besoin [Doukidis et Angelides 1994].

Les vastes potentiels de cette représentation (et par conséquent de la PG) sont dus également à sa capacité de traiter des données à la fois quantitatives (continues ou discrètes) et qualitatives (nominales) [Markham *et al.* 2000, Shahzad et Mebarki 2012]. Elle permet le manque d'homogénéité dans les données et présente des avantages vis-à-vis des réseaux de neurones en termes de vitesse de développement [Markham *et al.* 2000]. Nous

pouvons encore souligner d'autres avantages d'après [Shahzad et Mebarki 2012] : les méthodes d'apprentissage reposant sur des arbres de décision nécessitent peu de préparation des données, sont robustes et donnent de bons résultats avec de grandes quantités de données et en peu de temps.

La programmation génétique possède ainsi plusieurs propriétés intéressantes qui la rendent attractive pour notre étude. Par ailleurs, c'est une technique bien reconnue et largement utilisée dans différents domaines dont voici quelques variantes :

- Linéaire [Nordin 1994, Banzhaf *et al.* 1998, Brameier 2004, Brameier et Banzhaf 2007] ;
- Basée sur des graphes [Teller et Veloso 1996] ;
- Cartésienne [Miller et Thomson 2000] ;
- Linéaire arborescente [Kantschik et Banzhaf 2001] ;
- À couches [Lin *et al.* 2008].

Citons également quelques exemples de champs d'applications :

- Prévision de séries chronologiques [Santini et Tettamanzi 2001] ;
- Classification de données médicales [Brameier et Banzhaf 2001, Lin *et al.* 2008] ;
- Ordonnancement [Beham *et al.* 2008] ;
- Agencement [Garces-Perez *et al.* 1996] ;
- Traitement de signal et d'images [Bolis *et al.* 2001] ;
- Reconnaissance de caractères manuscrits [Teredesai *et al.* 2001] ;
- Développement de stratégies pour jeux vidéo [Sun *et al.* 2009] ;
- Système de contrôle de robots impliquant souvent leur déplacement (trajectoire).

Ce dernier peut avoir plusieurs orientations comme par exemple :

- Jouer au football [Gustafson et Hsu 2001] ;
- Coordonner le mouvement de la main et la vision [Langdon et Nordin 2001] ;
- Lutter dans un combat de sumo [Sharabi et Sipper 2006] ;
- Gérer la circulation des visiteurs dans un espace d'exposition [Fukunaga *et al.* 2012].

Le fait que la PG n'ait jamais été utilisée à notre connaissance pour traiter notre type de problématique participe à l'originalité de cette thèse.

Toutefois, en étudiant plus en détail nos besoins et la littérature, il s'est avéré que la programmation génétique telle qu'introduite par [Koza 1992] n'était pas la plus adéquate. En effet, elle est surtout utilisée pour le développement de formules mathématiques bien qu'elle comporte des fonctions conditionnelles du type « si-alors ». Nous avons décidé de garder une représentation finale en format d'arbre, ce qui est visuellement attrayant et fournit au décideur un schéma à partir duquel une règle générale peut être extrapolée [Markham *et al.* 2000]. En revanche, nous avons opté pour un codage qui se rapproche à une structure de programme informatique ou règles de production, ce qui nous a conduits plus précisément vers la programmation génétique linéaire.

4.5.2 Programmation génétique linéaire

La programmation génétique linéaire (PGL) a été créée tout comme la PG pour le développement automatique de programmes. L'adjonction de « linéaire » précise ici que les programmes (ou codes) sont exprimés sous forme d'une simple suite d'instructions [Banzhaf *et al.* 1998] par opposition à la structure originelle d'arbre [Koza 1992]. La PGL n'a donc aucun rapport avec la programmation linéaire. Au contraire, les programmes représentent des solutions fortement non linéaires en raison de leur puissance inhérente d'expression [Brameier 2004, Brameier et Banzhaf 2007].

Les programmes sont écrits dans un format (presque) « naturel » : les instructions se rapprochent le plus possible d'un langage de programmation de haut niveau ou d'un langage machine. Parfois ils peuvent même être exécutés directement sans passer par une phase intermédiaire d'interprétation, ce qui résulte en un gain considérable en temps de calcul lors de l'évaluation de leur *fitness* [Banzhaf *et al.* 1998, Brameier et Banzhaf 2007]. Cette structure est un avantage de la PGL : son champ d'application peut s'étendre à plusieurs autres problèmes pour lesquels une modélisation arborescente est impossible ou inadéquate tout en profitant de la richesse de la PG. La PGL allie donc la puissance et la flexibilité de la PG sous forme d'arbre à la représentation itérative d'un code machine traditionnel, le tout permettant notamment l'utilisation de sauts conditionnels et d'instructions manipulant la mémoire [Mahboub 2011].

La structure de la PGL permet de respecter la logique impérative du code qui exécute ses instructions unes à unes sans aucune hiérarchie particulière. Par conséquent, là où la PG traditionnelle sera plus adaptée à une formulation mathématique simple à représenter à l'aide d'un arbre, la PGL visera davantage une structuration plus séquentielle dans laquelle toute ligne de code possède le même statut [Mahboub 2011]. La position des instructions détermine ainsi l'ordre de leur exécution, l'ajout ou la suppression d'un ou de plusieurs éléments pouvant modifier complètement la sémantique du programme. Le contrôle de flux du programme n'est plus implicite comme l'est le cas des arbres : il est assuré par un mécanisme de sauts conditionnels. En effet, un programme peut contenir de nombreuses bifurcations, chacune pouvant conduire à une autre partie du programme. Ceci définit un flux de données à base de graphes et permet une utilisation multiple de certaines parties du programme accédées par différents chemins sans avoir à les recopier. D'autre part, il faut également signaler l'existence de segments de code non effectifs appelés introns, impliquant un nombre d'instructions plus important que dans un programme constitué uniquement de code effectif.

Les dernières caractéristiques évoquées ont des implications importantes sur l'application de la PGL [Banzhaf *et al.* 1998, Brameier 2004, Poli *et al.* 2008]. Tout d'abord, les branchements et sauts conditionnels permettent une gestion plus efficace des fonctions logiques « ET » et « OU ». La réutilisation permet des solutions plus compactes en taille et pouvant exprimer des calculs plus complexes avec moins d'instructions par rapport à la représentation arborescente. Quant aux introns, ils correspondent à des instructions sans

aucune influence sur le comportement du programme, c'est-à-dire n'ayant pas d'impact sur son résultat. Nous avons par exemple des portions de code jamais exécutées (victimes de branchements conditionnels systématiques) ou encore exécutées mais inutilement car n'apportant rien à l'exécution du programme (opérations annulatrices ou branchements avec « si » et « sinon » pointant sur les mêmes instructions). Les introns dont l'intérêt peut paraître à priori nul sont en fait considérés comme bénéfiques lors de l'évolution (processus d'apprentissage). Ils réduisent efficacement l'effet parfois indésirable voire destructeur des variations sur le code effectif résultant des croisements et mutations. Ils permettent ainsi à plus de variations de rester neutres en termes de changement du *fitness*. Les introns favorisent alors la stabilité des solutions mais, puisqu'ils fournissent plus de liberté pour les générations de programmes futurs, ils servent également à leur diversité. Cette dernière est aussi liée au fait déjà évoqué que des simples changements (ordre des instructions, paramètre d'une instruction, etc.) peuvent générer des programmes ayant des comportements très différents.

Un autre aspect à prendre compte concerne les opérateurs génétiques utilisés dans le but de déterminer la suite d'instructions la plus adaptée à un problème. L'application de certains de ces opérateurs issus de la PG devient inefficace voire impossible puisqu'ils sont dédiés aux arbres. L'évolution de programmes informatiques avec ces opérateurs devrait alors être compliquée. Ceci n'est cependant pas le cas. Les programmes sont des séquences d'instructions dont les frontières sont bien définies [Banzhaf *et al.* 1998]. Les instructions sont généralement spécifiées de sorte qu'elles soient autonomes : leur combinaison quel que soit l'ordre constitue un programme valide. Les croisements et mutations opèrent généralement à la frontière entre deux instructions, ce qui rend ces deux types d'opérateurs très simples à mettre en œuvre.

En somme, la PGL est une variante encore très peu usitée de nos jours mais qui possède des propriétés remarquables [Mahboub 2011].

4.6 Structure d'une logique décisionnelle du point de vue de la programmation génétique linéaire

En programmation génétique, un arbre est définit comme des fonctions manipulant des terminaux. Dans le cadre de la PGL, un programme est une séquence d'instructions qui manipulent des registres mémoire. Il y a une certaine analogie entre fonctions et instructions bien que ces dernières soient définies de façon plus complète. De même, il existe une analogie entre terminaux et registres mémoire. Les composantes d'un programme appartiennent à des ensembles prédéfinis : un pour les instructions et un autre pour les registres mémoire. La capacité de la PGL à trouver une solution dépend fortement de l'expressivité de ces composantes, alors que la dimension de son espace de recherche augmente exponentiellement avec leur nombre [Brameier et Banzhaf 2007]. Par la suite, nous

décrirons ces composantes en rapport avec la logique décisionnelle φ qui nous intéresse. Elles définissent son codage, phase essentielle à la résolution d'un problème.

4.6.1 Registres mémoire

Comme nous l'avons vu dans les sections précédentes, une logique décisionnelle récupère des informations provenant du système de production. Nous pouvons ainsi dire que ce dernier possède des capteurs dont les valeurs sont transmises à φ via des registres mémoire. La fonction φ procède ensuite à leur traitement, ce qui consiste à exécuter le programme qu'elle représente. Une fois cette opération terminée, les valeurs de sortie calculées sont transmises au système de production toujours via des registres mémoire. Ces derniers ne sont cependant pas liés à de simples capteurs, ils activent également les responsables de la mise en place des actions qui leur ont été communiquées, à savoir des ressources matérielles et/ou humaines. Ce raisonnement supposant un système fortement automatisé et contrôlé par un ordinateur pourrait ne pas trouver beaucoup d'applications. Cependant, il est tout à fait compatible avec l'intervention de gestionnaires de production. Dans ce cas, ces derniers peuvent être nécessaires pour recueillir les données d'entrée requises par φ mais surtout, ils restent les responsables des véritables décisions à appliquer au système. Ainsi, φ sert à leur envoyer des alertes et peut conduire à de bonnes pratiques tandis qu'ils gardent le pouvoir de décision selon leurs propres interprétations.

Les registres mémoire sont par définition le moyen de stockage et de modification de l'information numérique. Ils sont regroupés dans un ensemble prédéfini T' décomposé en trois sous-ensembles :

- Registres d'entrée (T_V) : Ils correspondent aux données accessibles en lecture seule par φ pour le calcul de la valeur des autres registres et font référence aux composantes de S_t (variables d'état, historiques résumés par des indicateurs, indicateurs temporaires de performance) ;
- Registres de sortie (T_A) : Il s'agit des valeurs de retour accessibles et modifiables par φ qui correspondent aux actions d_t devant être appliquées au système ;
- Registres (d'entrée) constants (T_C) : Également appelés Constantes, ce sont des valeurs fixes accessibles en lecture seule. Elles comprennent d'une part les actions $\mathcal{D}$ applicables au système et d'autre part les paramètres $\mathcal{P}$ et valeurs auxiliaires typiquement utilisées en tant que seuils. Nous décomposons donc T_C en deux catégories selon leur usage : $T_{CA} = \mathcal{D}$ et $T_{CV} = T_C \setminus T_{CA} = T_C - T_{CA}$.

Notons que les paramètres $\mathcal{P}$ peuvent aussi faire partie de T_V selon l'usage souhaité. C'est le cas par exemple si l'on cherche une logique décisionnelle commune à des systèmes équivalents dont la seule différence est la valeur de leurs paramètres. C'est aussi le cas si les paramètres sont fixes pour la période étudiée mais que l'on envisage de les modifier définitivement ou de les rendre variables dans un futur proche. Ces cas ne serons cependant pas analysés dans le cadre de cette thèse.

D'autres registres dits temporaires sont parfois utilisés de façon provisoire pour des calculs intermédiaires avant de stocker les résultats finaux dans les registres de sortie, leur nombre devant être limité afin d'éviter une explosion combinatoire [Mahboub 2011]. Bien que ce type de registre constitue un sous-ensemble de T' à part, nous ne nous en servirons pas.

La taille de l'espace de recherche est alors en lien avec la cardinalité de l'ensemble T' de registres mémoire qui peut être définie par l'Équation 4.5 ci-dessous :

$$|T'| = |T_V| + |T_A| + |T_C| \tag{4.5}$$

Si des intervalles réels $[v_{min}; v_{max}]$ sont nécessaires pour définir T_C, l'espace de recherche serait infini. Il peut cependant être réduit en remplaçant les intervalles réels par des intervalles d'entiers $[v_{min} \cdot 10^{nc}; v_{max} \cdot 10^{nc}]$ où nc correspond au nombre de décimales à prendre en compte. Une conversion inverse est donc nécessaire lors de leur utilisation. Ceci revient à discrétiser les intervalles initiaux avec un pas égal à 10^{-nc}.

4.6.2 Instructions

Traditionnellement, un programme informatique est composé d'une suite d'instructions distinctes. Nous pouvons citer les simples affectations à un seul registre, celles comprenant des calculs intermédiaires (opérations arithmétiques et booléennes, fonctions exponentielles et trigonométriques) mais aussi les branchements conditionnels et les boucles. Ces deux dernières instructions peuvent également être simples ou comprendre des calculs. Nous n'utiliserons ici que deux types d'instructions : l'affectation et la conditionnelle. Chacune sera sous sa forme simple, la conditionnelle possédant deux déclinaisons. Remarquons que l'approche restera facilement adaptable à l'utilisation d'autres instructions.

Un programme est également doté d'un pointeur qui détermine l'instruction à exécuter et donc son état d'avancement. Ce pointeur joue un rôle crucial dans le bon enchaînement des instructions : il permet à tout moment de faire les branchements conditionnels (ou boucles) qui viennent rompre le flot d'exécution afin de l'interrompre ou d'accéder à d'autres parties du programme. Il suppose l'usage d'un compteur ordinal et éventuellement d'étiquettes. Ces dernières constituent selon [Mahboub 2011] une source potentielle d'erreurs au sein des programmes. Le compteur débute naturellement à la première instruction. L'exécution du programme s'effectue alors séquentiellement et ce jusqu'à ce qu'il soit finalisé, le pointeur étant déplacé à chaque fois d'une unité ou comme indiqué par les conditionnelles dont le test s'avère positif.

4.6.2.1 Affectation

Les affectations sont des instructions prenant en paramètre un registre par définition modifiable. Dans notre cas, elles se limiteront aux registres de sortie mais il serait

également possible de considérer des registres temporaires.

L'affectation utilisée ici consiste alors plus précisément à mettre à jour la valeur d'un registre de sortie en lui appliquant une de ses valeurs possibles. Ceci peut être exprimé sous la forme de l'Équation 4.6 ci-dessous :

$$d_t = d'_t \tag{4.6}$$

où d_t (introduit dans la Section 4.3) est le vecteur de variables auquel on souhaite attribuer une valeur. Il représente le registre de sortie T_A tandis que d'_t prend l'une des valeurs autorisées pour ce registre : $d'_t \in T_{CA} = \mathcal{D}$.

Puisque c'est l'affectation qui détermine nos variables $d_{n,t}$ et qu'elle les détermine toutes à la fois, nous avons ici trois particularités dans notre modélisation :

- Lorsqu'une affectation est faite, l'exécution du programme s'arrête ;
- L'affectation ne peut pas être la première instruction d'un programme ;
- Chaque programme doit avoir une affectation comme dernière instruction.

La première particularité peut être expliquée par le simple fait que l'affectation nous retourne une décision concernant les adaptations à mettre en place ou non : puisque c'est exactement ce que l'on cherche comme résultat, il n'y a pas de raison d'aller plus loin dans l'exécution du programme. La deuxième est liée au fait que, si une affectation est faite sans qu'aucune condition ne soit analysée, la logique décisionnelle ne correspondrait pas à une stratégie d'adaptation puisque la décision prise serait toujours la même. Enfin, la troisième assure que la fin de l'exécution du programme retourne toujours une décision. Notons que cette dernière correspond le plus souvent à une décision par défaut lorsqu'aucune autre décision n'a été retenue auparavant.

Nous verrons par la suite que l'affectation « pure » (Équation 4.6) sera en fait utilisée seulement une fois, à la fin du programme.

4.6.2.2 Conditionnelle

Les concepts de programmation tels que les branchements et les boucles permettent de modifier le flux de données d'un programme et même de les définir selon l'entrée spécifique utilisée [Brameier et Banzhaf 2007]. Nous nous limiterons ici aux branchements qui sont au cœur de la logique décisionnelle à laquelle nous nous intéressons. En PGL, ils permettent une gestion plus efficace des fonctions logiques « ET » et « OU », surtout lorsqu'ils sont associés à des mécanismes que la programmation impérative appelle sauts conditionnels ou *goto/jumpTo* (« aller à »). Cette déclinaison de la conditionnelle est responsable du déplacement du flot d'exécution à un autre endroit du programme, ce qui est fait ici à l'aide d'étiquettes :

$$\text{if } (v_1 \; cmp \; v_2) \text{ jumpTo } nL \tag{4.7}$$

où jumpTo indique une redirection du flot d'exécution et nL correspond à l'étiquette de la ligne où le pointeur du programme doit se déplacer. En outre, *cmp* est un opérateur comparatif $(=, \neq, >, \geq, <, \leq)$ qui sépare ses deux opérandes v_1 et v_2. Afin d'optimiser l'efficacité de l'exécution, notre modélisation interdit la comparaison de deux constantes de sorte que v_1 corresponde toujours à un registre d'entrée, v_2 pouvant être un registre d'entrée ou une constante.

En théorie, ces sauts peuvent être positifs ou négatifs. Les premiers déplacent le pointeur en avant, donc plus « loin » dans le code, tandis que les seconds font le contraire. Ici nous interdisons les sauts négatifs qui engendrent souvent la création de boucles infinies [Mahboub 2011]. Ainsi, si la condition est vérifiée (vraie), le jumpTo a lieu et la séquence d'exécution est déplacée à la ligne indiquée par l'étiquette nL qui se trouve forcément plus loin dans le code. Si en revanche la condition n'est pas vérifiée (ce qui équivaut au « sinon »), l'exécution du programme se poursuit normalement, ce qui dans notre cas revient à passer à la ligne suivante.

Notons que nL indique ici une étiquette générique. Sa véritable valeur au sein d'un programme sera attribuée lors de la construction de celui-ci, et ce pour chaque instruction en ayant besoin.

Comme nous avons indiqué précédemment, la conditionnelle peut prendre une deuxième déclinaison. En effet, il est aussi possible que cette instruction arrête tout simplement l'exécution du programme lorsqu'elle ne sert pas à effectuer un saut mais à déterminer une valeur de retour. Ceci est fait en utilisant l'affectation à la place du jumpTo nL, ce qui se présente comme suit :

$$\text{if } (v_1 \ cmp \ v_2) \ d_t = d'_t \tag{4.8}$$

où v_1, v_2 et *cmp* sont définis de la même façon que pour l'Équation 4.7 tout comme d_t et d'_t le sont pour l'Équation 4.6.

Dans ce cas, si la condition est vérifiée, la valeur du registre de sortie est affectée et l'exécution s'arrête. Sinon, l'exécution du programme se poursuit normalement, passant à la ligne suivante comme dans le cas précédent.

Nous avons alors :

$$v_1 \in T_V \tag{4.9}$$
$$v_2 \in T_V \cup T_{CV} \tag{4.10}$$
$$d_t \in T_A \tag{4.11}$$
$$d'_t \in T_{CA} \tag{4.12}$$

ainsi que :

$$cmp \in \{=; \neq; >; \geq; <; \leq\} \tag{4.13}$$

4.6.3 Programme

Un programme est une séquence d'instructions permettant de lire des registres mémoire d'entrée et éventuellement d'attribuer une valeur à chaque registre mémoire de sortie. Le résultat dépend de comment les instructions sont représentées mais surtout de comment elles sont interprétées. Nous avons vu qu'une instruction de type conditionnelle est définie par les deux opérandes devant être comparées et le comparateur à utiliser. Pour être complète, elle comprend également ce qui doit être fait si le résultat de la comparaison est vrai : une affectation ou un saut vers une nouvelle instruction. Le « sinon » (*else*) ne fait cependant pas partie de sa définition : il est découplé du « si-alors » (*if-then*) et correspond à l'instruction qui se trouve à la ligne suivante.

4.6.3.1 Concrétisation par un exemple

À titre illustratif, nous reprendrons ici l'exemple sur la décision d'un sportif d'aller jouer ou non selon les conditions météorologiques (Section 3.2.2.1.1). Selon le Tableau 3.1 de cette section, nous avons les registres mémoire suivants :

- $T_V = \{Ciel\,;\ Température\,;\ Humidité\,;\ Vent\}$
- $T_A = Classe$
- $T_{CV} = \{Ensoleillé\,;\ Couvert\,;\ Pluvieux\,;\ Élevée\,;\ Moyenne\,;\ Basse\,;\ Forte\,;\ Normale\,;\ Oui\,;\ Non\}$
- $T_{CA} = \{OUI\,;\ NON\}$

Ils constituent les données pouvant être exploitées par les instructions définies dans les Équations 4.6, 4.7 et 4.8 pour lesquelles nous pouvons considérer que $cmp \in \{=; \neq\}$. Ci-dessous un exemple de programme ayant le même comportement que l'arbre de décision de la Figure 3.4 :

if $(Ciel = Ensoleillé)$ jumpTo *label1*
if $(Ciel = Couvert)$ $Classe = OUI$
if $(Vent = Oui)$ $Classe = NON$
if $(Vent \neq Oui)$ $Classe = OUI$
label1 : if $(Humidité = Normale)$ $Classe = OUI$
$Classe = NON$

Puisque nous avons déterminé qu'une affectation arrête l'exécution du programme, des affectations qui se suivent n'ont pas raison d'exister : seule la première serait prise en compte. Pour éviter une quantité de code non effectif plus importante que nécessaire et puisqu'une affectation peut être faite lors d'une conditionnelle, nous avons opté pour ne pas admettre des instructions du type affectation pure tout au long du programme mais seulement lorsqu'il touche à sa fin.

Ce choix implique qu'une affectation ne peut jamais avoir lieu directement lors d'un « sinon », sauf si celui-ci coïncide avec la dernière instruction du programme. En théorie, c'est bien le cas puisque ce sont uniquement les tests positifs qui peuvent déclencher une affectation. En pratique, nous aurons parfois des conditions qui s'avèrent toujours vraies : si elles sont simplifiées et que seul compte le code effectif, les affectations peuvent tout de même apparaître sans conditionnelle associée dans le programme et donc correspondre au « sinon » de la condition précédente. C'est par exemple le cas de la condition « if ($Vent \neq Oui$) $Classe = OUI$ » du programme précédent : cette condition est toujours vraie puisque l'on y arrive seulement si la précédente « if ($Vent = Oui$) » est fausse.

Les deux seules manières d'achever l'exécution d'un programme sont soit d'atteindre une affectation suite à une conditionnelle, soit d'atteindre l'ultime ligne de code qui correspond elle aussi à une affectation. Ainsi, si l'on résume le fonctionnement d'un programme, des nouvelles conditions s'enchaînent jusqu'à ce qu'une affectation soit atteinte. Notons qu'un enchaînement de conditions correspond à l'utilisation de la fonction logique « ET », ce qui se traduit souvent par une suite de sauts. D'autre part, une même portion de code accédée par des chemins différents correspond quant à elle à l'utilisation de la fonction logique « OU », par exemple par des sauts ayant des origines différentes.

Le programme que nous venons de donner en exemple peut encore être écrit comme suit :

if ($Ciel = Ensoleillé$) jumpTo *label1*
if ($Ciel = Couvert$) $Classe = OUI$
if ($Vent = Oui$) jumpTo *label2*
if ($Vent \neq Oui$) $Classe = OUI$
label1 : if ($Humidité = Normale$) $Classe = OUI$
label2 : $Classe = NON$

Dans ce cas et en extrayant quelques règles, nous pouvons voir deux façons d'accéder au *label2*, chacune ayant besoin de plus d'une condition vérifiée : if ($Ciel = Ensoleillé$ ET $Humidité \neq Normale$) OU ($Ciel \neq Ensoleillé$ ET $Ciel \neq Couvert$ ET $Vent = Oui$) : $Classe = NON$.

Profitons de cet exemple pour en tirer une autre conclusion. Malgré les différences structurelles qu'il peut y avoir entre deux programmes, les résultats qu'ils fournissent peuvent être identiques. Ceci nous mène à une constatation mentionnée dans de nombreux travaux en PG [Blickle et Thiele 1994, Nordin et Banzhaf 1995, Miller et Smith 2006, Poli *et al.* 2008] : une même solution peut avoir différentes représentations, ce qui est souvent dû aux parties redondantes ou inaccessibles de ces représentations. Cette propriété est considérée comme utile lors de la recherche d'une « bonne » solution [Koza 1992, Garces-Perez *et al.* 1996, Tunstel et Jamshidi 1996].

4.6.3.2 Validité d'un programme : restrictions basées sur les types

Si nous analysons maintenant les données dont nous disposons et l'ensemble d'instructions défini par les Équations 4.6, 4.7 et 4.8, il nous manque un aspect-clé de la construction automatique des programmes : Comment assurer que son espace de recherche se restreigne aux programmes syntaxiquement et sémantiquement valides ? La réponse se trouve au niveau des instructions de type conditionnelle qui doivent être partiellement redéfinies pour éviter les potentielles erreurs.

Pour la plupart des applications, la définition des registres mémoire et des instructions telle que nous l'avons faite jusque-là est suffisante. Cette définition est cependant incomplète dans notre cas en raison des différentes natures de nos registres. En effet, elle n'est pas conçue pour gérer plusieurs types de données [Montana 1995] puisque des placements arbitraires de registres voire de comparateurs dans les programmes pourraient conduire à des violations syntaxiques graves. Des règles supplémentaires de construction doivent donc être considérées [Montana 1995, Tunstel et Jamshidi 1996, Bot 2000]. Citons les variables de type qualitatif ou binaire pour lesquelles seuls les comparateurs d'égalité ($=$) et d'inégalité ($\neq$) seront disponibles.

Afin d'assurer que les instructions soient valides, nous ne permettrons pas toutes les combinaisons possibles des données disponibles. Nous opterons plutôt pour constituer différentes sous-classes d'instructions. Chacune d'entre elles regroupera des registres d'entrée qui utilisent les mêmes comparateurs et qui peuvent être comparées aux mêmes registres d'entrée et/ou constantes. Dans cet exemple en particulier, chaque registre d'entrée sera à l'origine d'une sous-classe d'instruction avec leurs valeurs possibles respectives, ce qui évite des comparaisons qui n'ont pas de sens. Pour chaque sous-classe de conditionnelle, les deux déclinaisons sont considérées : l'affectation ou le renvoi. Nous avons donc l'ensemble suivant d'instructions :

- $Classe = \{OUI\,;\, NON\}$
- if ($Ciel$ $\{=\,;\,\neq\}$ $\{Ensoleillé\,;\, Couvert\,;\, Pluvieux\}$) jumpTo nL
  if ($Ciel$ $\{=\,;\,\neq\}$ $\{Ensoleillé\,;\, Couvert\,;\, Pluvieux\}$) $Classe = \{OUI\,;\, NON\}$
- if ($Température$ $\{=\,;\,\neq\}$ $\{Élevée\,;\, Moyenne\,;\, Basse\}$) jumpTo nL
  if ($Température$ $\{=\,;\,\neq\}$ $\{Élevée\,;\, Moyenne\,;\, Basse\}$) $Classe = \{OUI\,;\, NON\}$
- if ($Humidité$ $\{=\,;\,\neq\}$ $\{Forte\,;\, Normale\}$) jumpTo nL
  if ($Humidité$ $\{=\,;\,\neq\}$ $\{Forte\,;\, Normale\}$) $Classe = \{OUI\,;\, NON\}$
- if ($Vent$ $\{=\,;\,\neq\}$ $\{Oui\,;\, Non\}$) jumpTo nL
  if ($Vent$ $\{=\,;\,\neq\}$ $\{Oui\,;\, Non\}$) $Classe = \{OUI\,;\, NON\}$

Notons que si nous avions une variable supplémentaire *Motivation* pouvant prendre les valeurs *Oui* ou *Non* comme la variable *Vent*, elle ferait partie des deux dernières sous-classes que nous représenterions comme suit :

- if ($\{Vent\,;\, Motivation\}$ $\{=\,;\,\neq\}$ $\{Oui\,;\, Non\}$) jumpTo nL
  if ($\{Vent\,;\, Motivation\}$ $\{=\,;\,\neq\}$ $\{Oui\,;\, Non\}$) $Classe = \{OUI\,;\, NON\}$

Pour former un programme, nous pouvons utiliser les différentes conditionnelles en n'importe quel nombre et dans n'importe quel ordre, toujours avec une affectation à la fin. Rappelons que cet exemple a été choisi de sorte à faciliter la compréhension, tout en sachant que nous nous intéressons à apprendre à adapter un système de production par le déplacement de machines, le recrutement d'opérateurs ou autre.

4.6.3.3 Représentation finale

Une logique décisionnelle φ est un programme composé d'une part de registres mémoire permettant de s'interfacer avec l'environnement et d'autre part d'une liste d'instructions permettant de manipuler (accéder et/ou modifier) la valeur de ces registres. Encore une fois, soulignons que ceci peut bien évidemment être fait par l'intermédiaire et sous la validation de gestionnaires, φ correspondant à de bonnes pratiques qui leur sont communiquées. Dans la Section 4.5.1, nous avons indiqué que bien que ce codage soit sous la forme d'un programme informatique, nous souhaitons une représentation finale de φ sous la forme d'un arbre de décision.

Dans cette optique, les feuilles d'un arbre peuvent être de trois types : les décisions, les informations disponibles (variables d'état ou indicateurs) et les constantes. Ces données sont manipulées par les nœuds internes de l'arbre : des conditions du type *if-then-else*, ce qui explique que nous ayons un arbre de degré 4. Autrement dit, chaque condition prend quatre arguments comme nœuds enfants : deux devant être comparés et deux autres de conclusion qui correspondent aux résultats vrai et faux de la comparaison. Ces conclusions sont soit une décision par l'affectation des variables de décision, soit un envoi vers une instruction plus loin dans le code. Cette instruction sera quant à elle une affectation ou une nouvelle condition.

Pour mieux visualiser le résultat, la Figure 4.7 illustre l'arbre de décision obtenu par le dernier programme que nous avons donné en exemple. Notons que dans cette représentation finale sous forme d'arbre, l'affectation n'est pas explicite : puisque nous avons un seul registre de sortie, nous indiquons directement la valeur qu'il prendra (les feuilles ont directement la valeur d'_t à la place de l'Équation 4.6 : $d_t = d'_t$).

Toutefois, seul le code effectif sera considéré pour la représentation finale d'une solution, ce qui est utile à sa compréhension [Quinlan 1987, Koza 1992, Garcia-Almanza et Tsang 2006]. Pour des facilités de lecture, nous adopterons également la version la plus connue d'un arbre : binaire. La Figure 4.8 illustre l'arbre dans sa version binaire et simplifiée. La liaison en gras indique la simplification faite tandis que celles en pointillé correspondent tout simplement à la suppression d'étiquettes pour une lecture plus directe.

Dans une solution obtenue par des méthodes du type PG-PGL, l'existence de redondances ou de code non effectif est connue et utile mais, pour des raisons de clarté, il est souhaitable de les élaguer à la fin. Puisqu'une simplification automatique est difficile [Poli *et al.* 2008], une alternative est de procéder comme [Garcia-Almanza et Tsang 2006] :

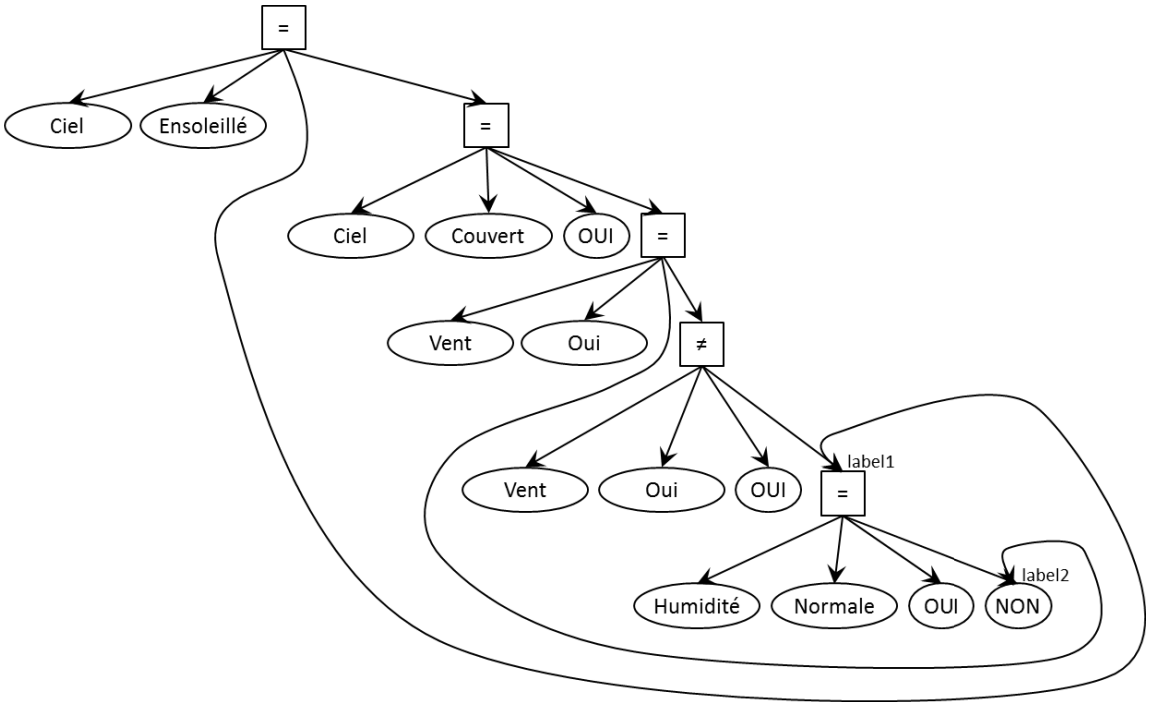

FIGURE 4.7 – Exemple d'arbre de décision à 4 degrés

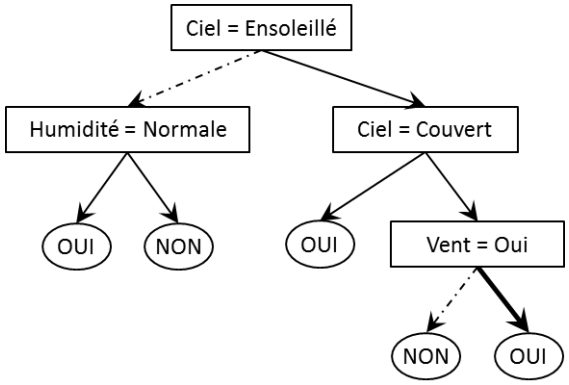

FIGURE 4.8 – Exemple d'arbre de décision simplifié

utiliser une méthode pour traiter à posteriori les arbres de décision afin d'en produire des plus compacts et compréhensibles (réduits en taille et complexité). La méthode se base sur l'analyse et l'évaluation des composants d'un arbre. S'en suivent la suppression manuelle des parties inaccessibles (lorsqu'une condition s'avère toujours vraie ou fausse) et des simplifications évidentes (par exemple un « si » et un « sinon » se fusionnant). Nous utiliserons donc cette méthode et ce, uniquement sur nos solutions finales.

4.7 Apprentissage par programmation génétique linéaire

La structure générale d'un programme est maintenant connue et par voie de conséquence, celle d'un arbre de décision aussi. Chacun de ces deux éléments correspondent à des formes de représentation d'une logique décisionnelle. Dans cette thèse, c'est l'application de la programmation génétique linéaire couplée à la simulation qui nous permettra d'en obtenir une. Le fonctionnement de notre approche expliqué dans la Section 4.4.2 sera donc complété par quelques particularités provenant de l'usage de la PGL.

La programmation génétique (linéaire ou autre) est un algorithme à population (Section 3.2.2.4). Ainsi, à chaque étape de l'évolution, nous avons potentiellement plusieurs nouveaux individus qui doivent être évalués. L'évolution fait ici référence au processus d'apprentissage. Par ailleurs, chaque étape de l'évolution est appelée génération et correspond donc à une population dont chaque solution est appelée individu.

Dans ce qui suit, nous utiliserons la notation suivante où les termes génération et population cohabitent selon leur usage le plus récurrent dans la littérature du domaine :

- P_{max} : Nombre maximum de générations de l'algorithme ne comprenant pas la population initiale ;
- $P_{fin} \leq P_{max}$: Nombre effectif de générations limité à P_{max} ;
- $p \in \{0; \ldots; P_{fin}\}$: Indice pour les populations (y compris l'initiale désignée par 0) ;
- V_{max} : Taille maximale d'une population pour l'algorithme ;
- $V_{max,p} \leq V_{max}$: Nombre d'individus dans la population p limité à V_{max} ;
- $v \in \{1; \ldots; V_{max,p}\}$: Indice pour les individus d'une population ;
- L_p : $p^{ième}$ population ;
- $\varphi_{v,p} \in L_p$: $v^{ième}$ individu de la $p^{ième}$ population.

Et puisque $\mathcal{L}$ regroupe toutes les logiques décisionnelles pouvant être trouvées par les registres mémoire et instructions disponibles (voir Sections 4.3 et 4.6), nous avons $L_p \subset \mathcal{L}$.

Pour son fonctionnement, la PGL a besoin de la définition d'un certain nombre de paramètres. Comme pour tout algorithme évolutionniste :

- Chaque individu est évalué par son *fitness* qui doit être spécifié, comme par exemple les coûts de production (à minimiser) ;
- Il faut définir un ou plusieurs critères d'arrêt, le plus commun étant un nombre maximum de générations (donc P_{max}) ;
- Des paramètres de contrôle tels que la taille de la population (donc V_{max}) et les opérateurs de croisement et mutation doivent également être précisés.

Avant de les détailler, le fonctionnement général peut être décrit comme suit. L'idée générale reste la même : apprendre itérativement à partir d'un modèle de simulation (Section 4.4). Puisque la PGL est un algorithme à population, une population initiale

$L_0 \subset \mathcal{L}$ de logiques décisionnelles est générée au départ, aléatoirement ou non. À chaque génération, les nouveaux individus doivent être évalués. Soient L_p et $\varphi_{v,p}$ la population courante et l'individu en train d'être évalué. La simulation, suite à plusieurs réplications, attribuera à $\varphi_{v,p}$ la valeur de son *fitness* : $\mathbb{E}[\vec{f}(\varphi_{v,p}, \xi)]$. Ce processus concerne chaque individu $v = 1..V_{max,p}$ de la population ayant besoin d'être évalué. Autrement dit, le processus d'évaluation résumé auparavant par la Figure 4.4 est donc répété pour chacun des nouveaux individus de la population. Une fois tous évalués, leur valeur de *fitness* sont communiquées à la PGL qui modifie la population courante de logiques décisionnelles. Nous obtenons ainsi L_{p+1}, ce qui est fait dans le but d'optimiser la fonction objectif en appliquant les opérateurs génétiques qui seront décrits dans les sections suivantes. Ce cycle est ensuite répété pour la nouvelle population et ainsi de suite jusqu'à ce qu'un critère d'arrêt soit atteint. L'Algorithme 4.1 reprend les principes de base de l'approche avec les particularités provenant de la programmation génétique linéaire.

Données : Taille V_{max} de la population, Critère(s) d'arrêt
$p := 0$
Génération de la population initiale L_0, où un individu correspond à une logique décisionnelle comme décrit dans la Section 4.6
tant que *population courante non entièrement évaluée* **faire**
    Sélection d'un individu non évalué de la population
    Évaluation du *fitness* de l'individu sélectionné via simulation
tant que *critère d'arrêt non atteint* **faire**
    Application des opérateurs génétiques sur la population afin d'en créer une nouvelle
    pour *chaque nouvel individu non évalué* **faire**
        Évaluation du *fitness* de l'individu via simulation
    $p := p + 1$
    Sélection des individus composant la nouvelle population L_p
Sauvegarde de la solution : Individu avec le meilleur *fitness*

Algorithme 4.1 : Fonctionnement d'une approche de programmation génétique linéaire basée sur la simulation

Pour résumer, la PGL génère une population initiale de logiques décisionnelles utilisées unes par unes par la simulation qui lui renvoie chaque performance. Cela va guider son processus d'apprentissage, c'est-à-dire de mise à jour des logiques.

4.7.1 Initialisation

En PGL, l'initialisation est souvent aléatoire [Cornuéjols et Miclet 2003, Brameier et Banzhaf 2007]. Il est cependant possible de la faire de façon plus « rationnelle », en insérant

volontairement des individus ayant des caractéristiques que l'on juge intéressantes. Cette pratique fait polémique. En effet, elle peut accélérer le processus d'apprentissage en lui indiquant dès le départ de « bonnes » zones à exploiter. Cependant, elle prive aussi le moteur d'apprentissage de son potentiel de « créativité », ce dernier offrant une plus grande diversité au départ et permettant d'exploiter des zones prometteuses qui ne le sont pas par intuition.

Pour utiliser une initialisation aléatoire, il est nécessaire de définir des bornes supérieure et inférieure pour la taille d'une solution (c'est-à-dire, le nombre d'instructions d'une logique décisionnelle). La taille de chaque solution est donc choisie aléatoirement dans cet intervalle, le plus souvent avec une probabilité uniforme [Brameier et Banzhaf 2007].

Notons que la structure de logique décisionnelle que nous adoptons (Section 4.6) garantit déjà que seules des solutions viables sont générées, et ce même dans le cas d'une procédure d'initialisation aléatoire. Cette structure inclue par construction des restrictions basées sur les types telles que définies dans [Montana 1995] pour les éléments qui peuvent être choisis pour chaque nœud d'un arbre en PG. Chaque argument d'une instruction voit son contenu restreint selon son placement de sorte que seuls les types « corrects et compatibles » soient insérés, éliminant les possibles incohérences (Section 4.6.3.2).

4.7.2 Sélection et Remplacement

Nous entendons ici par sélection l'opérateur qui détermine les individus qui vont subir les opérations génétiques (parents) pour donner naissance à de nouveaux individus (enfants). Cela peut être fait en utilisant par exemple une méthode largement connue dans le domaine appelée sélection par tournoi. Avec celle-ci, un nombre d'individus égal à la taille du tournoi est sélectionné aléatoirement dans la population, le meilleur d'entre eux (en termes de *fitness*) étant choisi. Si l'opérateur génétique devant être appliqué requiert deux individus, deux tournois seront faits. Notons que si la taille du tournoi est de 1, le choix est complètement aléatoire et la sélection est dite uniforme. À l'opposé, si la taille est égale à celle de la population, la sélection est dite déterministe par rang et choisit toujours les individus possédant les meilleurs *fitness*. Nous pouvons également utiliser une sélection en fonction du classement, appelée sélection par roulette, où chaque individu a une probabilité d'être choisi qui est proportionnelle à sa position dans la population selon son *fitness*.

Le remplacement quant à lui concerne la sélection parmi les nouveaux et les anciens individus afin de former la nouvelle population. C'est pourquoi il est parfois considéré comme un opérateur de sélection. Nous avons typiquement deux méthodes : l'une consiste à garder les meilleurs individus toutes populations confondues (stratégie « plus »), l'autre à remplacer complètement la population de parents par celle d'enfants (stratégie « comma » ou de remplacement générationnel). Tandis que la première peut conduire à une convergence

prématurée de l'algorithme vers un optimum local, la deuxième peut perdre le meilleur individu au cours de l'évolution. Ainsi, plusieurs variantes existent. Nous pouvons par exemple opter pour une comparaison entre enfants et parents respectifs où l'on garde le ou les meilleurs de la « famille » en cours et non pas les meilleurs parmi toute la population de parents-enfants. Dans ce cas, certains « mauvais individus » ont des chances de survivre. Dans le cas du remplacement générationnel, une alternative reconnue consiste à garder un certain nombre de « bons individus » par ce que l'on appelle l'élitisme : les meilleurs individus sont toujours maintenus dans la population, ce qui demande la définition de la taille de l'élite (nombre d'individus « immortels »). Ceci évite de perdre le meilleur individu. Notons cependant que si le nombre d'individus retenus est trop important, l'élitisme peut également conduire à une convergence prématurée. C'est pourquoi la taille de l'élite est généralement définie comme un pourcentage peu élevé de la taille de la population.

4.7.3 Opérateurs génétiques

Les opérateurs génétiques doivent être définis de sorte que les enfants générés soient conformes à la définition d'un individu (Section 4.6). Rappelons que d'après cette définition, un individu est un programme et chaque instruction dans un programme correspond à une ligne de code. Pour les explications qui suivent, nous utiliserons donc le numéro des lignes pour faire référence à une instruction précise.

Tout d'abord, nous avons pris une précaution au niveau des instructions du type conditionnelle et plus précisément de celles ayant pour résultat un saut. Bien que ces sauts soient ici indiqués par jumpTo suivi d'une étiquette, ils sont gérés en interne par un entier indiquant à combien de lignes plus loin il faut se déplacer. Les étiquettes sont alors créées à posteriori et automatiquement insérées au début de la ligne correspondante. Ceci facilite l'application des opérateurs génétiques (croisement ou mutation) et évite la création de sauts négatifs pouvant entraîner des boucles infinies.

Comme dans la plupart des applications, les opérateurs de croisement produisent deux nouveaux individus tandis que ceux de mutation en produisent un seul. Pour les croisements, deux individus de la population courante sont combinés. Pour les mutations, un seul individu est aléatoirement modifié. Ces opérateurs doivent bien évidemment respecter les tailles minimale et maximale définies pour l'initialisation [Cornuéjols et Miclet 2003].

Pour ces deux types d'opérateur, différentes variantes existent. Nous détaillerons le principe de celles les plus couramment trouvées dans la littérature.

Commençons par les deux types de croisement : avec un ou deux points de découpe. Le premier (onePointImpreciseCrossover) sélectionne aléatoirement une ligne pour chacun des deux parents participant à l'opération et ensuite échange leurs sous-programmes (portion du programme allant de la ligne sélectionnée jusqu'à la dernière ligne). Ceci est

illustré par la Figure 4.9.

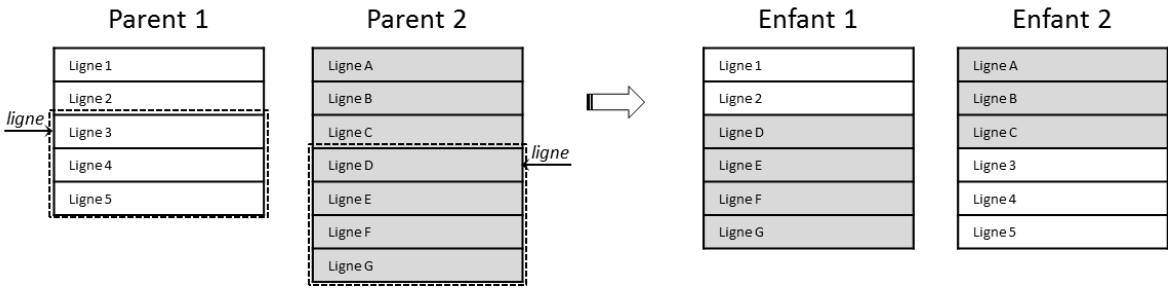

FIGURE 4.9 – Application de l'opérateur de croisement à un point de découpe

Le deuxième croisement (twoPointImpreciseCrossover) sélectionne quant à lui deux lignes par parent : $ligne_H$ et $ligne_B$ sachant que $ligne_H$ désigne une ligne toujours plus « haute » dans le code que $ligne_B$ et que la sélection des lignes est indépendante entre parents. Les sous-programmes délimités par ces lignes sont alors échangés entre parents pour donner naissance à deux enfants. Notons que la portion de programme à échanger va toujours de la $ligne_H$ jusqu'à la ligne précédant la $ligne_B$ (donc $ligne_B$ 1). La Figure 4.10 en donne un exemple.

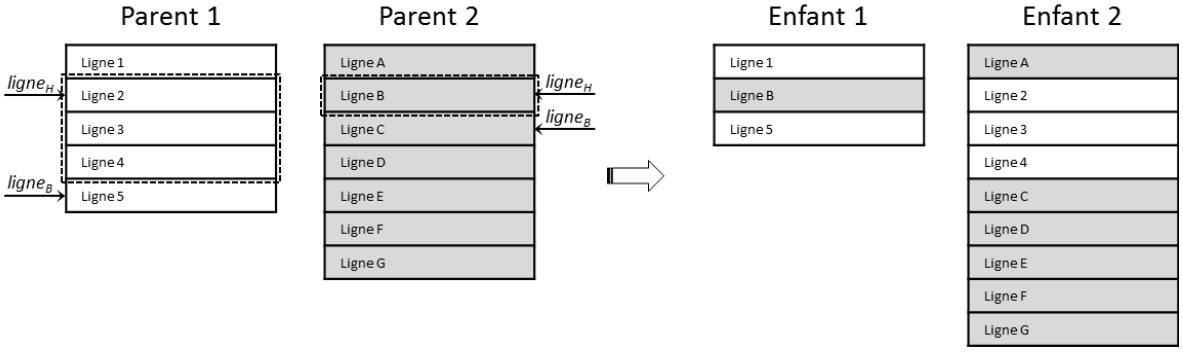

FIGURE 4.10 – Application de l'opérateur de croisement à deux points de découpe

Lors des mutations, il est possible d'ajouter (insertionMutation) ou de supprimer (removalMutation) des instructions mais aussi de modifier ou de remplacer les existantes. Dans le cas d'une modification, celle-ci peut être faite soit sur un seul composant d'une conditionnelle (singleParameterAlterationMutation), soit sur tous ses composants (alterationMutation). Le composant modifié prendra une autre valeur possible selon le groupe auquel appartient la conditionnelle, ce qui assure que le programme enfant restera valide. Dans le cas d'un remplacement, une conditionnelle appartenant à un autre groupe sera insérée à la place de la conditionnelle faisant objet de la mutation (replacementMutation). Nous avons ainsi cinq mutations différentes, toutes illustrées dans la Figure 4.11. Il s'agira d'une macro-mutation ou d'une micro-mutation selon celle choisie [Mahboub 2011].

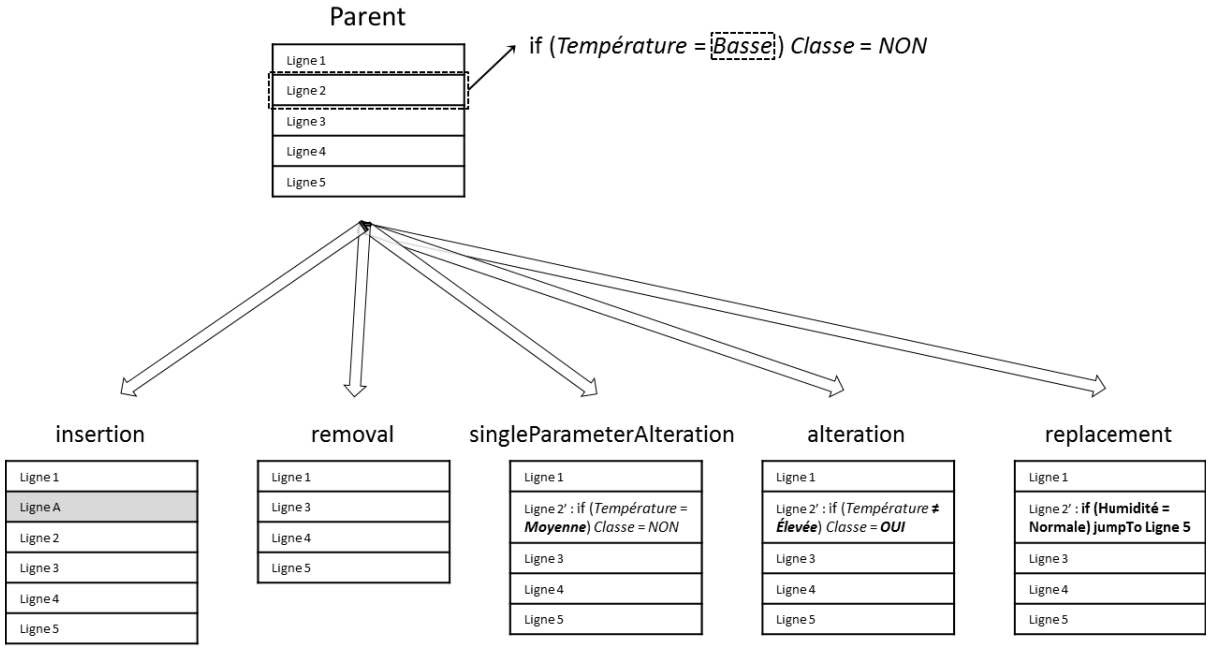

FIGURE 4.11 – Application des cinq opérateurs de mutation à un même programme d'origine

4.7.4 Critères d'arrêt

Différents critères (ou conditions) d'arrêt existent. Ils peuvent être utilisés exclusivement ou ensemble, auquel cas le premier critère atteint indique la fin de l'exécution. Citons comme exemples de critère le nombre maximum de générations, le nombre maximum de générations sans amélioration, le nombre maximum d'évaluations, le temps maximum d'exécution ou encore la performance cible. Pour ce dernier, il est souhaitable d'en ajouter un autre (plus maîtrisable) afin d'assurer la fin de l'exécution.

Ces critères sont intimement liés au nombre de solutions à évaluer. Ils peuvent même fixer directement ce dernier si le nombre maximum d'évaluations est le seul critère d'arrêt. Dans le cas contraire, il est toujours judicieux de combiner ce critère avec d'autres pour garantir une borne supérieure du nombre d'évaluations, quel que soit le critère visé : le temps de calcul en dépend. Toutefois, signalons que le nombre total d'évaluations aura tendance à dépasser légèrement cette borne puisque chaque génération (composée de plusieurs nouveaux individus à évaluer) est atomique.

4.8 Implémentation

Puisque notre approche s'est basée sur deux techniques complémentaires, notre implémentation se base elle aussi sur deux modules qui seront détaillés dans ce qui suit : l'un pour la simulation, l'autre pour l'apprentissage.

En termes d'outils informatiques, nous nous sommes servis de :

- Arena, version 14.5 : Simulation à événements discrets ;
- μGP, version 3.2.0 : Programmation Génétique Linéaire.

Un transfert de données direct entre le module de simulation Arena et le module d'apprentissage μGP ne nous semble pas possible, leurs entrées/sorties se faisant par intermédiaire de fichiers (Figure 4.12). De plus, ceux générés par μGP contenant les logiques décisionnelles ne sont pas directement exploitables par Arena. Une interface est donc nécessaire pour les décoder et pouvoir par la suite y accéder à travers le modèle de simulation. Nous avons opté pour la définition d'un troisième module, à part, responsable de cette connexion Arena-μGP en particulier. Ce module a été réalisé en langage C++ à l'aide de Visual Studio Professionnel 2012.

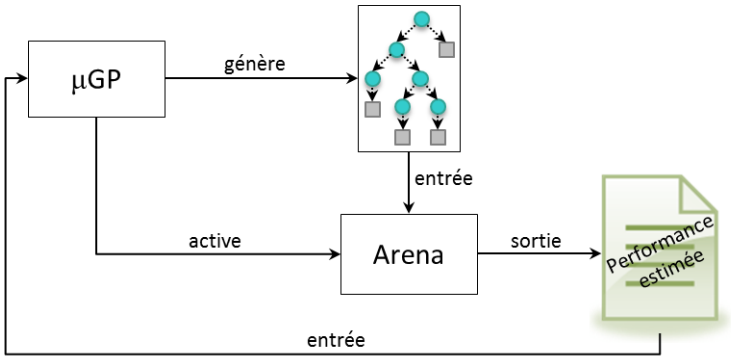

FIGURE 4.12 – Interactions μGP-Arena pour l'extraction de connaissances

D'autres schémas seront utilisés pour mieux expliquer les interactions à différents niveaux de détails.

4.8.1 Arena : le module de simulation

Arena est un produit de Rockwell Automation utilisant Siman comme langage intégré. Au-delà des fonctions de base pour lire et écrire des données dans des fichiers, il supporte notamment l'utilisation de fichiers DLL permettant l'ajout de code utilisateur externe sous forme de bibliothèque dynamique. De plus, le logiciel peut être utilisé de façon « externe » en une ligne de commande (application console) ou à travers un programme qui y fait appel. Des détails sur ces fonctionnalités sont réunis dans l'Annexe B. Pour d'autres informations, des introductions à Arena sont disponibles dans [Collins et Watson 1993, Hammann et Markovitch 1995, Pegden *et al.* 1995, Altiok et Melamed 2007, Kelton *et al.* 2015] ainsi que de façon plus détaillée dans les guides pour l'utilisateur fournis par Rockwell [Rockwell Software 2007a, Rockwell Software 2007b].

Dans cette thèse, nous avons tiré profit de la capacité d'Arena à interagir avec un environnement C++ via l'outil Visual Studio C++. L'utilisation d'une DLL et l'appel de

Siman en ligne de commande ont constitué deux aspects importants de l'implémentation de notre approche.

La DLL constitue notre module à part pour l'interprétation et l'exploitation des logiques décisionnelles bien que nous l'ayons introduite en lien avec la simulation. En association avec la logique décisionnelle, elle peut être vue comme une base de données externe à laquelle la simulation se connecte plutôt qu'une composante codée en interne. Le schéma 4.13 montre leur interaction.

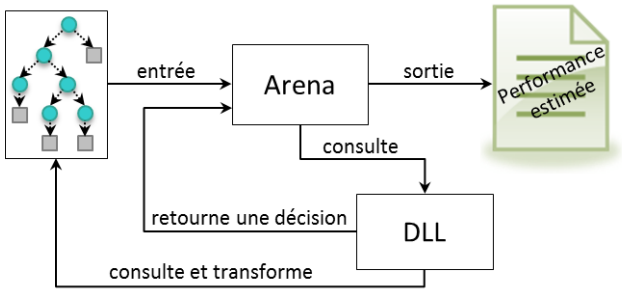

FIGURE 4.13 – Interactions Arena-DLL pour la prise de décision dans un système de production

4.8.2 μGP : le module d'apprentissage

μGP est quant à lui un logiciel libre et gratuit écrit en C++. Une fois compilé, l'application console (exécutable nommé « ugp3.exe ») est contrôlée par un ensemble de fichiers de configuration écrits en XML et qui constituent la principale interface avec l'utilisateur. Quelques options de ligne de commande sont également disponibles. Pour plus de détails, se référer à [Squillero 2005, Sanchez *et al.* 2011, Sanchez *et al.* 2012].

Pour vérifier la pertinence du logiciel vis-à-vis de notre problématique, nous avons développé deux exemples de *benchmark* pour lesquels le résultat est connu à l'avance. Ils sont disponibles dans l'Annexe C.

Différents paramètres sont nécessaires au bon fonctionnement d'un algorithme de programmation génétique linéaire. En effet, les performances de notre approche, que ce soit en termes de *fitness* ou de temps de calcul, dépendent fortement des « configurations » utilisées pour la PGL. Une configuration correspond ici à une combinaison spécifique de paramètres donnant lieu à une expérimentation. Elle définit le niveau de complexité des solutions puisqu'elle en impose les tailles minimale et maximale. D'autre part, une configuration définit également le nombre de solutions à évaluer. Elle le fait de façon indirecte puisque ce nombre est fonction de la taille de la population, des critères d'arrêt ainsi que du nombre de nouveaux individus par génération (lui-même fonction du nombre d'opérateurs à appliquer et de leur taux d'occurrence). Elle le fait aussi de façon approximative en raison des facteurs aléatoires.

Dans ce que suit, nous nous concentrerons donc sur les différents paramètres qui seront utilisés dans nos expérimentations. Ils seront bien évidemment détaillés pour chaque cas d'application dans le chapitre suivant, le but étant ici de fournir quelques précisions concernant nos choix pour ces paramètres et/ou leur fonctionnement dans μGP.

4.8.2.1 Initialisation et Tailles d'individus

Comme déjà mentionné, une configuration indique tout d'abord les tailles minimale et maximale que peut avoir un individu, ce qui correspond ici à une plage de valeurs pour le nombre d'instructions dans une logique décisionnelle. Ces tailles sont valables tout au long de l'évolution y compris pour la population initiale.

Nous utiliserons dans cette thèse l'initialisation aléatoire. La taille des individus ainsi générés par μGP suit une distribution normale dont la moyenne et l'écart-type constituent deux paramètres supplémentaires devant être précisés.

4.8.2.2 Opérateurs génétiques

Rappelons que l'affectation pure (non conditionnelle) est utilisée exclusivement en fin de programme. De plus, nous avons opté pour qu'elle représente toujours la décision par défaut de ne rien faire. Elle n'est donc pas concernée par les différents opérateurs que nous pourrons avoir.

Nous utiliserons un total de sept opérateurs génétiques, tous déjà disponibles dans le logiciel. Ce sont plus précisément les sept explicités dans la Section 4.7.3 : onePointImpreciseCrossover, twoPointImpreciseCrossover, insertionMutation, removalMutation, singleParameterAlterationMutation, alterationMutation et replacementMutation dont les notations sont inspirées de celles du logiciel. Au démarrage de l'algorithme, ils sont tous associés à des probabilités d'activation équivalentes, à savoir $\frac{1}{7}$. Ensuite, un facteur d'auto-adaptation disponible dans le logiciel est responsable de faire évoluer ces probabilités automatiquement et progressivement selon la performance de leur opérateur respectif [Belluz *et al.* 2015].

Le logiciel contient un paramètre qui régule la force des opérateurs. Il est normalement plus élevé au départ pour obtenir des individus plus différents, le facteur d'auto-adaptation l'ajustant au fur et à mesure de l'évolution. Les cinq mutations peuvent alors se répéter plusieurs fois pour un même individu selon la valeur courante de ce paramètre.

À noter que le logiciel présente une particularité concernant l'utilisation des opérateurs génétiques : il requiert comme paramètre le nombre d'opérateurs génétiques qui sera appliqué à chaque génération plutôt que le nombre d'enfants à générer [Squillero 2005, Sanchez *et al.* 2011]. Ces deux paramètres sont forcément liés mais puisque le choix des opérateurs est stochastique, nous ne pouvons pas en définir un en fonction de l'autre.

Que ce soit pour le croisement ou la mutation, si un opérateur produit un individu ne respectant pas les tailles minimale ou maximale, ledit individu sera écarté mais l'opération sera tout de même comptabilisée. Bien que rare, si cette situation venait à se produire à répétition, il faudrait redéfinir les limites à des valeurs plus raisonnables.

4.8.2.3 Paramètres divers

Les notations pour les paramètres de la PGL sont résumées dans le Tableau 4.1 ci-dessous, bien que certaines aient déjà été introduites dans la Section 4.7. Elles seront suivies de quelques précisions sur leur fonctionnement spécifique dans μGP.

Paramètres	**Notations**
Taille maximale de la population	V_{max}
Taille de l'élite	V_{elite}
Immortalité	Immortalité
Nombre d'opérations génétiques à appliquer	$NbOp$
Nombre maximum de générations	P_{max}
Nombre maximum de générations sans amélioration	P_{stagne}
Taille du tournoi de sélection	$V_{tournoi}$

Tableau 4.1 – Paramètres de la PGL

Le paramètre « Immortalité » indique si les individus dans la population survivent ou non d'une génération à l'autre (Section 4.7.2). Lorsqu'il est activé, les meilleurs individus sont toujours gardés, parents et enfants compris. Dans le cas contraire, la population d'enfants remplace complètement celle de ses parents. C'est uniquement dans ce cas que la taille de l'élite a du sens : V_{elite} définit le nombre d'individus « immortels » (un pourcentage peu élevé de V_{max}) correspondant aux meilleurs de toute la population en termes de *fitness*.

Comme nous venons de l'expliquer dans la section précédente, le logiciel μGP prend en paramètre, en plus des taux d'occurrence des opérateurs génétiques, le nombre d'opérations (donc d'opérateurs) à appliquer à chaque itération. Précisons ici que ce nombre est généralement défini en fonction de la taille de la population. Pour des expérimentations où l'immortalité n'est pas activée, il est d'usage de le fixer entre V_{max} et $2 \cdot V_{max}$. Notons que, le plus souvent, il arrivera de générer plus d'individus que nécessaire dans la population (nombre d'enfants $> V_{max}$). Pour déterminer lesquels d'entre eux seront gardés, ils seront tous évalués, le choix étant fait d'après leur *fitness*. Lorsque l'immortalité est activée, nous pouvons compter sur les individus de la population de parents et donc générer moins d'enfants, le *fitness* étant toujours nécessaire pour les départager.

La sélection est ici faite par tournoi, ce qui a été empiriquement défini. À ce propos, μGP offre la possibilité de définir non pas une taille de tournoi précise mais une plage de valeurs permettant l'auto-adaptation de la taille réellement utilisée au cours de l'évolution.

Enfin et concernant les critères d'arrêt, il s'agit de deux des plus classiques utilisés

simultanément de façon à ce que le processus d'apprentissage s'arrête lorsque l'un des deux est atteint. D'autres critères sont également disponibles dans le logiciel.

Nous pouvons observer que plusieurs comparaisons entre individus sont faites au cours d'un processus d'évolution. Elles déterminent le gagnant dans un tournoi, les survivants qui seront retenus dans la nouvelle population ou encore, à la fin, le meilleur individu. Il est clair que, dans des contextes stochastiques, la comparaison de deux solutions afin d'en déterminer la meilleure exige des tests de comparaison statistiques, augmentant ainsi le temps de calcul. En revanche, comme l'indiquent [Pierreval et Tautou 1997], cela s'avère moins important pour les méthodes à base de population de solutions qui ne suivent pas un chemin particulier dans l'espace de recherche. Il est ainsi possible de procéder comme si le modèle était déterministe, au détail près que le *fitness* utilisé correspond à la valeur moyenne obtenue sur plusieurs réplications et non pas à une constante.

4.9 Fonctionnement pas à pas

Le schéma de la Figure 4.14 illustre notre implémentation. Pour faciliter sa compréhension et mettre en évidence les différentes boucles existantes (population, réplications, évolution du temps), l'Algorithme 4.2 est fourni en complément, dont les étapes font référence aux numéros de la figure.

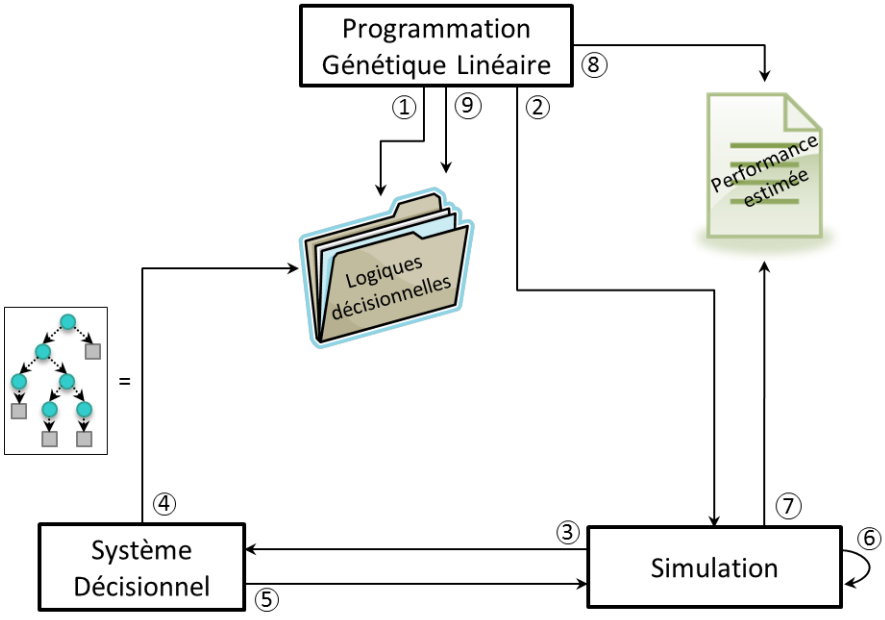

FIGURE 4.14 – Interactions entre les différents acteurs de l'approche proposée

Deux diagrammes de séquence sont également fournis dans l'Annexe D pour résumer le fonctionnement de notre approche d'un point de vue plus technique. Ils sont complémentaires et mettent en évidence les échanges entre les différents acteurs (utilisateur et logiciels).

Données : Taille de la population, Nombre de réplications R de la simulation, Durée H de simulation, Critère(s) d'arrêt

PGL : Génération de la population initiale de logiques décisionnelles, chacune écrite dans un fichier texte (1)
tant que *critère d'arrêt non atteint* **faire**
  pour *chaque individu non évalué de la population* **faire**
    `// Simulation: Initialisation des réplications`
    $numeroReplication := 1$
    PGL : Appel à la simulation (en passant le fichier à évaluer en paramètre) pour évaluer le *fitness* de l'individu correspondant (2)
    tant que $numeroReplication \leq R$ **faire**
      `// Simulation: Initialisation de la réplication courante`
      $dateCouranteSimulation := 0$
      `// Simulation: Avancement de l'horloge de la simulation`
      $dateCouranteSimulation :=$ date du prochain événement déclenchant une décision
      tant que $dateCouranteSimulation \leq H$ **faire**
        Simulation : Appel au système décisionnel pour la prise de décision (3)
        si *premier appel au système décisionnel pour cet individu* **alors**
          DLL : Lecture du fichier contenant la logique décisionnelle (4)
        DLL : Accès à la valeur des variables dans la simulation
        DLL : Envoi d'une décision en retour à la simulation (5)
        `// Simulation: Application de la décision et avancement de l'horloge de la simulation`
        Simulation : $dateCouranteSimulation :=$ date du prochain événement déclenchant une décision
      `// Simulation: Passage à la réplication suivante`
      $numeroReplication := numeroReplication + 1$ (6)
    Simulation : Écriture de la performance estimée après plusieurs réplications dans un fichier texte dédié (7)
    PGL : Lecture du fichier contenant la performance de la logique décisionnelle évaluée (8)
  si *critère d'arrêt non atteint* **alors**
    PGL : Génération de la nouvelle population de logiques décisionnelles par l'application des opérateurs génétiques sur la population courante (9)
PGL : Sauvegarde de la solution dans un fichier texte : Individu avec le meilleur *fitness*

Algorithme 4.2 : Interactions de la Figure 4.14 en détail

4.10 Conclusion

Pour qu'un système de production puisse s'adapter à un environnement changeant, il doit être capable de déterminer par lui-même chaque moment où il est judicieux de faire un changement mais aussi le contenu de ce changement. Il faut donc des outils d'aide à la décision pour piloter cette adaptation, ce qui peut se traduire par un besoin de connaissances utiles au gestionnaire de production ou à l'auto-adaptation.

Ces connaissances peuvent être à la portée des décideurs grâce à l'approche que nous avons proposée dans ce chapitre. En effet, elle fournit comme délivrable une logique décisionnelle pour l'adaptation d'un système, logique directement compréhensible par un gestionnaire de production et pouvant être facilement exploitée par un système d'aide à la décision automatisé en ligne.

L'approche développée ici extrait des connaissances à partir de la simulation, ce qui présente l'avantage de ne pas avoir besoin d'ensembles d'apprentissage et montre justement que cela est possible. En identifiant ainsi une logique décisionnelle efficace qui détermine comment un système de production devrait s'adapter, c'est une approche qui peut fortement aider dans le domaine. Elle permet de traiter différents types d'adaptation et, de par l'usage de la simulation, peut être adaptée à différents types de systèmes. Toutefois, bien que cette approche soit capable d'apporter une plus-value (nous le verrons dans le chapitre suivant), nous avons conscience que des problèmes d'adaptation requièrent également d'autres types de connaissances. Nous ne prétendons donc pas qu'elle puisse être capable de résoudre tous les problèmes liés à l'adaptation.

Nous avons introduit les bases de la programmation génétique linéaire retenue pour l'étude de cette approche. Par ailleurs, nous avons également développé une logique de communication entre les logiciels Arena et μGP afin de les adapter à notre problématique de façon pertinente. Maintenant qu'ils sont combinés et associés à la formalisation proposée, nous pouvons illustrer le potentiel de notre approche. Cela sera fait à l'aide d'exemples que nous détaillerons au cours du prochain et dernier chapitre.

Chapitre 5

Applications

5.1 Introduction

Dans cette thèse, nous nous sommes intéressés à la mise en œuvre d'une approche d'apprentissage par simulation afin d'étudier dans quelle mesure une telle approche pourrait contribuer à l'adaptation des systèmes de production.

Ce chapitre a deux objectifs principaux. Le premier est de montrer la faisabilité de l'approche proposée dans le chapitre précédent, approche qui se base sur la programmation génétique linéaire. Cela passe évidemment par la vérification de l'adéquation de nos choix en termes de logiciels. Comme nous l'avons signalé dans la Section 4.8, ceci a été fait à l'aide de deux exemples qui font l'objet de l'Annexe C.

Le deuxième objectif, centré sur notre contribution, est de confronter notre approche à des exemples pour en illustrer les bénéfices. À cet égard, nous étudierons l'adaptation dans le cadre d'un système à flux tiré extrait de la littérature mais aussi d'un système de production à ressources partagées. L'idée fut d'exploiter différents types de système et/ou d'adaptations par des exemples spécifiquement conçus pour. Nous avons aussi cherché à illustrer des décisions pouvant avoir lieu sur différents horizons temporels.

5.2 Adaptation dynamique du nombre de cartes dans un système ConWIP

5.2.1 Contexte

Pour notre premier cas d'étude, nous nous intéressons aux systèmes de production à flux tiré qui fonctionnent à base de cartes. Dans ce type de système, les cartes autorisent et contrôlent ainsi la production [Bollon *et al.* 2004, González-R *et al.* 2012]. Chaque produit dans le système est rattaché à une carte disponible. Le système Kanban est l'un des plus connus avec une boucle de cartes pour chaque étape du processus de production [Monden 1981, Lage Junior et Godinho Filho 2010]. Nous étudions ici plus précisément un ConWIP où une seule boucle est utilisée, le flux étant poussé une fois la première étape du processus autorisée [Spearman *et al.* 1990, Framinan *et al.* 2003, Prakash et

Chin 2014, Burlat 2015a, Burlat 2015c]. La Figure 5.1 ci-dessous illustre leur principe de fonctionnement. Signalons que ces systèmes sont largement étudiés et sont même à l'origine de logiciels dédiés comme WipSim, un simulateur de flux en mode ConWIP [Burlat 2015b].

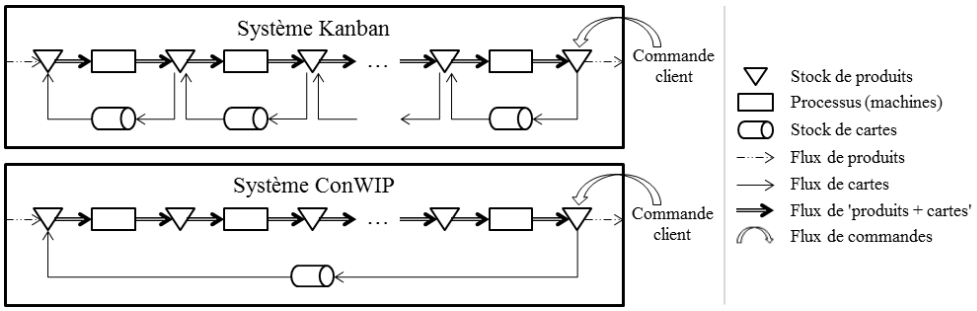

FIGURE 5.1 – Systèmes de contrôle à flux tiré de types Kanban et ConWIP

Dans ce que suit, nous nous concentrerons sur le cas des systèmes ConWIP. L'extrapolation aux systèmes Kanban reste cependant tout à fait possible et nous avons d'ailleurs formalisé le problème sous ce point de vue dans [Belisário et Pierreval 2015].

Dans un système ConWIP dit traditionnel, l'optimisation consiste à trouver le nombre de cartes qui minimise les coûts. Ces coûts sont souvent un compromis entre le stockage (encours et stock final) et la non-satisfaction des clients (rupture de stock). Une telle optimisation part de l'hypothèse que la demande est stable, ce qui est rarement le cas avec les marchés et les besoins des clients qui changent. De plus, tout système de production est soumis à de nombreuses perturbations dues à son caractère stochastique ou encore à la présence de pannes. Ces difficultés ont conduit de nombreux chercheurs à proposer l'adaptation dynamique du nombre de cartes via des cartes dites supplémentaires afin de rendre ces systèmes plus réactifs aux changements et donc performants sur le long terme. Malgré les recherches, le développement de tels systèmes reste encore un défi scientifique. Pour y répondre, l'utilisation d'une approche d'apprentissage par simulation telle que nous la proposons n'avait pas encore été rapportée à notre connaissance et nous en avons donc fait une publication : [Belisário et Pierreval 2015].

Dans ce qui suit, nous nous servirons du cadre conceptuel du Chapitre 2 pour spécifier le système et, par cette même occasion, les bases du problème que nous traiterons. Nous nous servirons également des notations provenant de la formalisation proposée dans la Section 4.3 du Chapitre 4.

5.2.2 Description du système

5.2.2.1 Éléments constituant le système

Le système de production sélectionné ici comme cas d'étude est extrait de [Tardif et Maaseidvaag 2001]. C'est un exemple largement repris dans la littérature sur les systèmes de production à flux tiré réactifs à des fins comparatives [Framinan *et al.* 2006, Shahabudeen et Sivakumar 2008, Sivakumar et Shahabudeen 2008, González-R *et al.* 2011, Pierreval *et al.* 2013].

Il s'agit d'un système mono-produit basé sur l'hypothèse qu'il n'existe pas de blocage au niveau des matières premières qui restent disponibles à tout moment et en quantités nécessaires. Le système comporte une station de travail dotée de 10 machines en parallèle dont les temps de traitement suivent une distribution exponentielle de moyenne 6 (ou $\mu = \frac{1}{6}$). Les commandes arrivent selon un processus de Poisson et aucune information n'est disponible sur la demande future. Ceci caractérise un processus de fabrication à un seul étage dont le flux est tiré par les commandes des clients. Celles ne pouvant pas être satisfaites immédiatement ne sont pas perdues : elles restent en attente et entraînent donc une pénalisation. À ce propos, le but est ici que le système soit le plus rentable possible (objectif G) étant donné le coût par unité de temps pénalisant la rupture de stock de chaque élément, coût 1000 fois plus élevé que celui des encours (stock final compris).

L'aspect adaptatif du système concerne le nombre de cartes qui y circulent et qui est autorisé à varier. Le système compte ainsi un seul objet modifiable. Par conséquent, $J = 1$ et l'ensemble d'objets modifiables est $O = \{o_1\}$ où o_1 représente la politique de flux tiré. Cet objet est caractérisé par un nombre d'attributs variables $K_1 = 1$, donc le vecteur correspondant est $A_1 = \langle a_{1,1} \rangle$ où $a_{1,1}$ représente le nombre de cartes circulant dans le système. Son instanciation dans le temps est notée $a_{1,1,t}$ et indique sa valeur courante à l'instant t.

Nous pouvons généralement faire deux suppositions concernant ce type de système. Il doit y avoir à tout instant un nombre minimum K_{Min} de cartes dites permanentes pour que le système puisse fonctionner correctement. Le cas extrême serait de 1. D'autre part, il ne serait pas pertinent d'en avoir plus qu'un nombre maximum K_{Max} donné. Ainsi, $a_{1,1}$ peut prendre sa valeur dans l'ensemble $E_{1,1} = [K_{Min}; K_{Max}]$, ce qui peut être exprimé par la Contrainte 5.1. La différence $K_{Max} - K_{Min}$ indique le nombre de cartes dites supplémentaires qui est quant à lui noté K_{sup}. Pour que des adaptations puissent avoir lieu, nous avons $K_{sup} \geq 1$. Le système peut alors utiliser jusqu'à K_{sup} cartes supplémentaires en plus de ses K_{Min} cartes permanentes.

$$K_{Min} \leq a_{1,1,t} \leq K_{Max}, \forall t \tag{5.1}$$

Pour pouvoir tirer profit de ce facteur de flexibilité, un seul type de changement

est nécessaire ($L = 1$). Il constitue à lui seul l'ensemble $C = \{c_1\}$ où c_1 représente l'ajout ou le retrait de cartes du système. Il est associé à un horizon décisionnel $H_{D,1}$ de type opérationnel : les temps de mise en place et de validité sont négligeables, donc $H_{durée,1} = H_{min,1} = 0$.

Cette description initiale fait que nous avons une seule variable de décision : $N = 1$. Le vecteur d'actions $d_t = \langle d_{1,t} \rangle$ indique le nombre de cartes à ajouter ou retirer du système à un instant t donné. L'ensemble $\mathcal{D}$ des actions associées se résume à $\mathcal{D} = D_1$. Dans le cas le plus restreint, nous avons $\mathcal{D} = [-1; 1]$ et dans l'extrême opposé $\mathcal{D} = [-K_{sup}; K_{sup}]$.

Notons que la valeur $a_{1,1,t}$ induite par une action d_t est obtenue par $a_{1,1,t} := a_{1,1,t} + d_t$ et doit satisfaire la Contrainte 5.1, à savoir maintenir le nombre minimum de cartes K_{Min} et ne pas en dépasser le nombre maximum K_{Max}, et ce à chaque instant t.

5.2.2.2 Fonctionnement du système

Définissons tout d'abord l'état initial du système avant l'arrivée de toute commande. Il existe un stock initial de K_{Min} produits finis prêts à être livrés aux clients (et donc K_{Min} cartes se trouvent attachées à eux) ; il n'y a pas de commandes en attente ; et le processus de fabrication est à l'arrêt puisqu'il est vide (il n'y a pas encore de commandes à traiter).

Signalons également que le temps de transport est considéré comme négligeable, que ce soit celui entre le stock initial et une machine, entre une machine et le stock final ou même entre les stocks final et initial. Ce dernier cas correspond aux cartes libérées à la sortie du système et devant être rattachées à une nouvelle demande de production.

Le fonctionnement du système peut alors être décrit comme suit. Lorsqu'une commande arrive et dès qu'un produit fini peut être livré au client, la carte de ce produit est relâchée. Elle est alors immédiatement rattachée à une demande pour la production d'un nouveau produit à l'entrée du système, sauf si elle doit être retirée de circulation (en raison d'une demande d'adaptation). En effet, avant toute livraison ou vérification si la nouvelle commande peut être satisfaite, une décision doit être prise quant à l'adaptation ou non du système à cet instant. Nous avons ici la définition de notre stratégie de déclenchement T : l'arrivée d'une commande dans le système constitue alors notre événement déclencheur suite auquel nous décidons quant à l'ajout ou au retrait de cartes. Bien évidemment, la décision peut être également de garder le système tel qu'il est, ce qui constitue notre décision par défaut.

Ces décisions cherchent à minimiser le coût total qui représente les encours, le stock final et les commandes en attente, ou autrement dit, les objectifs dans G évalués selon leurs valeurs moyennes sur une période d'étude. C'est ici qu'interviendra notre approche puisque ces décisions seront prises de façon centralisée (structure S) en utilisant une logique décisionnelle φ obtenue par l'algorithme de programmation génétique linéaire sur la

base des expérimentations d'un modèle de simulation (mécanisme de changement M). Notons que M correspond plus précisément à φ tandis que l'approche de PGL par simulation indique le principe utilisé pour l'obtenir. Ce principe peut toutefois être explicité comme étant M selon ce que l'on souhaite préciser (Section 2.3.3.2). De même, dans le cadre de nos expérimentations, la logique décisionnelle obtenue est directement exploitée par le système (de façon automatisée), ce qui nous a conduits à définir la structure S comme étant centralisée. Ceci est d'ailleurs nécessaire durant la phase d'apprentissage. Cette structure pourrait cependant être hiérarchique si l'on considère l'intervention de gestionnaires de production lors de la véritable utilisation de φ (pour le contrôle du système réel).

Pour la prise de décision, FSL_t et U_t sont à disposition [Tardif et Maaseidvaag 2001]. Ils constituent notre vecteur de variables d'état S_t, partie dynamique de notre ensemble d'informations I. FSL_t désigne le stock de produits finis à l'instant t ou les commandes en attente en cas de rupture de stock. $U_t = a_{1,1,t} - K_{Min}$ indique quant à lui le nombre maximum de cartes qui peuvent être retirées du système à ce même instant, c'est-à-dire le nombre de cartes supplémentaires couramment utilisées. Ainsi, la variable d'état U_t est associée à celle de décision et fait ici bien partie de S_t. Rappelons qu'il ne nous est pas possible de nous servir de prévisions, celles-ci n'étant pas disponibles.

5.2.2.3 Comportement de la demande

Pour qu'un système de production soit efficace, il faut qu'il soit bien dimensionné mais surtout qu'il s'adapte toujours au mieux à la situation du moment. Notre approche présentée dans le chapitre précédent a justement pour but d'aider à la décision de ces adaptations. Pour l'illustrer, nous utiliserons plus précisément trois versions de cet exemple de système ConWIP que nous venons de décrire. Elles diffèrent par l'arrivée des demandes au cours du temps qui suivent des *patterns* différentes auxquelles le système devrait s'adapter. Les demandes sont caractérisées par un processus de Poisson qui évolue selon l'une des trois possibilités ci-dessous :

- Une *pattern* dite stable où la moyenne reste constante et égale à 5 (ou $\lambda = \frac{1}{5}$) tout au long de la période d'étude ;
- Une *pattern* dite cyclique où la moyenne change à chaque 25 unités de temps, passant de 3 à 5 et puis à 7, d'où elle recommence à 3 et ainsi de suite pour toute la durée de la période d'étude ;
- Une *pattern* dite triangulaire où la moyenne augmente linéairement de 3 à 7 pendant 75 unités de temps (donc à un taux de $\frac{4}{75}$), puis diminue inversement, de 7 à 3 (taux de $-\frac{4}{75}$), au cours des 75 prochaines unités de temps, ce qui se répète pour la durée de la période d'étude.

Ces différentes *patterns* sont graphiquement représentées dans la Figure 5.2. Les deux premières versions sont identiques à celles de [Tardif et Maaseidvaag 2001], tandis que la troisième fait varier la moyenne de façon linéaire. Nous illustrons ainsi des exemples

typiques de *patterns* de demande [Thiel 1996, Dégrés *et al.* 2008]. Puisque nous utilisons la simulation pour représenter l'évolution du système au cours du temps, des variations du taux de demande moyen sont introduites dans notre modèle en accord avec chaque version étudiée.

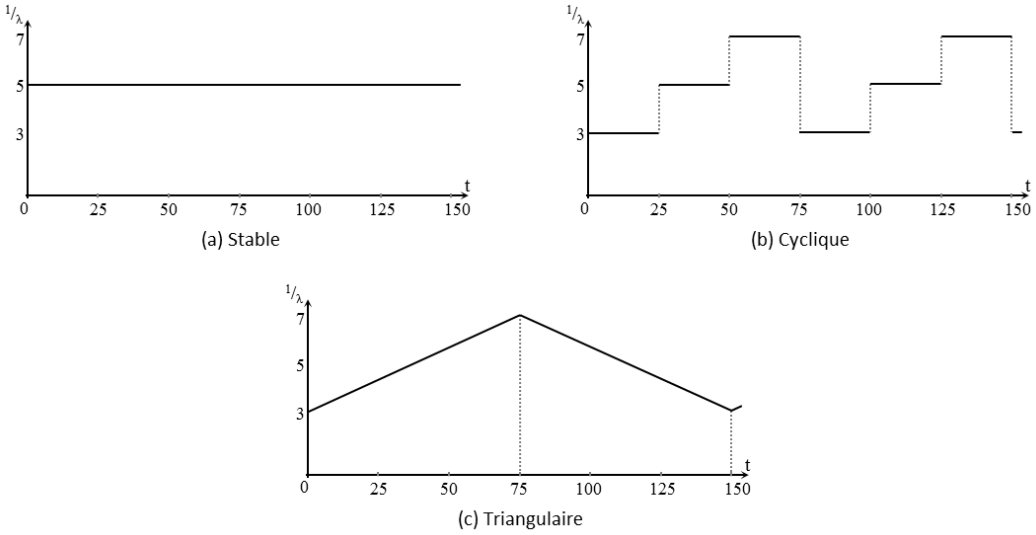

FIGURE 5.2 – *Patterns* utilisées pour la variation de la demande dans la simulation du système ConWIP

Dans ce qui suit, nous ferons référence aux différentes versions de notre exemple par le terme « scénario ». Chaque *pattern* est donc à l'origine d'un scénario et définit ce à quoi le système doit être en mesure de s'adapter.

5.2.3 Conditions d'expérimentations

5.2.3.1 Généralités

Comme décrit dans le chapitre précédent, tous nos modèles de simulation à événements discrets ont été construits avec un logiciel de simulation bien connu du marché : Arena. La composante programmation génétique linéaire a quant à elle été réalisée à l'aide du logiciel libre et gratuit μGP.

Nos expérimentations ont toutes été effectuées sur un PC équipé d'un processeur Intel Core i7-3770 (4 cœurs) à 3,40 GHz, de 16 Go de RAM et fonctionnant sous Windows 7 Professionnel 64 bits. Signalons également qu'elles ont été réalisées en deux étapes pour lesquelles la durée (conséquente) de simulation H est identique. La première constitue le processus même d'apprentissage qui utilise des simulations basées sur 5 réplications. Elle est directement à l'origine des courbes de convergence qui seront présentées par la suite. Puis, la seconde étape concerne uniquement la meilleure solution qui est de nouveau simulée à raison de 30 réplications afin d'en obtenir des estimations suffisamment

précises et de permettre des comparaisons statistiques. Sauf mention du contraire, tous les tableaux de résultats présentés ci-après correspondront donc aux valeurs obtenues après 30 réplications de la solution concernée.

5.2.3.2 Valeurs de référence pour la comparaison de nos résultats

Pour mieux évaluer nos résultats, nous procéderons à des comparaisons d'après deux bases de valeurs que nous pouvons considérer de référence. Nous nous comparerons d'une part aux résultats pour le ConWIP traditionnel et d'autre part à ceux obtenus par l'approche adaptative de [Tardif et Maaseidvaag 2001].

Dans le but de rendre notre comparaison aussi équitable que possible, nous avons établi la valeur des variables des deux approches de référence de sorte que ces dernières aient la meilleure performance possible. Pour ce faire, nous avons utilisé l'optimisation via simulation par l'optimiseur OptQuest intégré au logiciel Arena. Pour plus de détails sur OptQuest, se référer à [Rockwell Software 2004, Kleijnen et Wan 2007, Eskandari *et al.* 2011].

Pour que les résultats puissent être comparés statistiquement, nous avons défini là aussi un nombre de réplications égal à 30. La durée de simulation a été établie telle que dans [Tardif et Maaseidvaag 2001] : 1 million d'unités de temps. Nous avons également repris leur fonction objectif qui peut être énoncée comme suit :

$$Minimiser\ z = \mathbb{E}[\overline{fs} + \overline{wip} + 1000 \cdot \overline{bo}] \tag{5.2}$$

où chaque variable voit son coût par unité de produit et de temps. $\overline{fs}$ correspond au stock moyen de produits finis et $\overline{wip}$ aux encours moyens, ces deux variables ayant un coût de 1. Enfin, $\overline{bo}$ correspond à la moyenne des demandes en retard dont le coût est de 1000.

Dans le cas du ConWIP traditionnel, l'optimisation du système consiste à trouver le nombre de cartes K_{Min} qui minimise les coûts exprimés par l'Équation 5.2 ci-dessus, et ce sans aucune carte supplémentaire à disposition ($K_{sup} = 0$). Le Tableau 5.1 indique les résultats obtenus, à savoir la valeur optimale de K_{Min} et la performance qui en découle pour chacun des trois scénarios définis précédemment.

	Stable	**Cyclique**	**Triangulaire**
K_{Min}	6	7	6
z	6,391456	7,354815	7,183995

Tableau 5.1 – Optimisation d'un ConWIP traditionnel

Passons maintenant à l'approche de [Tardif et Maaseidvaag 2001]. Largement reprise dans la littérature, cette approche peut être illustrée de façon générique et en accord avec notre représentation de logique décisionnelle (Figure 5.3). Elle se base sur deux seuils, l'un pour l'ajout ($Seuil_A$) et l'autre pour le retrait de cartes ($Seuil_R$). De plus, K_{Min} et

K_{sup} constituent également des variables de décision. Le Tableau 5.2 résume les résultats obtenus pour chaque scénario après l'optimisation via OptQuest.

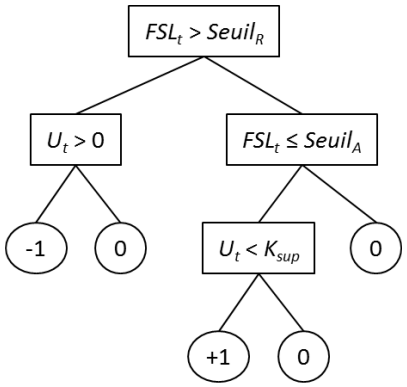

FIGURE 5.3 – Logique décisionnelle générique de [Tardif et Maaseidvaag 2001]

	Stable	**Cyclique**	**Triangulaire**
$K_{Min}/K_{sup}/Seuil_A/Seuil_R$	5/5/2/3	5/5/3/4	5/5/3/4
z	5,658755	6,260968	5,998775

Tableau 5.2 – Optimisation du ConWIP adaptatif de [Tardif et Maaseidvaag 2001]

5.2.3.3 Hypothèses

Nous nous intéressons à l'adaptation du système ConWIP que nous venons de décrire et non pas à sa conception initiale : nous supposons donc à partir de maintenant que cette dernière est connue. Entendons par là que les nombres minimum et maximum de cartes dans le système (K_{Min} et K_{Max}) ont été déterminés au préalable. Puisque le système est extrait de [Tardif et Maaseidvaag 2001] et que nous utiliserons leurs résultats à des fins comparatives, nous avons fixé K_{Min} et K_{Max} comme étant les mêmes que ceux des auteurs. Comme précisé dans la section précédente, ils définissent K_{Min} et K_{sup}, donc K_{Max} s'obtient par la relation $K_{Max} = K_{Min} + K_{sup}$.

De plus, dans l'approche qu'ils proposent, ces données constituent des variables de décision et non pas des constantes. Nous les avons donc récupérées après avoir résolu le problème d'optimisation correspondant à chacun des trois scénarios. Nous pouvons constater d'après le Tableau 5.2 que les valeurs de K_{Min} et K_{sup} sont identiques quel que soit le scénario : $K_{Min} = K_{sup} = 5$, donc $K_{Max} = 10$.

Nous pouvons maintenant nous concentrer sur notre résolution par programmation génétique linéaire, elle aussi basée sur une durée de simulation longue et s'élevant à 1 million d'unités de temps.

5.2.3.4 Formulation du problème dans le cadre de la programmation génétique linéaire

Commençons par adapter nos données aux éléments nécessaires à la PGL pour constituer une solution, à savoir les différents registres mémoire et instructions pouvant apparaître dans une logique décisionnelle (Section 4.6).

Puisque nous nous intéressons à comment déterminer les adaptations à faire par rapport au nombre de cartes dans le système, nous avons un registre de sortie unique : $T_A = \{d_t\}$. Soit d_{max} le maximum de cartes qu'il est possible d'ajouter ou de retirer du système simultanément, d_t peut prendre sa valeur dans l'ensemble $T_{CA} = \mathcal{D} = [-d_{max}; d_{max}]$ où $1 \leq d_{max} \leq K_{sup}$. Pour nos expérimentations, $d_{max} = 2$ et donc $d_t \in T_{CA} = [-2; 2]$: jusqu'à deux cartes peuvent être manipulées à la fois.

Pour satisfaire la Contrainte 5.1 sur le nombre de cartes dans le système (Section 5.2.2.1), quelques précautions se font nécessaires à propos de T_{CA}. Tel que nous venons de le définir, toutes les actions sont considérées comme étant disponibles à tout instant t. Cependant, l'ensemble d'actions valides change dans le temps. Pour prendre en compte ce changement, deux solutions « immédiates » existent : soit définir un sous-ensemble à chaque instant t contenant seulement les actions valides à cet instant, soit faire un traitement à posteriori responsable de corriger l'action proposée si besoin. Nous avons ici opté pour cette deuxième solution (pour plus de détails sur la première, se référer à l'Annexe E).

Certaines actions d_t proposées par la logique décisionnelle à l'instant t peuvent donc avoir besoin d'être corrigées. Ici, ces corrections sont faites directement dans notre modèle de simulation en changeant l'interprétation du résultat de la logique décisionnelle comme suit :

$$d_t := \begin{cases} min\{d_t; K_{Max} - a_{1,1,t}\} & \text{si } d_t > 0 \\ max\{d_t; K_{Min} - a_{1,1,t}\} & \text{si } d_t < 0 \\ d_t & \text{si } d_t = 0 \end{cases} \tag{5.3}$$

La valeur proposée est ainsi convertie en une valeur dite effective qui tient compte des valeurs les plus extrêmes autorisées à cet instant. Ainsi, chaque action d_t violant ou susceptible de violer la Contrainte 5.1 est remplacée par l'action valide la plus proche. Par exemple, si à un moment t donné l'action proposée est de retirer 2 cartes du système ($d_t = -2$) alors que seules les cartes permanentes sont utilisées ($a_{1,1,t} = K_{Min}$), aucune carte ne pourra être récupérée : $d_t := max\{-2; 0\} = 0$.

Passons maintenant aux registres mémoire d'entrée. Ils sont composés des mêmes variables d'état utilisées par [Tardif et Maaseidvaag 2001] et déjà explicitées dans la Section 5.2.2.2 : $T_V = \{FSL_t; U_t\}$. Ces variables seront comparées à des seuils pouvant prendre des valeurs dans les intervalles définis par T_{CV}. Selon [Tardif et Maaseidvaag 2001], FSL_t peut être comparé à des entiers allant jusqu'à $K_{Min} + d_{max}$. Il n'y a pas de précision concernant la borne inférieure. Nous nous basons donc sur leurs expérimentations

où elle n'est jamais inférieure à 1. Pour U_t, c'est assez intuitif : dans un extrême aucune carte supplémentaire n'est utilisée et il vaut 0, dans l'autre elles le sont toutes et il vaut K_{sup}. Ainsi, $T_{CV} = \{[1;7];[0;5]\}$ où le premier intervalle est associé à FSL_t tandis que le deuxième à U_t.

Nous avons donc l'ensemble de registres mémoire constants $T_C = T_{CV} \cup T_{CA} = \{[1;7];[0;5];[-2;2]\}$ où le dernier intervalle spécifie le domaine de la variable de décision dans T_A.

Une fois les registres précisés et en considérant les différents comparateurs possibles, nous avons l'ensemble suivant d'instructions :

- $d_t = 0$
- if (FSL_t $\{=;\neq;>;\geq;<;\leq\}$ $\{[1\,;\,7]\}$) jumpTo *label*
- if (FSL_t $\{=;\neq;>;\geq;<;\leq\}$ $\{[1\,;\,7]\}$) $d_t = \{[\text{-}2\,;\,2]\}$
- if (U_t $\{=;\neq;>;\geq;<;\leq\}$ $\{[0\,;\,5]\}$) jumpTo *label*
- if (U_t $\{=;\neq;>;\geq;<;\leq\}$ $\{[0\,;\,5]\}$) $d_t = \{[\text{-}2\,;\,2]\}$

où la forme que prend l'affectation pure définit « ne rien faire » comme action par défaut.

Nous souhaitons cependant nous rapprocher au plus de ce qui est pris en compte par [Tardif et Maaseidvaag 2001]. Ainsi, quelques adaptations sont faites et l'ensemble effectivement utilisé est le suivant :

- $d_t = 0$
- if (FSL_t $\{>;\geq\}$ 7) jumpTo *label*
- if (FSL_t $\{>;\geq\}$ 7) $d_t = \{[\text{-}2\,;\,0]\}$
- if (FSL_t $\{<;\leq\}$ 1) jumpTo *label*
- if (FSL_t $\{<;\leq\}$ 1) $d_t = \{[0\,;\,2]\}$
- if (FSL_t $\{>;\geq;<;\leq\}$ $\{[2\,;\,6]\}$) jumpTo *label*
- if (FSL_t $\{>;\geq;<;\leq\}$ $\{[2\,;\,6]\}$) $d_t = \{[\text{-}2\,;\,2]\}$
- if ($U_t > \{[0\,;\,1]\}$) jumpTo *label*
- if ($U_t > \{[0\,;\,1]\}$) $d_t = \{[\text{-}2\,;\,2]\}$
- if ($U_t < \{[4\,;\,5]\}$) jumpTo *label*
- if ($U_t < \{[4\,;\,5]\}$) $d_t = \{[\text{-}2\,;\,2]\}$

Nous avons ainsi des sous-ensembles de T_{CA} ainsi que de chaque intervalle dans T_{CV}. Lorsque le niveau du stock est élevé (supérieur ou égal à la borne supérieure considérée), seules les actions de ne rien faire ou de retirer des cartes sont envisagées. Dans l'extrême opposé, lorsque le niveau tend vers la rupture, les actions possibles sont de ne rien faire ou d'ajouter des cartes. Quant au nombre de cartes supplémentaires utilisées, seuls les extrêmes de l'intervalle sont vérifiés (avec une marge de $d_{max} - 1$).

D'autres paramètres sont également nécessaires au fonctionnement d'un algorithme de programmation génétique linéaire. Nous trouverons dans le Tableau 5.3 ceux qui sont

communs à nos différentes expérimentations, lesquels ont été empiriquement déterminés. Ils concernent les tailles minimale et maximale des individus ainsi que la moyenne et l'écart-type servant à l'initialisation aléatoire. Rappelons que la taille fait ici référence au nombre d'instructions composant une logique décisionnelle.

Tailles d'individus	Valeur
Minimale	5
Maximale	50
Moyenne	10
Écart-type	25

Tableau 5.3 – Paramètres concernant les individus pour le problème du ConWIP

Les autres paramètres pour lesquels la notation a été introduite dans la Section 4.8.2.3 varient selon l'expérimentation. Nos expérimentations seront regroupées en différents groupes de test, quelques-unes en faisant partie de plusieurs à la fois. Pour faciliter leur assimilation, elles seront présentées dans la section suivante dédiée aux résultats, et ce au fur et à mesure de leur utilisation. Elles sont également fournies de façon détaillée dans l'Annexe F.

Rappelons que les probabilités d'activation de nos 7 opérateurs génétiques décrits dans le chapitre précédent ne figureront pas parmi les paramètres des tableaux qui suivront puisqu'elles démarrent à égalité ($\frac{1}{7}$) pour être ensuite auto-adaptées au cours de l'évolution (Sections 4.7.3 et 4.8.2.2).

5.2.4 Résultats

5.2.4.1 Population et critères d'arrêt

5.2.4.1.1 Paramètres de la PGL

Pour notre premier groupe de test, nous définissons l'immortalité comme inactive et la taille de tournoi $V_{tournoi}$ à 2. Nous utilisons également un certain nombre de relations entre paramètres comme illustré dans le Tableau 5.4.

Paramètres	Valeurs
V_{elite}	$0,1 \cdot V_{max}$
$NbOp$	V_{max}
P_{stagne}	$0,5 \cdot P_{max}$

Tableau 5.4 – Relations entre paramètres de la PGL pour le problème du ConWIP

Il nous reste donc à définir la taille V_{max} de la population et le nombre maximum P_{max} de générations. Nous considérons deux valeurs possibles pour chacun d'entre eux :

$V_{max} \in \{20; 30\}$ et $P_{max} \in \{50; 100\}$. Par leurs différentes combinaisons, nous obtenons les quatre configurations résumées dans le Tableau 5.5 ci-dessous.

Paramètres	$Config^{--}$	$Config^{++}$	$Config^{-+}$	$Config^{+-}$
V_{max}	20	30	20	30
P_{max}	50	100	100	50

Tableau 5.5 – Configurations de base utilisées pour le problème du ConWIP

5.2.4.1.2 Convergence et temps de calcul

L'évolution de la meilleure solution de la population selon les différentes configurations est présentée dans les Figures 5.4, 5.5 et 5.6, chacune reprenant l'un des trois scénarios : stable, cyclique et triangulaire. Notons cependant que $Config^{--}$ suivra toujours le même profil que $Config^{-+}$ mais en s'arrêtant plus tôt : entre les deux, seul change la valeur des critères d'arrêt. Il en est de même pour $Config^{+-}$ vis-à-vis de $Config^{++}$. Pour une meilleure visualisation, les courbes pour $Config^{--}$ et $Config^{+-}$ ne font pas partie des figures, nous indiquons simplement leur point d'arrêt.

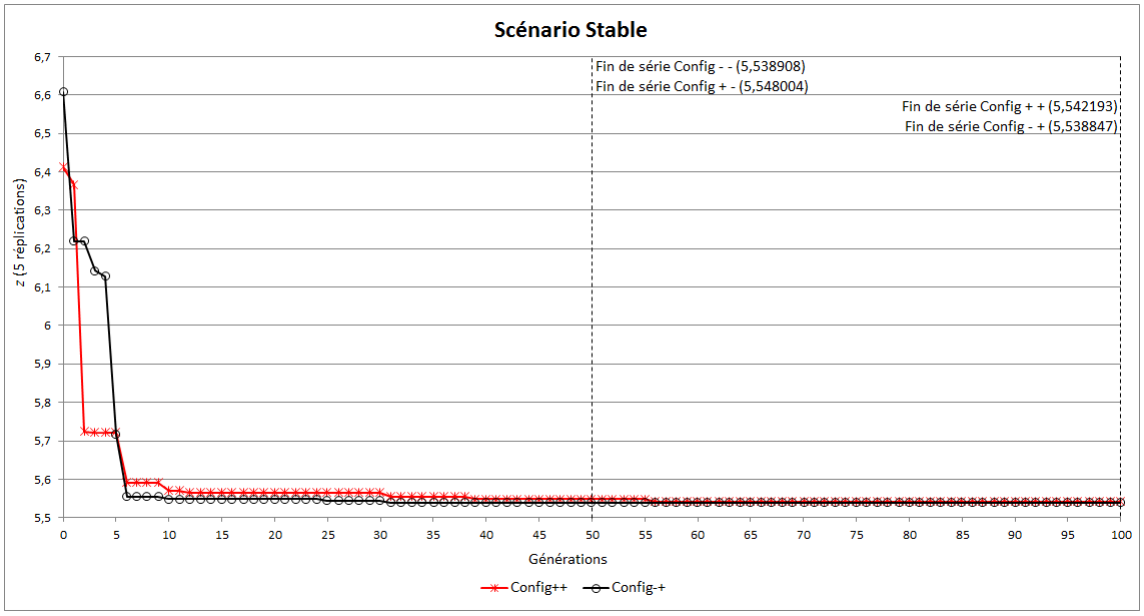

FIGURE 5.4 – Courbes d'apprentissage pour le scénario stable

Commençons par constater que les populations initiales obtiennent des performances relativement mauvaises. Ceci est parfaitement logique de par leur construction purement aléatoire. Ensuite, ces courbes d'apprentissage montrent toutes une bonne convergence, les résultats étant rapidement améliorés en quelques générations. La PGL a donc été capable d'améliorer considérablement sa propre capacité à prendre de bonnes décisions, et ce tout au long de l'évolution. Les configurations avec une plus grande taille de population ($Config^{++}$ et $Config^{+-}$) convergent plus vite mais peut-être de façon prématurée

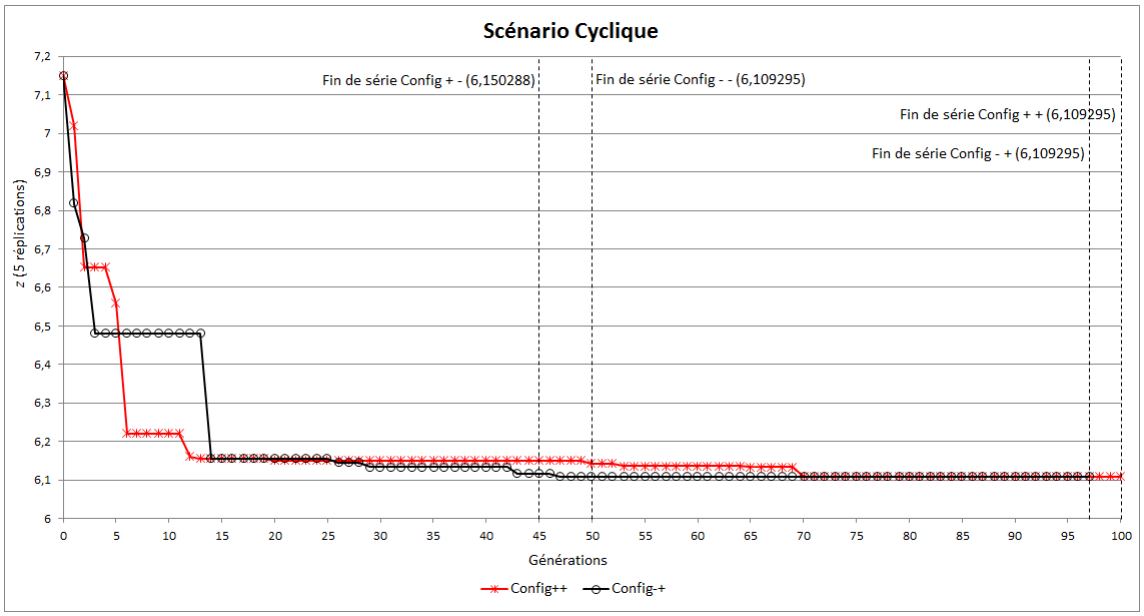

FIGURE 5.5 – Courbes d'apprentissage pour le scénario cyclique

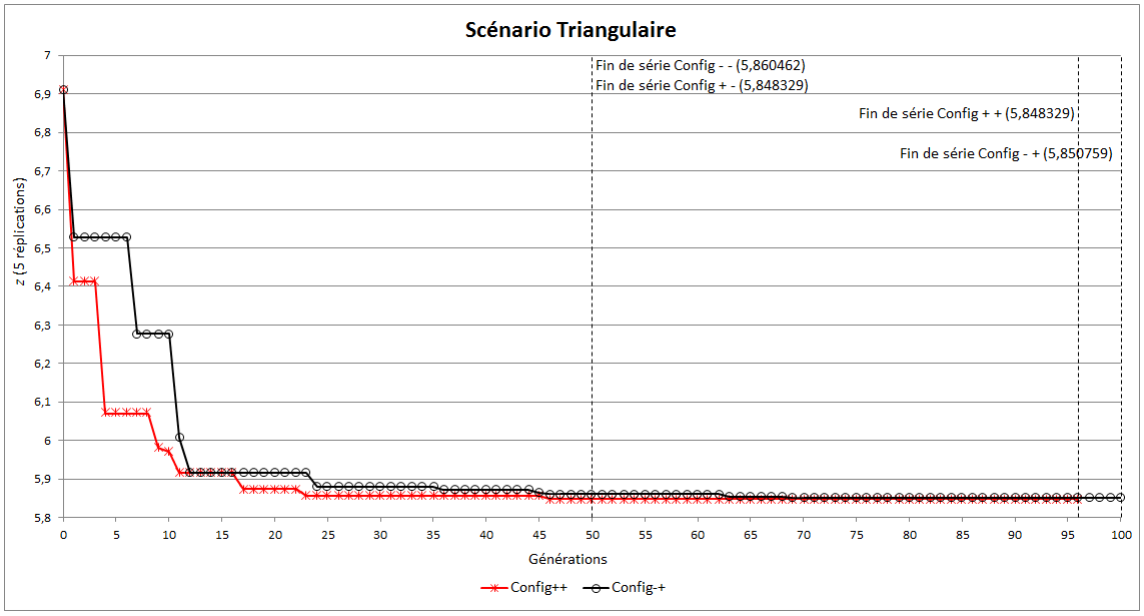

FIGURE 5.6 – Courbes d'apprentissage pour le scénario triangulaire

puisque, nous le verrons, ce sont les autres ($Config^{--}$ et $Config^{-+}$) qui atteignent les meilleurs résultats. Nous pouvons également constater que l'apprentissage s'est arrêté au nombre maximum de générations dans 9 cas sur 12. Pour les trois autres, un nombre trop élevé de générations sans améliorations a entraîné leur arrêt prématuré (il leur manquait seulement jusqu'à 5 générations pour arriver au maximum).

Cette analyse peut être complétée par le temps de calcul nécessaire à notre approche d'apprentissage pour l'obtention d'un résultat (bien que les courbes montrent que l'on aurait pu l'arrêter plus tôt). Les valeurs (en heures) sont disponibles dans les colonnes

Temps du Tableau 5.6. Nous avons dans le meilleur des cas un apprentissage conclu en 6,66 heures et dans le pire des cas en 1,15 jour, ce qui n'est pas négligeable. Les longs temps de calcul semblent cependant inévitables pour les approches basées sur la simulation. Néanmoins, il est important de signaler que ce temps correspond à l'étape de découverte de la logique décisionnelle, il est donc appliqué hors ligne. Pour le pilotage réel (en ligne) des adaptations, seul importe le temps requis pour utiliser la logique décisionnelle retenue. Pour avoir un ordre d'idée de ce temps, nous indiquons le nombre total de solutions évaluées dans la colonne *NbEval*. L'évaluation de chaque solution au cours du processus d'apprentissage requiert donc un temps moyen de 20,53 secondes. Puisqu'il correspond au temps nécessaire pour exécuter 5 réplications du modèle de simulation sur 1 million d'unités de temps, nous pouvons conclure que les décisions d'adaptation en ligne sont faites quasi-instantanément. La logique décisionnelle obtenue convient donc tout à fait à une telle utilisation.

Configurations	**Stable**		**Cyclique**		**Triangulaire**	
	Temps (h)	*NbEval*	*Temps (h)*	*NbEval*	*Temps (h)*	*NbEval*
$Config^{++}$	22,06	4854	22,77	4644	27,71	4623
$Config^{--}$	6,66	1385	7,23	1458	9,00	1444
$Config^{+-}$	12,78	2349	11,63	1950	16,03	2335
$Config^{-+}$	15,92	2899	16,30	2966	23,09	2983

Tableau 5.6 – Temps de calcul et nombre de solutions évaluées par les différentes configurations de la PGL pour le problème du ConWIP

Ajoutons que depuis notre publication [Belisário et Pierreval 2015], nos temps de calcul ont été divisés par 3 suite à l'amélioration de notre algorithme.

5.2.4.1.3 Comparaisons quantitatives

La pertinence d'une méthode se mesure aussi et surtout par rapport à sa performance en termes de *fitness*. Le Tableau 5.7 fournit les *fitness* obtenus pour chaque combinaison <configuration, scénario> (Équation 5.2, Section 5.2.3.3). Comme nous l'avons indiqué précédemment, chaque valeur correspond ici à la moyenne obtenue sur 30 réplications. Les deux premières lignes identifiées par CT et TM reprennent les résultats pour le ConWIP traditionnel et pour l'approche de [Tardif et Maaseidvaag 2001] respectivement (Tableaux 5.1 et 5.2, Section 5.2.3.2).

Pour mieux évaluer ces résultats quantitatifs, comparons les performances de notre approche à celles du ConWIP traditionnel. Nos résultats mettent en avant l'intérêt d'une méthode adaptative puisque pour chaque scénario, le *fitness* obtenu est toujours significativement plus bas (donc meilleur) que pour un système à nombre de cartes fixe (tests t de Student). De plus, ce résultat est vérifié même pour le cas où le taux moyen de demande reste constant. Cela montre que l'adaptabilité est utile même pour répondre à

Configurations	Stable	Cyclique	Triangulaire
CT	6,391456	7,354815	7,183995
TM	5,658755	6,260968	5,998775
$Config^{++}$	5,555161	6,168619	5,888884
$Config^{--}$	5,553611	6,168619	5,886490
$Config^{+-}$	5,561377	6,176671	5,888884
$Config^{-+}$	5,553596	6,168619	5,878418

Tableau 5.7 – *Fitness* obtenus pour le problème du ConWIP

des changements dus uniquement au hasard. Ces résultats sont en accord avec ceux déjà observés dans la littérature à travers d'autres méthodes adaptatives [Gupta et Al-Turki 1997, Tardif et Maaseidvaag 2001, Sivakumar et Shahabudeen 2008].

Comparons également nos performances à celles de l'approche de [Tardif et Maaseidvaag 2001]. Encore une fois et d'après des tests t de Student, nos *fitness* sont significativement meilleurs. Ces résultats confirment le potentiel de notre approche pour atteindre de bonnes performances. D'après les résultats de nos douze expérimentations, le gain estimé oscille entre 1,35% et 2,01% pour une valeur moyenne de 1,72%. Notons qu'il est plus faible pour le scénario cyclique. Ces valeurs découlent de la formule suivante : $\frac{z[TM]-z[PGL]}{z[TM]}$ où $z[TM]$ indique la performance de la méthode de [Tardif et Maaseidvaag 2001] pour un scénario donné alors que $z[PGL]$ correspond à celle d'une expérimentation donnée d'après notre approche pour ce même scénario.

5.2.4.1.4 Analyse qualitative

Passons maintenant à une analyse qualitative de nos résultats. Elle sera faite sur la base de la meilleure solution obtenue pour chaque scénario. Les Figures 5.7, 5.8 et 5.9 illustrent chaque solution directement sous sa forme d'arbre. Elles ne représentent que le code effectif des solutions qui sont ainsi plus intelligibles. Par ailleurs, tous les arbres de ce chapitre ont été simplifiés comme expliqué dans la Section 4.6.3.3. À noter que nous illustrerons toutes nos solutions directement par l'arbre binaire résultant. Cependant et à titre illustratif, un exemple de programme généré par μGP est disponible dans l'Annexe G.

Nous allons nous concentrer sur le scénario cyclique pour lequel trois configurations sur quatre ont atteint la même valeur de *fitness* (Tableau 5.7). Malgré les différences apparentes des arbres de la Figure 5.8, ceux-ci conduisent en réalité à la même décision quelle que soit l'entrée. Rappelons qu'avoir plusieurs représentations d'une même solution en PG-PGL est connu et que les redondances ou code non effectif permettant cette variété sont considérés comme utiles (Sections 4.5.2 et 4.6.3).

Notons que dans les arbres ci-dessus, certaines actions suggérées (en gras dans les

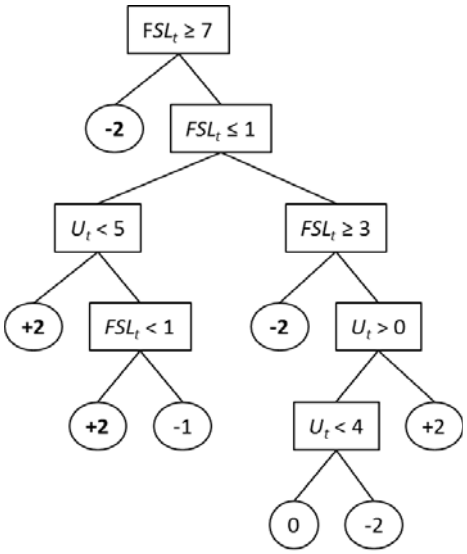

FIGURE 5.7 – Meilleure solution pour le scénario stable

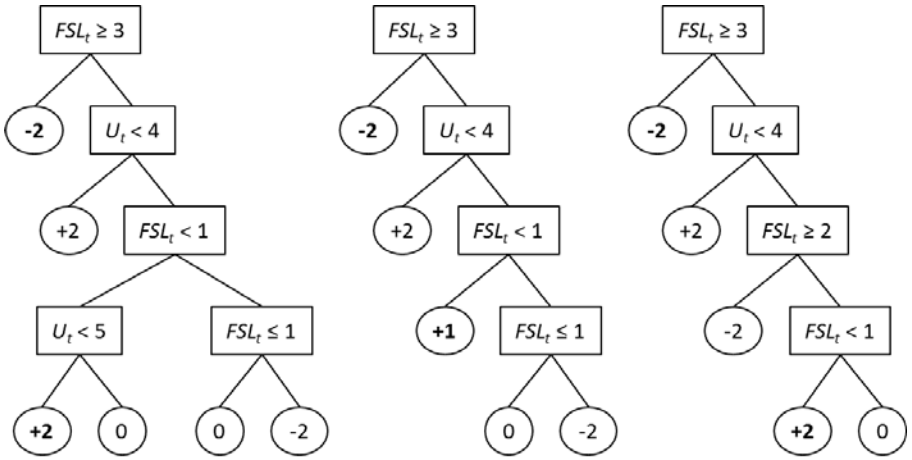

FIGURE 5.8 – Meilleures solutions pour le scénario cyclique

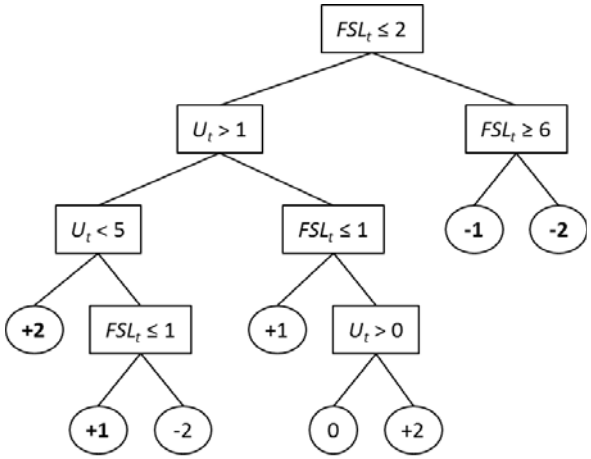

FIGURE 5.9 – Meilleure solution pour le scénario triangulaire

figures) ne respectent pas forcement la Contrainte 5.1. Pour le scénario triangulaire par exemple, l'ajout d'une carte est impossible lorsque toutes les cartes supplémentaires sont déjà utilisées. De même, lorsqu'aucune analyse n'est faite sur le nombre de cartes supplémentaires utilisées, le retrait d'une ou de deux cartes pourrait conduire à une violation de contrainte. Nous pouvons cependant apporter des corrections manuelles assez facilement ou même les intégrer automatiquement dans un algorithme exécuté avant la livraison du résultat afin que l'utilisateur n'ait pas à le faire lui-même. Ici, ces solutions sont corrigées comme expliqué dans la Section 5.2.3.4 et un exemple d'arbre final après corrections (en gras) est donné dans la Figure 5.10.

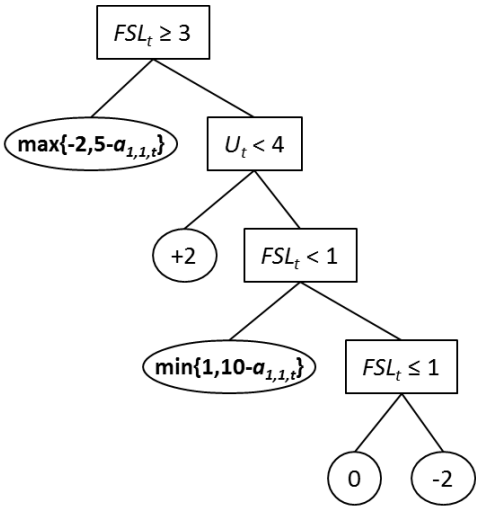

FIGURE 5.10 – Meilleure solution pour le scénario cyclique après correction

À partir des résultats numériques, notre approche produit en effet de bons résultats, meilleurs que ceux de [Tardif et Maaseidvaag 2001]. Rappelons toutefois que la difficulté majeure de la problématique d'adaptation des systèmes de production vient du manque d'expertise (ou d'exemples à partir desquels la construire). Par conséquent, nous souhaitons évaluer si des connaissances ont effectivement été générées. Autrement dit, nous voulons vérifier si les performances obtenues sont justifiables et non pas le fruit du hasard. Pour ce faire, nous analyserons le contenu de nos solutions, c'est-à-dire la cohérence et la fiabilité du raisonnement représenté par ces arbres.

Tout d'abord, constatons que notre approche obtient des logiques décisionnelles d'un plus grand niveau de détail que celle de [Tardif et Maaseidvaag 2001] illustrée par la Figure 5.3. Nous allons voir que cet apport d'informations est pertinent et que nos arbres constituent une connaissance plus complète du problème.

En analysant les arbres, nous voyons immédiatement que nous n'avons pas que deux seuils, l'un pour l'ajout et l'autre pour le retrait de cartes. Nous pouvons analyser certaines situations neutres, intermédiaires ou extrêmes afin de déclencher différentes actions. Lorsque le niveau de stock final est considéré comme trop élevé ou au contraire trop bas, nous pouvons retirer ou ajouter deux cartes supplémentaires à la fois, ce qui s'avère être

pertinent comme information. Toutefois, si la situation n'est pas si extrême, une seule carte est impliquée, ce qui est compréhensible car le risque de se retrouver en excédent ou rupture de stock est plus limité.

La redondance autorisée dans notre approche a permis d'identifier différents seuils et ainsi de traiter des cas plus spécifiques. Mais par-dessus tout, ces cas ont pu être exploités puisque contrairement à [Tardif et Maaseidvaag 2001], nous avons permis la manipulation de deux cartes à la fois. Cela a bien été « compris » par l'algorithme d'apprentissage qui a identifié par lui-même des situations où il était pertinent de le faire. En intégrant ces situations dans la logique, il a par conséquent généré de la connaissance.

En somme et grâce à la simulation, la PGL a été capable d'intégrer correctement des considérations supplémentaires non exploitées par [Tardif et Maaseidvaag 2001], d'où son intérêt. Cet apport d'information explique par ailleurs sa meilleure performance. Nous obtenons donc une expertise construite artificiellement dans un format simple, interprétable et utilisable directement par des gestionnaires.

Quelques tests préliminaires seront également faits pour analyser l'influence des deux autres paramètres de la PGL jusqu'alors fixes : la taille du tournoi et l'immortalité.

5.2.4.2 Sélection par tournoi

5.2.4.2.1 Paramètres de la PGL

Parmi les différentes expérimentations qui suivent, nous testerons la fonctionnalité de μGP d'auto-adapter la taille du tournoi utilisée pour la sélection, c'est-à-dire, la flexibilité pour la valeur de $V_{tournoi}$.

Nous considérons 5 valeurs possibles pour la taille du tournoi : 1 (pour une sélection complètement aléatoire), 2, 3, 4 ou encore l'intervalle $[2\,;4]$. Ce dernier, représenté par Δ, indique les valeurs minimale et maximale de $V_{tournoi}$ qui dans ce cas sera auto-adaptée.

Pour ce qui est des autres paramètres, nous nous baserons sur deux des configurations précédentes : $Config^{++}$ ($V_{max} = 30$, $P_{max} = 100$) et $Config^{--}$ ($V_{max} = 20$, $P_{max} = 50$) pour lesquelles nous avons des caractéristiques communes reprises dans le Tableau 5.8.

Paramètres	**Valeurs**
Immortalité	Non
V_{elite}	$0,1 \cdot V_{max}$
$NbOp$	V_{max}
P_{stagne}	$0,5 \cdot P_{max}$

Tableau 5.8 – Paramètres communs de la PGL lors des tests sur la taille de tournoi

Les configurations seront donc notées $Config^{++V_{tournoi}}$ et $Config^{--V_{tournoi}}$ selon que

la configuration de base soit $Config^{++}$ ou $Config^{--}$. Si $V_{tournoi} \neq 2$, elle sera remplacée par sa valeur ; sinon elle sera omise car nous retrouvons les configurations de base déjà utilisées.

5.2.4.2.2 Résultats quantitatifs

Les résultats sont présentés dans le Tableau 5.9 qui suit. Notons que dans sa première ligne (TM), il récapitule les performances de l'approche de [Tardif et Maaseidvaag 2001] servant de base pour nos comparaisons (Tableau 5.2).

Configurations	Stable	Cyclique	Triangulaire
TM	5,658755	6,260968	5,998775
$Config^{++1}$	5,560293	6,115379	5,855367
$Config^{++}$	5,555161	6,168619	5,888884
$Config^{++3}$	5,559392	6,120585	5,886869
$Config^{++4}$	5,558621	6,106146	5,898354
$Config^{++\Delta}$	5,560293	6,118976	5,895059
$Config^{--1}$	5,559422	6,753386	5,858623
$Config^{--}$	5,553611	6,168619	5,886490
$Config^{--3}$	5,602780	6,114837	5,894406
$Config^{--4}$	5,556173	6,123752	5,871350
$Config^{--\Delta}$	5,558538	6,090211	5,895059

Tableau 5.9 – Résultats pour le ConWIP en faisant varier la taille du tournoi

Ici encore, 29 performances sur 30 restent supérieures à celles de [Tardif et Maaseidvaag 2001]. La seule exception est le scénario cyclique combiné à la configuration $Config^{--1}$. Elle peut s'expliquer en partie par la sélection des parents de façon purement aléatoire. Pour le scénario triangulaire, cela a été bénéfique et a permis de meilleures performances. Pour le scénario cyclique, c'est le contraire. Ceci illustre bien les effets contradictoires apparaissant avec le hasard. Quant à l'utilisation d'une taille de tournoi qui s'auto-adapte, les résultats montrent qu'elle procure une performance comparable aux autres, parfois meilleure, parfois moins bonne. C'est donc une option à retenir pour simplifier le paramétrage de l'algorithme de programmation génétique linéaire.

5.2.4.3 Immortalité

5.2.4.3.1 Paramètres de la PGL

Nous avons défini ici deux groupes de test. Pour le premier, nous nous sommes basés sur la configuration $Config^{-+}$ détaillée dans le Tableau 5.10.

Paramètres	Valeurs
V_{max}	20
V_{elite}	$0,1 \cdot V_{max}$
P_{max}	100
P_{stagne}	$0,5 \cdot P_{max}$
$V_{tournoi}$	2

Tableau 5.10 – Paramètres communs de la PGL lors des tests sur l'immortalité

Pour l'analyse de l'immortalité, nous avons donc joué sur l'activation ou non de ce paramètre et fait varier le nombre d'opérations appliquées entre deux générations comme explicité dans le Tableau 5.11. L'idée est ici de tester ce qui est d'usage lorsque l'immortalité n'est pas activée, c'est-à-dire $NbOp$ variant de V_{max} à $2 \cdot V_{max}$ (Section 4.8.2.3). Lorsque l'immortalité est activée, nous pouvons compter sur les individus de la population de parents et donc générer moins d'enfants. Nous testons dans ce cas $NbOp$ variant de $\frac{V_{max}}{2}$ à V_{max}. Une valeur intermédiaire aux bornes a été déterminée dans chaque cas.

Paramètres	$Config^{-+}$	$Config^{M30}$	$Config^{M40}$	$Config^{I10}$	$Config^{I15}$	$Config^{I20}$
Immortalité	Non	Non	Non	Oui	Oui	Oui
$NbOp$	20	30	40	10	15	20

Tableau 5.11 – Configurations utilisées pour tester l'immortalité

Pour le deuxième groupe de test, nous avons introduit une nouvelle configuration $Config^{*+}$ qui compte une population de 100 individus. Cependant, en raison du temps de calcul plus important pour un tel nombre d'individus, nous l'avons comparée à une seule configuration différente. Pour cette même raison, $NbOp$ a été pour toutes les deux fixé au minimum : V_{max} si l'immortalité est activée et $\frac{V_{max}}{2}$ sinon. Pour ces deux configurations du Tableau 5.12, la taille de l'élite correspond à 15% de V_{max} et les autres paramètres coïncident avec ceux que nous venons d'énumérer dans le Tableau 5.10 pour la configuration $Config^{-+}$.

Paramètres	$Config^{*+}$	$Config^{*+I}$
Immortalité	Non	Oui
$NbOp$	100	50

Tableau 5.12 – D'autres configurations utilisées pour tester l'immortalité : Population de 100 individus

5.2.4.3.2 Résultats quantitatifs

Les résultats pour ces différentes configurations, y compris celle de base, sont détaillés dans le Tableau 5.13. Encore une fois, la première ligne (TM) récapitule les performances de référence (approche de [Tardif et Maaseidvaag 2001]).

Configurations	Stable	Cyclique	Triangulaire
TM	5,658755	6,260968	5,998775
$Config^{-+}$	5,553596	6,168619	5,878418
$Config^{M30}$	5,582117	6,120693	5,878286
$Config^{M40}$	5,558557	6,133769	5,895640
$Config^{I10}$	5,604077	6,148605	5,866619
$Config^{I15}$	5,557353	6,104146	5,936879
$Config^{I20}$	5,560293	6,142583	5,883247
$Config^{*+}$	5,560918	6,151302	5,873985
$Config^{*+I}$	5,554528	6,120404	5,890914

Tableau 5.13 – Résultats pour le ConWIP en faisant varier l'immortalité et le nombre d'opérations à appliquer

L'analyse n'est ici pas concluante : il n'y pas de configuration dominante (meilleure pour tous les scénarios) et il n'y a pas non plus de configuration dominée (pire pour tous). En considérant la moyenne générale (les deux groupes de test compris), il y a une légère préférence (estimée à 0,02%) pour les configurations où l'immortalité n'est pas activée (5,862992 contre 5,864137). Quant à la moyenne sur tous les scénarios pour une configuration donnée, la $Config^{*+I}$ s'avère la meilleure (5,855282) malgré l'activation de l'immortalité. $Config^{I10}$ est quant à elle la pire (5,873100). Mais là encore la différence estimée n'est pas importante (0,30%).

Une nouvelle fois, les performances restent significativement supérieures à celles de [Tardif et Maaseidvaag 2001] (d'après des tests t de Student), ce qui appuie la robustesse de notre approche. Toutefois, ces résultats n'écartent pas l'intérêt d'une analyse plus poussée sur la sensibilité de l'approche vis-à-vis des différents paramètres, analyse basée sur la définition de plans d'expériences.

5.2.4.4 Scénario mixte

5.2.4.4.1 Description du scénario

Signalons que pour toutes les expérimentations et analyses faites jusqu'alors, nous ne visions pas la découverte d'une logique décisionnelle universelle. Entendons par ce terme une logique générale qui serait pertinente pour différents systèmes dans différents environnements (configuration, conditions de fonctionnement et objectifs). Ainsi, même si des similarités peuvent être identifiées entre les arbres des trois scénarios (Figures 5.7, 5.8 et 5.9), le but était d'en découvrir un qui soit efficace pour chaque scénario.

Il nous semble tout de même intéressant et important de vérifier la pertinence de l'approche lorsqu'elle est cette fois-ci confrontée à plusieurs scénarios simultanément. Nous ne chercherons donc plus à trouver une logique décisionnelle qui convienne le mieux à

chacun des trois scénarios précédents mais une qui leur conviendra à tous.

Sachant que dans un contexte incertain il ne nous est pas possible de prévoir comment la demande (ou d'autres facteurs) évoluera dans le temps, un nouveau scénario est proposé pour un apprentissage dans des situations plus variées. Pour ce faire, nous nous basons toujours sur le même système ConWIP mais en alternant de façon aléatoire l'occurrence de nos trois *patterns*, chacune ayant une probabilité d'occurrence de $\frac{1}{3}$. Ceci se traduit dans le modèle de simulation par un tirage au sort indiquant la *pattern* devant être appliquée pendant 150 unités de temps. À la fin de cette période, un nouveau tirage au sort est effectué et ainsi de suite jusqu'à la fin du million d'unités de temps de simulation. Nous désignerons par mixte cette nouvelle *pattern*, tout comme le scénario associé.

5.2.4.4.2 Paramètres de la PGL

Pour cette étude, nous nous sommes basés sur les trois configurations précédentes de la PGL ayant les critères d'arrêt les plus élevées ($P_{max} = 100$) : $Config^{-+}$, $Config^{++}$ et $Config^{*+}$. Leurs différences sont détaillées dans le Tableau 5.14 tandis que le Tableau 5.15 présente un récapitulatif des autres paramètres que ces configurations utilisent.

Paramètres	$Config^{-+}$	$Config^{++}$	$Config^{*+}$
V_{max}	20	30	100
V_{elite}	2	3	15

Tableau 5.14 – Configurations de la PGL utilisées pour le scénario mixte

Paramètres	**Valeurs**
Immortalité	Non
$NbOp$	V_{max}
P_{stagne}	$0,5 \cdot P_{max}$
$V_{tournoi}$	2

Tableau 5.15 – Paramètres communs de la PGL lors des tests sur le scénario mixte

5.2.4.4.3 Résultats quantitatifs et qualitatifs

Les résultats sont présentés dans le Tableau 5.16. Rappelons que nous utilisons pour l'apprentissage le scénario mixte où les différentes *patterns* coexistent de sorte à obtenir une logique décisionnelle plus générale (pour des situations plus variées). La colonne « Mixte » indique les performances obtenues pour ce scénario d'apprentissage après 30 réplications. Les colonnes suivantes montrent quant à elles la performance obtenue en appliquant individuellement la logique décisionnelle provenant du scénario mixte à chacun des scénarios stable, cyclique et triangulaire. À titre de comparaison, les deux premières lignes du tableau indiquent les performances obtenues par le ConWIP traditionnel et la

méthode de [Tardif et Maaseidvaag 2001] respectivement. Pour ces lignes en particulier, la colonne « Configuration » correspond soit à K_{Min} soit à $K_{Min}/K_{sup}/Seuil_A/Seuil_R$, dans les deux cas obtenus par une optimisation sous le scénario mixte. Elles suivent donc la même logique, la même valeur des variables de décision étant appliquée individuellement à chaque scénario.

Configurations	Mixte	Stable	Cyclique	Triangulaire
6	7,042530	6,391456	7,792743	7,183995
5/5/3/4	6,007600	5,777657	6,260968	5,998775
$Config^{-+}$	5,868909	5,620432	6,142237	5,874399
$Config^{++}$	5,864886	5,564830	6,180428	5,878257
$Config^{*+}$	5,866137	5,622226	6,143029	5,878825

Tableau 5.16 – Résultats pour le ConWIP avec une logique décisionnelle basée sur le scénario mixte

Il apparaît clairement que nos résultats sont les plus performants, ce qui est d'ailleurs confirmé par des tests t de Student.

En comparant nos résultats à ceux d'un apprentissage sur un scénario spécifique (revoir Tableau 5.7), certaines performances sont en retrait (scénario stable par exemple). Cependant cette différence était prévisible en raison du caractère maintenant plus général (et donc moins spécifique) de la logique décisionnelle. Paradoxalement, d'autres performances résultant d'une logique basée sur ce scénario mixte surpassent celles provenant d'un apprentissage basé sur le scénario spécifique correspondant. C'est le cas des scénarios cyclique et triangulaire qui obtiennent leur meilleure performance ici avec la $Config^{-+}$. Le facteur stochastique de notre algorithme pourrait expliquer en partie ce phénomène mais une analyse plus approfondie s'avérerait intéressante et pourrait faire l'objet de travaux futurs.

L'arbre correspondant à la meilleure performance sur le scénario d'apprentissage est illustré par la Figure 5.11. Le programme associé à cet arbre et tel que livré par μGP est quant à lui fourni dans l'Annexe G à titre d'exemple.

5.2.4.5 Ajout de connaissance experte à la construction d'une logique décisionnelle

5.2.4.5.1 Motivation

Nous cherchons à acquérir des connaissances pouvant aider à l'adaptation d'un système de production lorsque, par manque de données, nous ne sommes pas en mesure de le faire par nous-mêmes. Pourtant et très souvent, une expertise partielle, même la plus minime qui soit, existe malgré la difficulté du problème. Il serait judicieux de pouvoir l'intégrer à notre approche de génération de connaissances et d'être en mesure de combiner

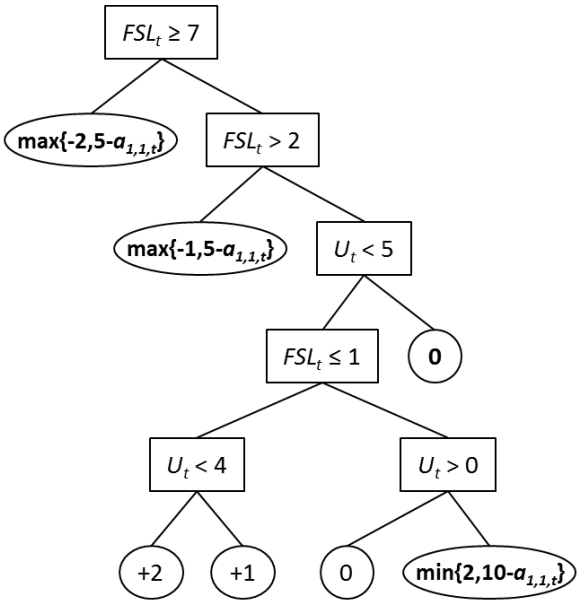

FIGURE 5.11 – Meilleure solution pour le scénario mixte

connaissances expertes et apprises. Les premières font référence à celles connues à priori, les secondes à celles extraites par la technique d'apprentissage (ici la PGL).

En reprenant le problème du ConWIP ici traité, nous possédons quelques connaissances, plus précisément par rapport à l'une des variables d'état manipulées. Nous savons que si toutes les cartes supplémentaires sont utilisées, nous ne pouvons plus en ajouter au système, les actions étant dans ce cas restreintes à [-2 ; 0]. En suivant ce même raisonnement pour les autres valeurs de U_t, nous avons :

- Si $U_t = 5$, alors $d_t \in [\ \ 2; 0]$ (situation que nous venons de décrire) ;
- Si $U_t = 4$, alors $d_t \in [\ \ 2; 1]$;
- Si $2 \leq U_t \leq 3$, alors $d_t \in [\ \ 2; 2]$ (ici rien ne change) ;
- Si $U_t = 1$, alors $d_t \in [\ \ 1; 2]$;
- Si $U_t = 0$, alors $d_t \in [0; 2]$.

5.2.4.5.2 Adaptation des conditions d'expérimentation

L'intégration d'une telle connaissance experte au développement de nos solutions peut être concrétisée par le développement non pas d'une logique décisionnelle intégrale mais de plusieurs sous-parties, une pour chacune des 5 situations identifiées ci-dessus. Autrement dit, il s'agit de fixer des « bouts d'arbre ». Ces sous-parties doivent définir les instructions disponibles selon leur besoin respectif :

- Instructions communes à toutes les sous-parties :
  $d_t = 0$
  if (FSL_t {$>; \geq$} 7) jumpTo *label*

if (FSL_t $\{<;\leq\}$ 1) jumpTo *label*
if (FSL_t $\{>;\geq;<;\leq\}$ $\{[2\,;\,6]\}$) jumpTo *label*

- Instructions spécifiques pour « if ($U_t = 0$) » :
  if (FSL_t $\{>;\geq\}$ 7) $d_t = \mathbf{0}$
  if (FSL_t $\{<;\leq\}$ 1) $d_t = \{[0\,;\,2]\}$
  if (FSL_t $\{>;\geq;<;\leq\}$ $\{[2\,;\,6]\}$) $d_t = \{[\mathbf{0}\,;\,2]\}$
- Instructions spécifiques pour « if ($U_t = 1$) » :
  if (FSL_t $\{>;\geq\}$ 7) $d_t = \{[\mathbf{-1}\,;\,0]\}$
  if (FSL_t $\{<;\leq\}$ 1) $d_t = \{[0\,;\,2]\}$
  if (FSL_t $\{>;\geq;<;\leq\}$ $\{[2\,;\,6]\}$) $d_t = \{[\mathbf{-1}\,;\,2]\}$
- Instructions spécifiques pour « if ($2 \leq U_t \leq 3$) » :
  if (FSL_t $\{>;\geq\}$ 7) $d_t = \{[-2\,;\,0]\}$
  if (FSL_t $\{<;\leq\}$ 1) $d_t = \{[0\,;\,2]\}$
  if (FSL_t $\{>;\geq;<;\leq\}$ $\{[2\,;\,6]\}$) $d_t = \{[-2\,;\,2]\}$
  if (U_t = {[2 ; 3]}) jumpTo *label*
  if (U_t = {[2 ; 3]}) d_t = {[-2 ; 2]}
- Instructions spécifiques pour « if ($U_t = 4$) » :
  if (FSL_t $\{>;\geq\}$ 7) $d_t = \{[-2\,;\,0]\}$
  if (FSL_t $\{<;\leq\}$ 1) $d_t = \{[0\,;\,\mathbf{1}]\}$
  if (FSL_t $\{>;\geq;<;\leq\}$ $\{[2\,;\,6]\}$) $d_t = \{[-2\,;\,\mathbf{1}]\}$
- Instructions spécifiques pour « if ($U_t = 5$) » :
  if (FSL_t $\{>;\geq\}$ 7) $d_t = \{[-2\,;\,0]\}$
  if (FSL_t $\{<;\leq\}$ 1) $d_t = \mathbf{0}$
  if (FSL_t $\{>;\geq;<;\leq\}$ $\{[2\,;\,6]\}$) $d_t = \{[-2\,;\,\mathbf{0}]\}$

où nous soulignons (en gras) ce qui change par rapport à l'instruction originelle associée.

À noter que la taille d'une solution n'est alors plus définie de façon globale mais individuellement pour chaque sous-partie à développer. Nous les avons toutes définies de la même façon, à savoir selon le Tableau 5.17. Puisque nous avons cinq sous-parties, leur somme respecte les tailles minimale et maximale définies auparavant pour une solution « complète » (Tableau 5.3).

Tailles de sous-parties	Valeur
Minimale	1
Maximale	10
Moyenne	3
Écart-type	5

Tableau 5.17 – Paramètres concernant chaque sous-partie d'un individu

Les croisements sont également redéfinis de sorte qu'ils concernent deux sous-parties équivalentes. De même, les mutations devront prendre en compte la sous-partie devant être génétiquement modifiée. Ceci est déjà intégré dans le logiciel μGP.

Les configurations utilisées pour la PGL afin de tester l'introduction de connaissances expertes à ce problème du ConWIP sont : $Config^{-+}$, $Config^{++}$ et $Config^{*+}$ (récapitulées lors du dernier groupe de test, Section 5.2.4.4.2).

5.2.4.5.3 Analyse des résultats

Le Tableau 5.18 indique les résultats obtenus par notre approche. Il fournit également un récapitulatif des résultats pour le ConWIP traditionnel (ligne CT) et pour l'approche de [Tardif et Maaseidvaag 2001] (ligne TM).

Configurations	Stable	Cyclique	Triangulaire
CT	6,391456	7,354815	7,183995
TM	5,658755	6,260968	5,998775
$Config^{-+}$	5,845754	6,830068	7,152117
$Config^{++}$	6,087438	6,187894	5,910196
$Config^{*+}$	5,553217	6,139487	5,877169

Tableau 5.18 – Résultats pour le ConWIP avec de la connaissance experte ajoutée

On aurait tendance à penser que notre niveau d'expertise, bien que faible, devrait conduire à une meilleure convergence mais les observations montrent plutôt le contraire. Les résultats restent significativement meilleurs que le ConWIP traditionnel mais 4 sur 9 sont moins bons que ceux de [Tardif et Maaseidvaag 2001]. Les résultats souvent moins bons que leur équivalent sans connaissance supplémentaire peuvent cependant être expliqués. La redéfinition des opérateurs rend la convergence plus lente : nous ne changeons pas globalement la logique décisionnelle mais seulement une de ses sous-parties à la fois, ce qui restreint fortement le niveau de diversité des enfants obtenus.

De plus, l'expertise ajoutée assure seulement qu'aucune correction n'est nécessaire avant l'utilisation de la logique décisionnelle obtenue. Pourtant, lorsque la connaissance n'est pas prise en compte, ces corrections sont faites après coup et les résultats « génériques » obtenus dans ce cas convergent plus rapidement vers l'objectif.

En somme, l'ajout d'expertise devrait aider l'apprentissage mais il demande plus de précision sur chaque bout d'arbre à développer. Cela rend la tâche plus difficile mais justifie en partie les résultats parfois meilleurs, parfois moins bons.

Le résultat n'est donc pas concluant mais ouvre des voies de recherche. Une analyse plus adéquate impliquerait l'ajout d'une expertise « effective » et non pas d'une expertise restreinte aux valeurs que peut prendre une variable selon son état courant. En effet, celle que nous avons ajoutée correspond à la deuxième solution envisagée pour prendre

en compte les contraintes : il s'agit de la définition de sous-ensembles $E_{j,k,t}$ et $D_{n,t}$ contenant seulement les éléments valides à un instant t (Section 5.2.3.4, Annexe E). Notons par ailleurs que le nombre de sous-parties risque d'augmenter exponentiellement avec le nombre de variables nécessitant un tel traitement. Les corrections à posteriori restent donc plus raisonnables et constituent la solution à retenir lorsqu'elle est possible.

5.3 Adaptation de la répartition des machines dans un réseau intra-entreprise

5.3.1 Contexte

Pour notre deuxième cas d'étude, nous nous sommes intéressés à une situation de plus en plus récurrente de nos jours : le partage de ressources en entreprise, que ce soit au sein du même ou entre plusieurs sites (Figure 5.12). Nous avons ici ce que l'on appelle un réseau logistique intra-entreprise (*intrafirm network*). Pour plus de détails, se référer par exemple à [Tsai et Ghoshal 1998, Colombo *et al.* 2011, Ren *et al.* 2013].

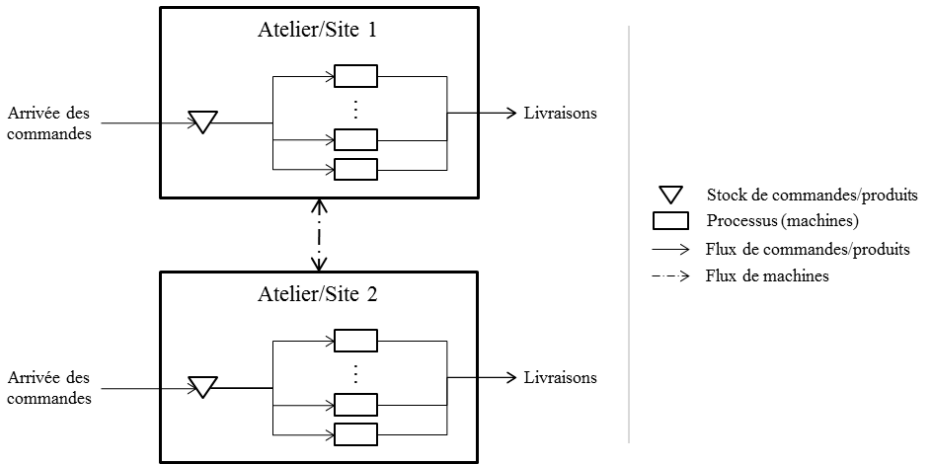

FIGURE 5.12 – Schéma simplifié d'un système à ressources partagées

À notre connaissance, il n'y a pas de travaux dans la littérature traitant du problème d'adaptation de la répartition des machines dans un système multi-ateliers, ce qui le rend d'autant plus intéressant. Bien que nous nous concentrions sur ce cas précis, les considérations et conclusions pourront être extrapolées au cas multi-site. Pour ce faire, certaines précautions seront nécessaires comme le changement dans l'échelle de temps et la prise en compte d'éventuelles contraintes supplémentaires.

Comme pour l'exemple précédent, nous nous servirons du cadre conceptuel du Chapitre 2 et de la formalisation du Chapitre 4 dans la description qui suit.

5.3.2 Description du système

5.3.2.1 Éléments constituant le système

Le système de production en question possède deux gammes de produits que nous noterons $Produit_A$ et $Produit_B$. Il est organisé en deux ateliers, chacun dédié à un type de produit exclusivement : $Atelier_A$ pour le $Produit_A$ et $Atelier_B$ pour le $Produit_B$. Dans ce qui suit, nous utiliserons l'indice $w \in W$ pour faire référence au type de produit, le même indice étant aussi utilisé pour désigner son atelier respectif. Ici, $W = \{A; B\}$.

Le processus de fabrication d'un produit est composé de plusieurs opérations, toutes réalisées par une même machine (sans interruption). Les temps de traitement sont donnés en minutes et ne dépendent pas de la machine utilisée mais du produit : ils suivent une distribution exponentielle dont la moyenne est 15 pour un $Produit_A$ et 20 pour un $Produit_B$ (taux de service $\mu_A = \frac{1}{15}$ et $\mu_B = \frac{1}{20}$). Le système compte un total de $Cap = 7$ machines polyvalentes. Elles peuvent donc être utilisées pour la production du $Produit_A$ et du $Produit_B$ à condition qu'elles se trouvent dans l'atelier approprié. Ceci confère au système son caractère adaptable tout en sachant que sa capacité totale Cap est figée.

Le but est ici d'intervenir sur la position des machines sans pour autant les définir individuellement, c'est-à-dire, sans nous occuper de leur emplacement précis au sein d'un atelier. Nous ne sommes donc pas au même niveau de détail que celui de l'horizon décisionnel d'ordre opérationnel. Ceci fait également que nous avons un seul objet modifiable ($J = 1$). En effet, puisqu'il n'y a pas besoin que chaque atelier soit vu comme un objet à part entière, nous avons $O = \{o_1\}$ où o_1 représente les différents ateliers réunis, soit le système multi-ateliers vu comme un tout.

Nous avons également un seul attribut variable ($K_1 = 1$). Le vecteur associé est $A_1 = \langle a_{1,1} \rangle$, où $a_{1,1}$ représente la répartition des Cap machines. Instanciée dans le temps ($a_{1,1,t}$), cette dernière prend la forme d'un vecteur de dimension $|W|$ où chaque composante notée $NbM_{w,t}$ correspond au nombre de machines à l'instant t sur un $Atelier_w$ donné. Ici, $|W| = 2$ et $a_{1,1,t} = \langle NbM_{A,t}; NbM_{B,t} \rangle$.

Lorsque le partage de ressources est choisi comme stratégie, l'hypothèse suivante peut généralement être énoncée : le système ne contient pas de « redondance », c'est-à-dire des machines de rechange qui restent stockées pour en remplacer d'autres en cas de pannes ou lorsque ces dernières ne sont plus performantes. S'il y avait de redondance, ces machines seraient plutôt mises en activité pour réduire voire éliminer le besoin de partage de ressources. Nous en déduisons donc une contrainte assez logique : $\sum_w NbM_{w,t} = Cap, \forall t$. Autrement dit, toute machine doit être attribuée à l'un des ateliers. Pour notre système en particulier, $NbM_{A,t} + NbM_{B,t} = 7, \forall t$.

Dans ce type de système, il doit y avoir à tout instant un nombre minimum $MMin_w$ de machines sur chaque $Atelier_w$ pour qu'il puisse fonctionner correctement. Ce nombre

n'est pas forcément le même pour tous les ateliers et dans le cas extrême, il vaut 1. Notons que le nombre total de machines doit non seulement être suffisant pour satisfaire le minimum requis dans chaque atelier mais il doit aussi permettre le partage de ressources : $Cap > \sum_w MMin_w$. Ici nous considérerons $MMin_w = 1, \forall w$.

D'autre part, il n'est pas pertinent d'avoir plus de machines par $Atelier_w$ qu'un nombre maximum donné $MMax_w$ car ceci impliquerait un taux d'utilisation trop faible des machines. Pour que le partage de ressource soit envisagé, le nombre total de machines est supposé inférieur à la somme du maximum admis pour chaque atelier : $Cap < \sum_w MMax_w$. Ici, $MMax_w$ est directement calculée pour chaque $Atelier_w$ à partir du nombre minimum défini pour les autres ateliers : $MMax_w = Cap - \sum_{w'} MMin_{w'}$, où $w' \in W \mid w' \neq w$. Ainsi, nous avons $MMax_w = 6, \forall w$.

Ces différentes considérations nous imposent une contrainte liée au bon fonctionnement du système :

$$MMin_w \leq NbM_{w,t} \leq MMax_w, \forall w, t \tag{5.4}$$

Par celle-ci, nous obtenons toutes les répartitions possibles des machines dans le système. Pour notre exemple, nous avons précisément $E_{1,1} = \{\langle 1;6\rangle; \langle 2;5\rangle; \langle 3;4\rangle; \langle 4;3\rangle; \langle 5;2\rangle; \langle 6;1\rangle\}$.

Notons que, puisque les Cap machines doivent toutes être réparties dans les ateliers du système, le nombre de machines dans un atelier est lié au nombre de machines dans les autres. Ce lien est encore plus fort dans le cas traité où nous avons seulement deux ateliers : quand la Contrainte 5.4 est satisfaite pour un atelier, elle l'est également pour l'autre. Nous pouvons donc nous limiter à l'analyser pour un seul des deux ateliers. Ceci n'est cependant pas valable si le système contient trois ateliers ou plus.

Pour pouvoir tirer profit de la flexibilité de notre système, l'ensemble C compte un type de changement ($L = 1$) : $C = \{c_1\}$ où c_1 représente le déplacement de machines. Le temps de mise en place d'un déplacement n'est pas négligeable et pour ce système nous avons $H_{durée,1} = 1$ heure. Bien que le temps de validité d'un changement soit normalement supérieur à celui de sa mise en place, rappelons que nous regardons la répartition de la capacité dans un système multi-ateliers d'un point de vue général. Ainsi, $H_{min,1}$ ne concerne pas le déplacement d'une machine en particulier mais un déplacement quelconque et aucune hypothèse n'est faite sur sa valeur, tout du moins pour l'instant. Dans tous les cas, ces décisions de déplacement sont ici associées à l'horizon décisionnel $H_{D,1}$ de type tactique. Notons cependant que lorsque $H_{durée,1}$ est d'ordre plus élevé, $H_{D,1}$ est de type stratégique.

Du point de vue de notre problème d'adaptation, nous avons une seule variable de décision ($N = 1$) qui doit préciser à la fois le nombre de machines à déplacer mais aussi depuis où et vers où le faire. Une décision $d_t = \langle d_{1,t}\rangle$ sera ici représentée par un entier dont la valeur absolue ($|d_t|$) et le signe indiquent respectivement le nombre de machines concernées et la direction du déplacement. Les actions positives représentent des déplacements de l'$Atelier_B$ vers l'$Atelier_A$ et inversement pour les négatives. L'action par défaut est $d_t = 0$ où aucun déplacement n'est fait. La nouvelle répartition est ainsi

définie comme suit : $\langle NbM_{A,t}; NbM_{B,t}\rangle := \langle NbM_{A,t} + d_t; NbM_{B,t} - d_t\rangle$. Bien évidemment, les nombres de machines $NbM_{A,t}$ et $NbM_{B,t}$ préconisés par l'action d_t doivent satisfaire la Contrainte 5.4. Notons également qu'une autre formulation est nécessaire pour les systèmes comptant plus d'ateliers.

Il est connu que la flexibilité de ce type de système est parfois limitée. Ceci se traduit par un nombre maximum d_{max} de machines qu'il est possible de déplacer en même temps dans le système. C'est par exemple le cas lorsqu'un chariot de transport est nécessaire mais qu'il ne peut prendre en charge qu'un volume limité de machines. L'ensemble d'actions est donné par $\mathcal{D} = D_1 = [-d_{max}; d_{max}]$. Dans le cas le plus restreint, $d_{max} = 1$ ($\mathcal{D} = [-1; 1]$). À l'autre extrême se trouve le cas le plus flexible. Ici, ce serait $d_{max} = 5$ ($\mathcal{D} = [-5; 5]$). Toutefois et spécifiquement pour notre système, $d_{max} = 1$ car une seule machine peut être déplacée à la fois.

Une variable auxiliaire $NbMEnDep_t$ indique le nombre de machines en déplacement à l'instant t. Par définition, $NbMEnDep_t \leq d_{max}, \forall t$, ce qui assure ici le maximum d'un seul déplacement à la fois. Cette variable est nécessaire puisque $H_{durée,1} > 0$ permet qu'une nouvelle décision soit prise avant la fin d'un autre déplacement. Ainsi, bien que $d_{max} = 1$, nous pouvons nous trouver face à des déplacements qui n'ont pas démarré au même instant mais qui se chevauchent.

Revenons maintenant à $H_{min,1}$. Désigner $H_{min,1} \geq H_{durée,1}$ interdit toute possibilité de chevauchement de déplacements ne démarrant pas ensemble. Si une telle interdiction est indésirable (ce qui suppose $d_{max} > 1$), par exemple lorsque plusieurs opérateurs/chariots peuvent se charger de déplacements en parallèle, il faut établir $H_{min,1} = 0$. Cependant, $H_{min,1} \geq H_{durée,1}$ vise une plus grande stabilité du système, ce qui est lié à la notion de nervosité discutée lors du Chapitre 2 (Sections 2.3.1.2 et 2.3.2.1). Ici, nous en analyserons deux cas : $H_{min,1} = H_{durée,1}$ et $H_{min,1} = 1$ jour. Ce dernier interdit deux décisions de déplacement à moins de 24 heures d'intervalle. La valeur de $H_{min,1}$ utilisée sera précisée pour chaque expérimentation dans la Section 5.3.4 dédiée aux résultats.

Dans tous les cas, le but est de garder le système le plus rentable possible, ce qui se traduit ici par la minimisation des encours moyens sans qu'il n'y ait de différentiation suivant le type de produit. La maximisation de la production totale (les deux types de produit confondus) sera également analysée afin de vérifier lequel semble le plus adéquat parmi ces deux objectifs dans G étudiés individuellement.

5.3.2.2 Fonctionnement du système

Dans ce système, les produits sont réalisés sur commande de sorte que le processus de production démarre une fois deux conditions remplies : l'arrivée d'une commande et la disponibilité d'une machine pour la traiter. Nous démarrons le système avec son processus de fabrication vide et toutes ses machines prêtes, donc en attente de l'arrivée de commandes. La répartition initiale est celle qui favorise l'équilibre de la production, c'est-

à-dire 3 machines dans l'$Atelier_A$ et 4 dans l'$Atelier_B$, ce qui correspond à une capacité moyenne de 12 produits par heure pour chacun des ateliers.

Le fonctionnement du système peut être décrit comme suit. Les commandes arrivent selon un processus de Poisson sachant qu'aucune information n'est disponible sur la demande future. Lorsqu'une commande arrive, elle est directement acheminée vers l'atelier qui la concerne. Le temps d'acheminement est considéré comme étant négligeable. Elle est alors traitée indifféremment par l'une des machines disponibles. Si toutes les machines sont déjà occupées, elle est placée dans une file d'attente commune à toutes les machines de l'atelier. Les commandes sont traitées par ordre d'arrivée : il s'agit de la règle de priorité FIFO (pour *first in first out*).

À la fin d'un traitement, le produit obtenu est directement livré au client sans aucun temps supplémentaire à comptabiliser. La machine est alors libérée et, s'il y a une commande en attente dans son atelier, elle la prendra en charge immédiatement. Cependant, si la machine doit être déplacée dans le cas d'une demande d'adaptation, ce déplacement est prioritaire sur la production et la machine ne prendra pas en charge la commande en attente. Elle sera déplacée et restera indisponible jusqu'au terme de son déplacement.

C'est lors de l'arrivée de nouvelles commandes qu'une décision peut être prise quant à l'adaptation ou non du système à cet instant. En effet, l'arrivée d'une demande ne déclenche pas toujours de prise de décision puisqu'aucun déplacement ne peut être envisagé si $NbMEnDep_t = d_{max}$. Mais lorsqu'une décision a lieu et si une adaptation est décidée, elle est toujours appliquée avant l'affectation des commandes aux machines. Nous avons ici la définition de notre stratégie de déclenchement T : l'arrivée d'une commande sujette à la condition « aucun déplacement actuellement en cours » constitue notre événement déclencheur suite auquel nous décidons du déplacement ou non d'une machine et, le cas échéant, de la direction associée.

Ces décisions cherchent soit à minimiser le coût total qui représente les encours soit à maximiser le gain total qui représente les produits livrés. Autrement dit, les objectifs dans G sont utilisés individuellement, le premier étant évalué par sa valeur moyenne sur une période d'étude et le deuxième par sa valeur cumulée sur toute la période d'étude. Quel que soit l'objectif, c'est maintenant qu'interviendra notre approche.

Les décisions seront donc prises de façon centralisée (structure S) en utilisant une logique décisionnelle trouvée par l'application de la programmation génétique linéaire basée sur la simulation (mécanisme de changement M). Pour ce faire, nous surveillons $nq_{w,t}$ qui désigne le nombre de produits à l'instant t dans la file d'attente de l'$Atelier_w$, et ce pour tout w. De plus, le nombre $NbM_{A,t}$ de machines dans l'$Atelier_A$ peut également être pris en compte. Il n'en est pas de même pour $NbM_{B,t}$ car nous le déduisons à partir de $NbM_{A,t}$, ce qui simplifie l'espace de recherche : un seul de ces nombres suffit pour connaître la répartition courante des machines. Ainsi, $nq_{A,t}$, $nq_{B,t}$ et $NbM_{A,t}$ constituent notre vecteur de variables d'état S_t, partie dynamique de notre ensemble d'informations I.

5.3.2.3 Variation de la demande

Nous nous intéressons à l'adaptation de la répartition des machines dans ce réseau intra-entreprise que nous venons de décrire. Une adaptation suppose des changements, ce qui se traduit ici par des variations au niveau de l'arrivée des demandes au cours du temps. Ces demandes sont caractérisées par un processus de Poisson dont la moyenne des durées inter-arrivées est notée $dia_w = \frac{1}{\lambda_w}$ pour chaque type w de produit.

Pour illustrer notre approche, nous avons donc introduit dans notre modèle de simulation des variations dans $dia_w, \forall w$ qui suivent une *pattern* dite saisonnière (puisqu'elle change tous les 3 mois) auxquelles le système devrait s'adapter. Pour le $Produit_A$, dia_A passe de 5,95 à 4,3 puis à 3,35 et enfin à 4,3, d'où elle recommence à 5,95. Pour le $Produit_B$, dia_B démarre à 5,75, pour passer à 7,9, puis à 12,9 et enfin à 7,9, d'où elle recommence elle aussi à sa valeur initiale de 5,75. Ces *patterns* (une pour chaque produit) se répètent tout au long de la période d'étude. Elles sont regroupées dans la Figure 5.13. Les valeurs sont exprimées en minutes.

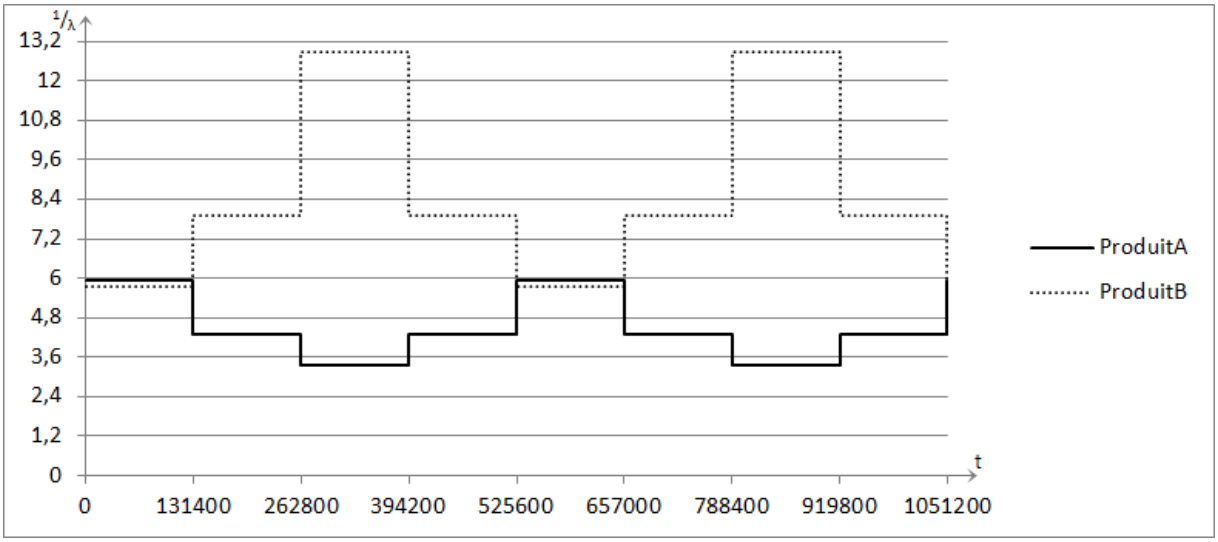

FIGURE 5.13 – *Pattern* utilisée pour la variation des demandes dans la simulation du système multi-ateliers

5.3.3 Conditions d'expérimentations

5.3.3.1 Généralités

Les conditions d'expérimentation sont les mêmes que celles énoncées dans la Section 5.2.3.1. Ainsi, nous utilisons les logiciels Arena et μGP, les expérimentations étant réalisées sur le même PC et toujours en deux étapes : 5 réplications de la simulation lors de l'apprentissage puis 30 de la meilleure solution.

Pour étudier l'adaptation de la répartition des machines, nous simulons plus précisément 2 ans du fonctionnement du système. La durée « fictive » de chaque réplication de la simulation est ainsi de 1051200 minutes, valeur qui reste identique pour les deux étapes d'expérimentation.

5.3.3.2 Objectif de l'étude

D'après la description du système, la fonction objectif de base pour notre problème peut être énoncée comme suit :

$$Minimiser\ z = \mathbb{E}[\overline{wip_A} + \overline{wip_B}] \tag{5.5}$$

où $\overline{wip_w}$ correspond à la valeur moyenne des encours de type w. Les coûts unitaires (par unité de produit et de temps) sont de 1 aussi bien pour le $Produit_A$ que pour le $Produit_B$: malgré les temps de traitement différents, aucune différentiation n'est faite quant au capital immobilisé par leur détention (en stock ou production).

Pour analyser la pertinence de ce critère de performance, nous ferons également d'autres tests avec une fonction objectif « alternative » :

$$Maximiser\ z' = \mathbb{E}[NbLivraisons_A + NbLivraisons_B] \tag{5.6}$$

où $NbLivraisons_w$ correspond au nombre total de produits livrés de type w : malgré les temps de traitement différents, aucune différentiation n'est faite non plus quant à leur valeur ajoutée.

Cette démarche nous permettra également d'analyser l'impact du critère de performance sur la solution obtenue. Pour cela, rappelons que ces deux fonctions objectif ne sont pas cumulées mais toujours étudiées indépendamment l'une de l'autre.

De plus, bien que le temps moyen de passage des produits dans l'atelier (*flow time*) ne soit pas évoqué comme un critère de performance (à minimiser) pour ce système, il sera fourni à titre indicatif dans tous les tableaux de performance qui suivront. Puisqu'aucune différenciation n'est faite par type de produit pour le calcul de z et z', cette même logique sera conservée ici. Notons cependant que si l'on indique la valeur moyenne du temps de passage de tous les produits sans distinction, celle-ci sera sensible à la proportion de chaque type de produit. Pour rester neutre, une première moyenne $\overline{FTM_w}$ doit d'abord être calculée pour chaque type w de produit. Il s'agit du temps moyen de passage pour les produits d'un type w donné :

$$\overline{FTM_w} = \frac{1}{NbLivraisons_w} \cdot \sum_{np} FT_{w,np}, \forall w \tag{5.7}$$

où $FT_{w,np}$ et $NbLivraisons_w$ désignent respectivement le temps de passage du $np^{ième}$ produit de type w dans l'atelier et le nombre de produits livrés de ce même type.

Par la suite, les deux moyennes obtenues sont utilisées pour obtenir une moyenne finale notée $\overline{FTM}$:

$$\overline{FTM} = \frac{\overline{FTM_A} + \overline{FTM_B}}{2} \tag{5.8}$$

Remarquons que ces équations pour le calcul de $\overline{FTM}$ prennent en compte une seule réplication de la simulation. Or, pour tous les tableaux qui suivront tout comme z et z', les valeurs indiquées correspondent déjà aux estimations après 30 réplications. Nous les noterons $FTM = \mathbb{E}[\overline{FTM}]$.

5.3.3.3 Valeurs de référence pour la comparaison de nos résultats

Contrairement au premier cas d'étude, nous n'avons pas cette fois-ci et à notre connaissance de travaux dans la littérature auxquels comparer (de façon quantitative ou qualitative) les résultats que nous pourrons obtenir. Ainsi, pour nous permettre d'évaluer la pertinence des performances obtenues par notre approche, nos comparaisons seront faites en fonction de cas créés artificiellement. Ils cherchent à caractériser une « mauvaise » et une « bonne » logiques décisionnelles.

Le premier cas correspond à une répartition fixe des machines où aucun changement n'est fait tout au long de la période d'étude. Il est par ailleurs nommé « statique ». Dans ce scénario représentatif d'une « mauvaise » logique décisionnelle, nous considérons malgré tout qu'une optimisation est faite pour définir la répartition de machines à retenir parmi les 6 possibilités : $\langle 1;6\rangle, \langle 2;5\rangle, \langle 3;4\rangle, \langle 4;3\rangle, \langle 5;2\rangle, \langle 6;1\rangle$. Nous avons simulé chacune d'entre elles pour en obtenir les performances (Tableau 5.19). Ceci a été fait en utilisant 30 réplications d'une simulation de 2 années de production, c'est-à-dire dans les mêmes conditions d'évaluation que celles d'une logique décisionnelle obtenue par la PGL.

Répartitions	z	z'	FTM
$\langle 1;6\rangle$	83116,3805	202706,63	173071,75
$\langle 2;5\rangle$	48042,0798	272808,90	103360,11
$\langle 3;4\rangle$	16911,0635	338752,27	35400,66
$\langle 4;3\rangle$	1815,7906	377387,13	5344,52
$\langle 5;2\rangle$	16916,4247	349995,17	68866,92
$\langle 6;1\rangle$	43238,4006	297370,37	174613,71

Tableau 5.19 – Performances pour les cas statiques de répartition de machines

Notons que le classement des solutions $\langle 3;4\rangle$ et $\langle 5;2\rangle$ n'est pas le même suivant le critère de performance pris en compte : $\langle 3;4\rangle$ est meilleure pour z et FTM tandis que $\langle 5;2\rangle$ l'est pour z'. De même, $\langle 6;1\rangle$ est quatrième sur six pour z et z' mais dernière pour FTM. Dans tous les cas, la répartition $\langle 4;3\rangle$ est retenue pour représenter le cas statique d'après ses meilleures performances estimées.

Passons au deuxième cas ici nommé le cas « automatique ». Il correspond quant à lui à la détection sans délai du changement de la moyenne des durées inter-arrivées des demandes. Ceci est possible car, bien que l'évolution de la demande soit considérée comme étant inconnue pour la résolution du problème, nous connaissons la *pattern* introduite dans le modèle de simulation (Figure 5.13). Suite à cette détection, l'adaptation « la plus adéquate » est immédiatement mise en œuvre. Encore une fois, ceci est rendu possible en connaissance de la *pattern* utilisée : chaque adaptation a donc pu être déterminée d'après la théorie des files d'attentes puis vérifiée par simulation. Ainsi et conformément à la *pattern* de la Figure 5.13, la répartition des machines change tous les 3 mois. Elle commence à $\langle 3; 4\rangle$, passant à $\langle 4; 3\rangle$, $\langle 5; 2\rangle$, $\langle 4; 3\rangle$ pour enfin revenir à celle initialement définie ($\langle 3; 4\rangle$).

Notons par ailleurs que parmi les répartitions fixes, les trois les plus pertinentes sont celles qui correspondent à une période de la *pattern* utilisée : $\langle 3; 4\rangle$, $\langle 4; 3\rangle$ et $\langle 5; 2\rangle$. Puisque nous passons 50% du temps sur des taux moyens de demandes qui requièrent la répartition $\langle 4; 3\rangle$, il était attendu que celle-ci procure les meilleures performances et soit donc retenue pour représenter le cas statique.

Le Tableau 5.20 nous donne les performances des deux cas qui seront utilisés par la suite pour nos comparaisons.

Répartitions	z	z'	FTM
Automatique	14,8418	377504,33	43,28
Statique	1815,7906	377387,13	5344,52

Tableau 5.20 – Performances pour les cas de référence retenus pour la répartition de machines

Ces résultats nous donnent déjà un aperçu de l'importance du critère de performance sur la solution obtenue. Tout d'abord et d'après les remarques que nous venons de faire, nous pouvons constater que le choix d'une solution peut changer selon le critère retenu. Notons qu'ici toutes les fonctions objectif ont un meilleur résultat avec la même solution mais il est possible que ce ne soit pas le cas. D'autre part, nous pouvons remarquer que les différences de performance sont beaucoup plus fortes pour z (tout comme pour FTM) que pour z'. Ainsi, selon l'objectif visé, le choix de la solution peut s'avérer peu ou très important.

Maintenant que nous avons un socle de valeurs de référence, passons à notre résolution par programmation génétique linéaire.

5.3.3.4 Formulation du problème dans le cadre de la programmation génétique linéaire

Afin de déterminer comment adapter la répartition des machines dans le système multi-ateliers traité ici, nous avons un registre de sortie unique : $T_A = \{d_t\}$. Puisqu'une seule

machine peut être déplacée à la fois, cette variable de décision d_t peut prendre une valeur de l'ensemble $T_{CA} = \mathcal{D} = [-1; 1]$ mais lorsque $NbMEnDep_t = 1$, la prise de décision n'a pas lieu.

Pour satisfaire la Contrainte 5.4 sur le nombre minimum (et par conséquent maximum) de machines dans chaque atelier, quelques précautions sont nécessaires à propos de T_{CA} dont les actions valides évoluent dans le temps. Comme dans l'exemple du ConWIP, nous en tenons compte en corrigeant au besoin l'action proposée par la logique décisionnelle, et ce directement dans notre modèle de simulation. Ceci est fait par un changement dans l'interprétation de ces actions :

$$d_t := \begin{cases} min\{d_t; MMax_A - NbM_{A,t}\} & \text{si } d_t > 0 \text{ (ici, si } d_t = 1) \\ max\{d_t; MMin_A - NbM_{A,t}\} & \text{si } d_t < 0 \text{ (ici, si } d_t = -1) \\ d_t & \text{si } d_t = 0 \end{cases} \tag{5.9}$$

La valeur proposée est ainsi convertie en une valeur effective donc autorisée à cet instant. Comme expliqué auparavant, puisque nous avons uniquement deux ateliers, nous pouvons utiliser la Contrainte 5.4 basée sur un seul atelier pour faire les corrections. Chaque action d_t violant cette contrainte ou susceptible de le faire est remplacée par l'action valide la plus proche. Dans notre cas précis, une violation entraînera toujours l'annulation de la décision. Par exemple, si à un moment t donné l'action proposée est de déplacer une machine de l'$Atelier_B$ vers l'$Atelier_A$ alors que ce dernier compte déjà six machines, aucune machine ne sera déplacée : $d_t := min\{1; 6 - 6\} = 0$. Si ce dernier compte cependant de 1 à 5 machines, la correction aura pour résultat la décision initiale puisque l'action proposée est déjà valide et donc effective.

Il ne nous est ici pas possible de nous servir de prévisions pour la prise de décision. Les registres mémoire d'entrée sont composés des variables d'état explicitées dans la section précédente : $T_V = \{nq_{A,t}; nq_{B,t}; NbM_{A,t}\}$. Ces variables seront comparées à des seuils définis par T_{CV}. Pour $NbM_{A,t}$, c'est assez intuitif : il varie de $MMin_A = 1$ à $MMax_A = 6$. Pour réduire partiellement la redondance et donc l'espace de recherche, l'intervalle retenu exclut les deux bornes. Cela est rendu possible puisque nous utilisons des comparateurs d'égalité, de supériorité et d'infériorité. Les deux premières variables sont quant à elles empiriquement associées à des seuils appartenant à l'intervalle [2 ; 60]. Puisque ce n'est pas leur valeur précise qui nous intéresse, le comparateur d'égalité n'est pas retenu pour ces deux variables. Ainsi, $T_{CV} = \{[2; 60]; [2; 60]; [2; 5]\}$ et $T_C = T_{CV} \cup T_{CA} = \{[2; 60]; [2; 60]; [2; 5]; [-1; 1]\}$.

Quelques tests seront également faits où seuls $nq_{A,t}$ et $nq_{B,t}$ sont mis à disposition. Nous évaluerons ainsi la pertinence d'avoir ou non dans le vecteur S_t la variable d'état $NbM_{A,t}$ associée à celle de décision.

En définissant « ne rien faire » comme action par défaut, ceci nous donne l'ensemble suivant d'instructions :

- $d_t = 0$

- if ($\{nq_{A,t}; nq_{B,t}\}$ $\{>; <\}$ $\{[2\,;\,60]\}$) jumpTo *label*
- if ($\{nq_{A,t}; nq_{B,t}\}$ $\{>; <\}$ $\{[2\,;\,60]\}$) $d_t = \{[\text{-}1\,;\,1]\}$
- if ($NbM_{A,t}$ $\{=; >; <\}$ $\{[2\,;\,5]\}$) jumpTo *label*
- if ($NbM_{A,t}$ $\{=; >; <\}$ $\{[2\,;\,5]\}$) $d_t = \{[\text{-}1\,;\,1]\}$

Toutefois, les deux dernières instructions ne seront pas utilisées pour les expérimentations excluant $NbM_{A,t}$. Nous avons donc deux grands groupes d'expérimentation : $Groupe_{QM}$ si ces instructions sont prises en compte et $Groupe_Q$ sinon.

Les logiques décisionnelles sont définies de sorte à avoir entre 5 et 25 instructions tout au long de l'évolution. L'initialisation de la population faite aléatoirement utilise un écart-type de 5 et diffère quant à la moyenne utilisée pour la distribution normale : celle-ci vaut 7 pour le $Groupe_Q$ et 14 pour le $Groupe_{QM}$. Le Tableau 5.21 récapitule ces valeurs empiriquement déterminées.

Tailles d'individus	**Valeur**
Minimale	5
Maximale	25
Moyenne	$\{7\,;\,14\}$
Écart-type	5

Tableau 5.21 – Paramètres concernant les individus pour le problème de répartition de machines

Les autres paramètres nécessaires au fonctionnement d'un algorithme de PGL sont présentés dans les Tableaux 5.22 et 5.23 qui suivent. Le premier contient des paramètres définis empiriquement et communs à toutes les expérimentations tandis que le deuxième spécifie les différences entre les trois configurations testées.

Paramètres	**Valeurs**
Immortalité	Non
V_{elite}	$\lceil 0,15 \cdot V_{max} \rceil$
$NbOp$	V_{max}
P_{max}	100
P_{stagne}	$0,5 \cdot P_{max} = 50$
$V_{tournoi}$	2

Tableau 5.22 – Paramètres de la PGL pour le problème de répartition de machines

Paramètres	$Config^P$	$Config^M$	$Config^G$
V_{max}	25	50	100
V_{elite}	4	8	15
$NbOp$	25	50	100

Tableau 5.23 – Configurations utilisées pour le problème de répartition de machines

Dans la section qui suit, nous ferons référence à ces configurations au fur et à mesure de l'arrivée des résultats. Nous utiliserons le plus souvent la paire <configuration - groupe> pour désigner une expérimentation précise, c'est-à-dire une configuration donnée appliquée à un groupe donné.

5.3.4 Résultats

5.3.4.1 Influence du critère de performance

Nous allons tout d'abord donner suite à l'analyse démarrée dans la Section 5.3.3.3 sur l'impact du choix du critère de performance dans la solution. Ceci sera fait pour $H_{min,1} = H_{durée,1} = 1$ heure. Nous avons un total de 12 expérimentations : 2 configurations pour un individu ($Groupe_Q$ et $Groupe_{QM}$) $\times$ 3 configurations pour la PGL ($Config^P$, $Config^M$ et $Config^G$) $\times$ 2 fonctions objectif (z et z').

5.3.4.1.1 Convergence et temps de calcul

Selon les différentes expérimentations, l'évolution de la meilleure solution de la population est présentée dans les Figures 5.14 et 5.15. La première représente la minimisation des encours moyens (z, Équation 5.5) alors que la deuxième représente la maximisation de la production totale (z', Équation 5.6).

Notons que ces figures ont été centrées sur une « zone d'intérêt » afin de mieux visualiser la progression des performances. Cet ajustement a pour effet d'éliminer certaines valeurs (très mauvaises). Toutefois, le Tableau 5.24 récapitule les performances initiales pour nous permettre leur analyse. Provenant directement du processus d'apprentissage, c'est le seul tableau où les performances qui y figurent correspondent à celles obtenues après 5 réplications.

Expérimentations	z	z'
$<Config^P$ - $Groupe_Q>$	16949,1525	338702
$<Config^M$ - $Groupe_Q>$	19,4024	377816
$<Config^G$ - $Groupe_Q>$	19,4024	377816
$<Config^P$ - $Groupe_{QM}>$	2506,2657	377891
$<Config^M$ - $Groupe_{QM}>$	752,4454	377891
$<Config^G$ - $Groupe_{QM}>$	752,4454	377891

Tableau 5.24 – Performances des meilleures solutions pour les populations initiales aléatoires

Nous pouvons constater que les populations initiales construites aléatoirement peuvent présenter des performances très divergentes. En effet, l'écart initial de la performance

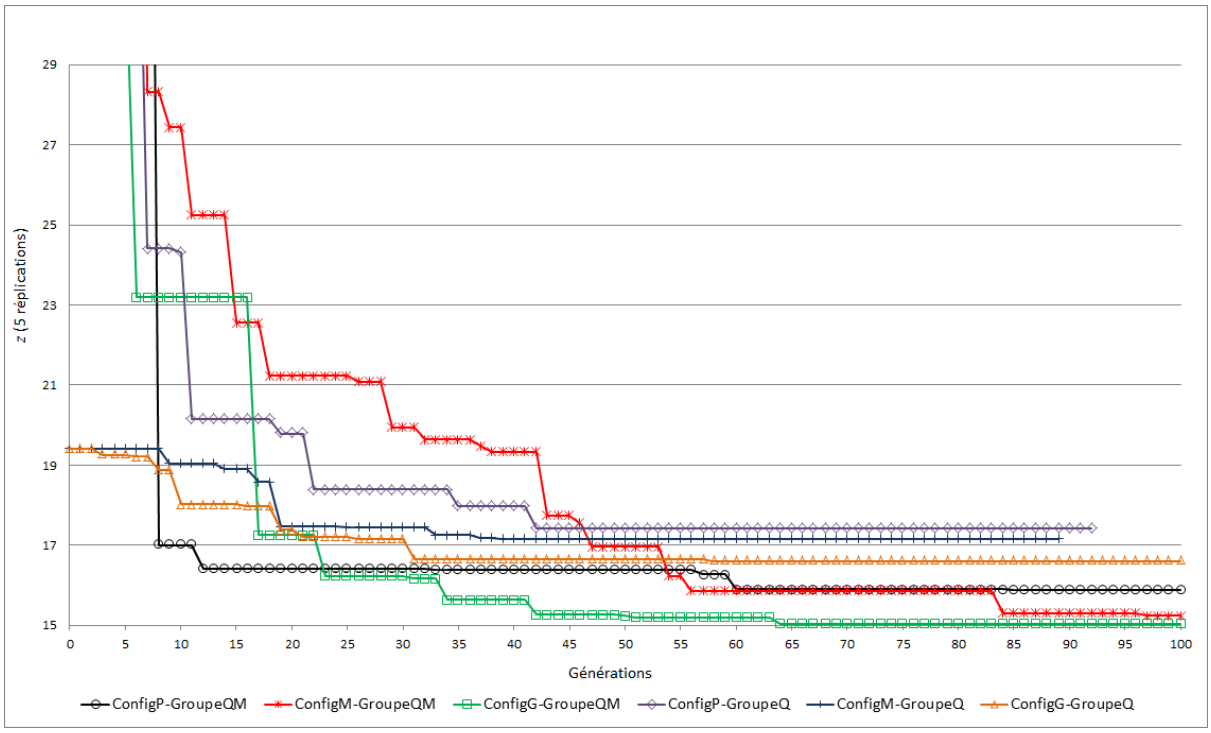

FIGURE 5.14 – Courbes d'apprentissage pour les encours après retrait des valeurs les plus excentrées

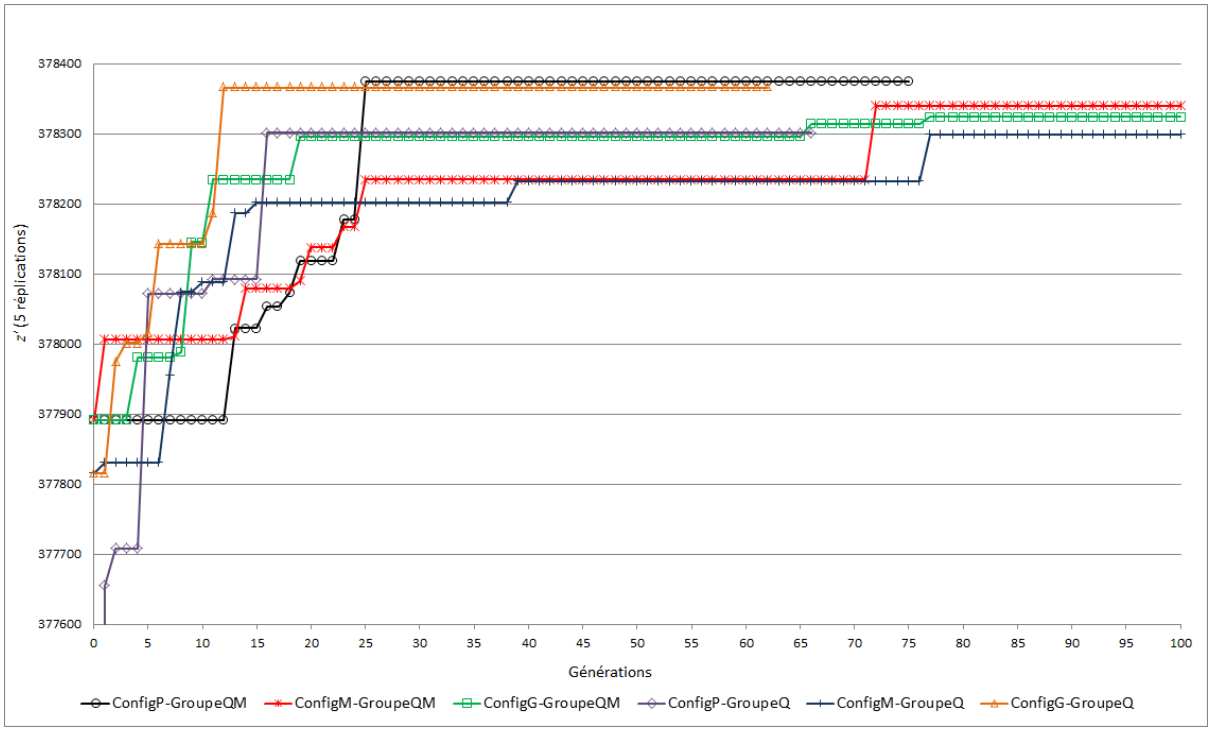

FIGURE 5.15 – Courbes d'apprentissage pour les livraisons après retrait d'une valeur excentrée

obtenue par $<Config^P$ - $Groupe_Q>$ vis-à-vis des autres expérimentations est considérable et ce, indépendamment de la fonction objectif. Si l'on considère les livraisons (z'), toutes les autres expérimentations semblent démarrer au même niveau. Quant aux encours (z), la très mauvaise performance de $<Config^P$ - $Groupe_Q>$ est suivie par celle de cette même configuration $Config^P$ (petite taille de population) appliquée au $Groupe_{QM}$. À l'opposé, les $Config^M$ et $Config^G$ sont de loin les meilleures lorsqu'appliquées au $Groupe_Q$ qui n'utilise pas l'information sur la répartition courante des machines.

En associant ces données à celles des Figures 5.14 et 5.15, nous pouvons constater que malgré leur qualité de départ différente, les courbes d'apprentissage montrent en général une bonne convergence. De même, les performances atteintes par les différentes expérimentations à la fin de leur processus d'apprentissage s'avèrent très proches. Tout au long de l'évolution, la PGL a donc encore une fois été capable d'améliorer considérablement la capacité de ses logiques décisionnelles à prendre de bonnes décisions. Ceci confirme sa capacité à apprendre.

Notons également qu'une meilleure solution de départ ne conduit pas forcément à une meilleure solution finale. Les deux expérimentations démarrant avec les meilleures performances pour les encours (z) ont fini avec les moins bonnes, conjointement à celle qui avait déjà la pire performance au départ.

D'autres constations peuvent encore être faites par rapport à ces courbes :

- Si z est retenu, le $Groupe_{QM}$ ayant accès à l'information sur la répartition courante des machines s'avère le plus performant, et ce indépendamment de la configuration : le complément d'information obtenu avec la variable d'état $NbM_{A,t}$ associée à celle de décision dans S_t semble ici bénéfique ;
- Si par contre c'est z' qui est retenu (donc les livraisons), les meilleures performances sont attribuées à deux cas « opposés » : une petite population ($Config^P$) ayant accès à la répartition courante des machines ($Groupe_{QM}$) ou alors une grande population ($Config^G$) ne l'ayant pas ($Groupe_Q$). De ce fait, la prise en compte de la variable d'état associée à celle de décision n'est pas cette fois-ci concluante ;
- Une expérimentation doit toujours être évaluée vis-à-vis du critère de performance pour lequel elle a été appliquée. Individuellement, la convergence ne sera souvent pas la même pour deux critères différents. Il en est de même pour le critère d'arrêt déterminant, c'est-à-dire, celui qui déclenche effectivement l'arrêt du processus d'apprentissage. En présence de plusieurs expérimentations, il va de soi que ce n'est pas forcément la même qui obtiendra la meilleure performance pour les deux critères.

Bien que nous en ayons tiré un certain nombre de conclusions, les valeurs de performance indiquées jusque-là ne sont que des estimations relatives à un faible nombre de réplications (5). Toute analyse faite doit ainsi être validée après la deuxième étape de notre approche qui consiste à simuler 30 réplications de la meilleure solution pour chaque expérimentation (Section 5.3.3.1). C'est seulement une fois cette étape accomplie qu'elles seront comparables statistiquement, ce qui est fait via des tests t de Student.

Avant de passer à la suite, nous indiquons dans le Tableau 5.25 le temps de calcul qu'il a fallu à notre approche d'apprentissage pour obtenir les résultats que nous venons d'analyser. Les valeurs (colonne *Temps*) sont données en heures. Il faut donc entre 16,24 heures et 3,76 jours de calcul hors ligne pour que le processus d'apprentissage soit conclu. Nous indiquons également le nombre de solutions évaluées pour chaque expérimentation car le temps de calcul en dépend fortement (colonne *NbEval*). D'après ces données, une évaluation requiert en moyenne 25,56 secondes. Rappelons que l'évaluation d'une solution lors de l'apprentissage représente 5 réplications d'un modèle simulé sur 2 ans de fonctionnement du système. Une fois encore, l'utilisation en ligne d'une logique décisionnelle est tout à fait envisageable puisque les décisions d'adaptation sont données quasiment instantanément.

Expérimentations	z		z'	
	Temps (h)	*NbEval*	*Temps (h)*	*NbEval*
$<Config^P$ - $Groupe_Q>$	16,24	2857	24,92	2608
$<Config^M$ - $Groupe_Q>$	42,93	5721	61,53	7590
$<Config^G$ - $Groupe_Q>$	87,31	13773	53,06	7621
$<Config^P$ - $Groupe_{QM}>$	23,39	3378	22,09	2694
$<Config^M$ - $Groupe_{QM}>$	41,22	6865	49,64	7415
$<Config^G$ - $Groupe_{QM}>$	88,44	13106	90,18	13944

Tableau 5.25 – Temps de calcul et nombre d'évaluations pour les différentes expérimentations avec la PGL pour le problème de répartition de machines

5.3.4.1.2 Résultats quantitatifs

Le Tableau 5.26 indique les performances de la meilleure solution pour chaque paire <configuration - groupe> cherchant à optimiser le nombre estimé z' de produits livrés. Le meilleur *fitness* obtenu est mis en évidence. Le Tableau 5.27 est construit de la même façon mais pour l'objectif z (encours moyens estimés). Rappelons que l'estimation du temps moyen de passage d'un produit est aussi indiquée (FTM). Enfin, dans ces tableaux, nous avons également ajouté dans une colonne nommée *NbDep* le nombre de déplacements effectués durant les 2 ans que représente la période d'étude (valeur estimée après 30 réplications). Ceci nous permettra de réaliser un complément à notre analyse.

Nous commencerons notre analyse par la comparaison de nos résultats aux deux scénarios de base retenus : *Automatique* pour de « bonnes » décisions d'adaptation et *Statique* pour l'absence d'adaptations (revoir Tableau 5.20).

Lorsque l'on s'intéresse à l'optimisation du nombre de produits livrés (Tableau 5.26), toutes nos solutions surpassent le cas statique. Le gain estimé est de 102,7 à 310,8 produits supplémentaires livrés, ce qui représente entre 0,027% et 0,082%. Comparés au cas automatique, nos résultats varient d'une perte estimée de 14,5 produits (-0,004%) à un

Expérimentations	Critère actif	Critères passifs		
	z'	z	FTM	$NbDep$
$<Config^P$ - $Groupe_Q>$	377515,67	66,5830	208,74	8802,17
$<Config^M$ - $Groupe_Q>$	377498,40	21,8947	61,48	725,33
$<Config^G$ - $Groupe_Q>$	**377697,93**	41,6019	116,08	9030,07
$<Config^P$ - $Groupe_{QM}>$	377544,63	23,1486	66,69	1205,73
$<Config^M$ - $Groupe_{QM}>$	377619,67	60,1000	202,06	11174,20
$<Config^G$ - $Groupe_{QM}>$	377489,83	50,5769	169,30	10039,67

Tableau 5.26 – Performances pour l'optimisation des livraisons

Expérimentations	Critère actif	Critères passifs		
	z	z'	FTM	$NbDep$
$<Config^P$ - $Groupe_Q>$	19,4985	377359,17	57,33	532,60
$<Config^M$ - $Groupe_Q>$	18,9774	377299,30	56,92	251,07
$<Config^G$ - $Groupe_Q>$	17,7497	377564,97	52,08	197,23
$<Config^P$ - $Groupe_{QM}>$	16,0521	377315,90	46,84	107,93
$<Config^M$ - $Groupe_{QM}>$	15,4403	377460,00	45,41	58,07
$<Config^G$ - $Groupe_{QM}>$	**15,3540**	377382,27	45,08	24,43

Tableau 5.27 – Performances pour l'optimisation des encours moyens

gain estimé de 193,6 produits (0,051%). Notre meilleure performance est donc supérieure mais d'autres sont moins bonnes comme prévu. Ces différences relatives ont été calculées par la formule $\frac{z'[PGL]-z'[Base]}{z'[Base]}$ où $z'[PGL]$ indique la performance atteinte par l'une des expérimentations de notre approche et $z'[Base]$ correspond à celle du cas statique ou automatique suivant la comparaison faite.

La meilleure performance est attribuée à $<Config^G$ - $Groupe_Q>$ et non pas à $<Config^P$ - $Groupe_{QM}>$ tel qu'attendu d'après la Figure 5.15. Cette dernière expérimentation se retrouve en troisième position. Ceci montre clairement qu'un apprentissage basé sur seulement 5 réplications peut conduire à des « erreurs » lors des différentes comparaisons entre individus. En effet, puisque les estimations ne sont pas assez précises, un individu estimé meilleur qu'un autre peut s'avérer être moins bon après un plus grand nombre de réplications, ces dernières étant nécessaires pour réaliser une comparaison statistique. Ces erreurs sont cependant acceptables compte tenu du temps de calcul qui aurait été nécessaire pour que le processus soit davantage fiable.

Toujours d'après les résultats pour les livraisons (z'), nous pouvons observer que le nombre moyen de déplacements induits est assez important : entre 0,99 et 15,31 par jour. En passant maintenant à l'optimisation des encours (critère de performance z, Tableau 5.27), ce nombre se réduit considérablement : entre 1,02 et 22,19 déplacements mais cette fois-ci par mois. Ici encore, toutes nos solutions surpassent de loin le cas statique avec un gain estimé pour z entre 98,93% et 99,15%. Le cas automatique représente cependant

un seuil inférieur à nos solutions dont l'écart estimé est situé entre 3,45% et 31,38%. Bien évidemment, puisqu'il s'agit maintenant d'un problème de minimisation, la formule calculant les différences relatives a été adaptée : $\frac{z[Base]-z[PGL]}{z[Base]}$.

Le nombre total de commandes reçues par le système correspond à la somme entre les encours et les produits livrés. Les critères de performance z et z' ne sont donc pas à priori contradictoires. En revanche, puisque z est minimisé d'après sa valeur moyenne pendant toute la période d'étude, celle-ci peut osciller pendant la simulation. z' est quant à lui maximisé en considérant sa valeur courante à la fin de la période, donc il ne fait qu'augmenter ou au pire stagner au fur et à mesure de la simulation. Ainsi, z' reflète moins la performance générale du système : si les demandes livrées l'ont été sans aucun délai ou avec de très longs temps d'attente, la performance qui en découle sera exactement la même. Minimiser les encours moyens (z) nous semble plus réaliste et pourrait par ailleurs favoriser une meilleure cadence de la production puisque FTM semble proportionnellement lié à ce critère d'après les valeurs obtenues. Par conséquent, seul z sera conservé dans les analyses qui suivent, les autres critères apparaissant à titre indicatif.

5.3.4.2 Analyse de la nervosité

5.3.4.2.1 Description

Nous avons pu remarquer que le nombre de changements pour arriver à de telles performances n'est pas négligeable. Il est aussi peu réaliste pour un environnement réel. Cette constatation nous a conduits à chercher une solution afin de réduire la nervosité du système (Section 2.3.1.2).

Nous avons constitué un nouveau scénario de test qui peut être décrit comme suit. D'une part, nous ajoutons le nombre de changements comme deuxième critère de performance de notre problème. Puisque ce n'est pas un objectif préalablement identifié, nous nous limitons ici à une optimisation dite hiérarchique : nous cherchons à minimiser en priorité le niveau des encours. Pour deux solutions équivalentes vis-à-vis de ce critère, celle impliquant le moins de changements est la meilleure. D'autre part, nous fixons $H_{min,1} = 1$ jour. Bien que permettre un déplacement par jour ne semble toujours pas réaliste, ceci constitue la deuxième et dernière valeur possible pour $H_{min,1}$ (Section 5.3.2.1), ce qui suppose des machines très mobiles (donc faciles à déplacer et à remettre en production).

5.3.4.2.2 Résultats quantitatifs

Le Tableau 5.28 ci-dessous détaille les résultats obtenus pour ce nouveau groupe de test.

Nous pouvons remarquer que la solution proposée reste très simple mais efficace. Seule

Expérimentations	Critères actifs		Critères passifs	
	z (1^{er})	$NbDep$ ($2^{ème}$)	z'	FTM
$<Config^P$ - $Groupe_Q>$	15,6765	18,20	377364,60	45,89
$<Config^M$ - $Groupe_Q>$	15,7219	14,33	377339,27	46,16
$<Config^G$ - $Groupe_Q>$	15,7481	18,63	377397,30	46,22
$<Config^P$ - $Groupe_{QM}>$	43,0174	721,67	377293,20	139,05
$<Config^M$ - $Groupe_{QM}>$	**15,5787**	**12,67**	377526,17	45,65
$<Config^G$ - $Groupe_{QM}>$	15,7113	21,27	377323,50	46,50

Tableau 5.28 – Performances pour l'optimisation des encours puis du nombre de déplacements avec interdiction de plus d'un déplacement par jour

l'expérimentation $<Config^P$ - $Groupe_{QM}>$ a présenté une véritable dégradation de performance, c'est pourquoi nous n'en tiendrons pas compte dans les analyses qui suivent. Avec le $Groupe_Q$ qui n'a pas d'information sur la répartition courante des machines, les valeurs obtenues pour les encours (z) sont significativement plus performantes qu'avant (revoir Tableau 5.27), et ce d'après des tests t de Student. Par ailleurs, nous avons maintenant au maximum un changement par mois, voire un tous les deux mois. Ceci s'avère bien plus acceptable d'autant plus que les performances restent bonnes : nous avons au maximum 2,57% de perte estimée par rapport à la meilleure performance lorsque $H_{min,1} = 1$ heure. Cela représente un écart estimé de 6,10% vis-à-vis du seuil inférieur (scénario *Automatique*, Tableau 5.20).

5.3.4.3 Analyse de l'arbre de décision généré

Complétons l'interprétation des résultats de cet exemple par une analyse qualitative en utilisant la logique décisionnelle obtenue par l'expérimentation $<Config^G$ - $Groupe_{QM}>$ du Tableau 5.27, c'est-à-dire celle ayant la meilleure performance estimée. Son code effectif est illustré dans la Figure 5.16.

Les informations effectives sont ici facilement identifiables et nous permettent d'évaluer la cohérence de ce qui est proposé dans les différentes circonstances. Nous analyserons les neuf décisions une à une, de la gauche vers la droite. Rappelons que -1 et +1 indiquent respectivement un déplacement de l'$Atelier_A$ vers l'$Atelier_B$ et vice-versa.

Pour la première décision, l'$Atelier_A$ contient 6 machines et aucun déplacement ne sera fait. Cette décision semble problématique : une fois $NbM_{A,t} = 6$, plus aucune adaptation ne peut être faite. Nous verrons cependant que les deux prochaines décisions ne permettent pas d'atteindre cette situation, ce qui la rend donc sans conséquence. En effet, si l'$Atelier_A$ contient 5 machines, deux seules décisions sont envisagées. Ainsi, la deuxième décision correspond au cas où l'$Atelier_B$ s'auto-suffit n'induisant pas de déplacement. La troisième montre quant à elle qu'une machine sera déplacée vers l'$Atelier_B$ si celui-ci est surchargé ($nq_{B,t} \geq 37$). Puisqu'un seul déplacement est fait à la fois, seul $NbM_{A,t} = 5$ pourrait être

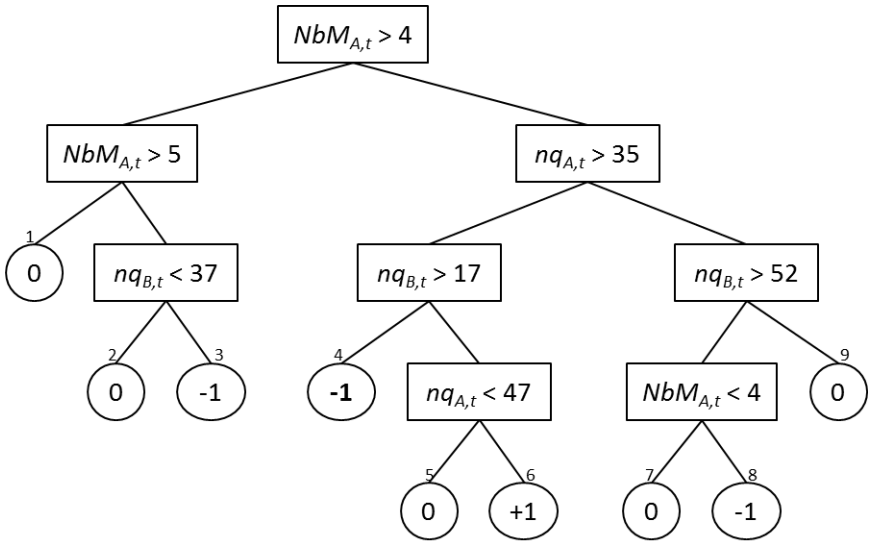

FIGURE 5.16 – Meilleure solution pour l'adaptation de la répartition des machines

à l'origine de la répartition $\langle 6; 1 \rangle$ mais jamais une décision dans ce sens n'est proposée.

Pour les six prochaines décisions, l'$Atelier_A$ compte au plus 4 machines. Si ce même atelier a une certaine charge ($nq_{A,t} > 35$) mais que l'autre aussi ($nq_{B,t} > 17$), c'est l'$Atelier_B$ qui a la priorité, la quatrième décision indiquant qu'un déplacement sera fait vers lui. En revanche, si l'$Atelier_B$ a un nombre réduit de produits à traiter, soit l'$Atelier_A$ n'est pas surchargé et la cinquième décision n'entraînera aucun changement, soit il l'est ($nq_{A,t} \geq 47$) et recevra une machine grâce à la sixième décision.

Si maintenant la charge de l'$Atelier_A$ est limitée ($nq_{A,t} \leq 35$), deux cas se présentent. Le premier intervient lorsque la charge de l'$Atelier_B$ est élevée. Il entraîne la septième décision lorsque moins de 4 machines sont dans l'$Atelier_A$: rien ne sera changé. Il entraînera en revanche la huitième décision si cet atelier compte exactement 4 machines : un déplacement vers l'$Atelier_B$. Terminons par le deuxième cas, valable lorsque la charge de l'$Atelier_B$ est bornée à 52 produits. La neuvième et dernière décision résultante n'apportera aucun changement à la répartition des machines dans le système.

Notons que différents seuils sont utilisés pour évaluer la charge de nos deux ateliers. Ceci permet de réagir de façon plus appropriée et selon chaque situation spécifique. L'expertise entièrement construite par notre approche semble pertinente. Les déplacements sont déclenchés lorsqu'un atelier nécessite une machine supplémentaire mais jamais au détriment de l'autre atelier. De plus, l'algorithme a pu identifier dans une situation intermédiaire la priorité d'un atelier par rapport à l'autre. Il s'avère donc capable d'analyser certains cas plus délicats, moins évidents pour un décideur.

Par son format simple, l'arbre peut être interprété et utilisé directement par des gestionnaires. Cette simplicité de lecture nous permet également de vérifier la validité des actions à entreprendre. Une seule des actions suggérées est susceptible de violer la Contrainte 5.4, il s'agit de celle en gras dans la Figure 5.16. En effet, le déplacement d'une machine de

l'$Atelier_A$ vers l'$Atelier_B$ n'est pas possible si le premier atelier ne contient qu'une seule machine. Cependant, cette information n'est pas prise en compte dans le cheminement jusqu'au déclenchement de la décision. Une correction est faite par le modèle de simulation si besoin est, ce qui nous conduit à la logique décisionnelle de la Figure 5.17. Notons que dans cette dernière, nous avons également éliminé la première branche qui n'était pas atteignable.

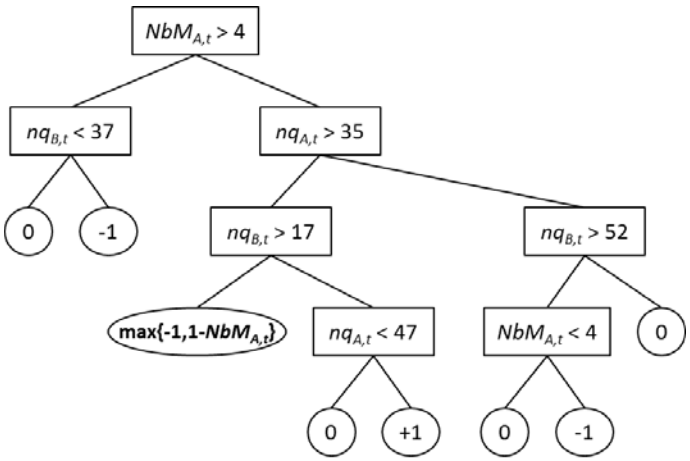

FIGURE 5.17 – Meilleure solution pour l'adaptation de la répartition des machines après correction

5.4 Adaptation simultanée de la répartition des machines et de la politique de réapprovisionnement dans un réseau intra-entreprise

5.4.1 Contexte

Ce troisième et dernier cas d'étude s'applique aussi à un réseau logistique intra-entreprise. Il s'agit plus précisément d'une adaptation du cas précédent de façon à le rendre plus complexe. L'objectif est d'analyser l'efficacité de notre approche lors du traitement de plus d'une décision à la fois. Pour ce faire, le cas inclut la notion de politique de réapprovisionnement (Figure 5.18). Pour plus de détails sur cette dernière, se référer à [Giard 2003, Ben Ammar 2014] ou également à des travaux tels que [Whybark et Williams 1976, Grasso et Taylor III 1984] qui ont cherché à comparer différentes politiques appliquées à des environnements variés.

Signalons que, à notre connaissance, ce problème combiné que nous proposons de traiter n'a jamais fait l'objet de publications dans la littérature.

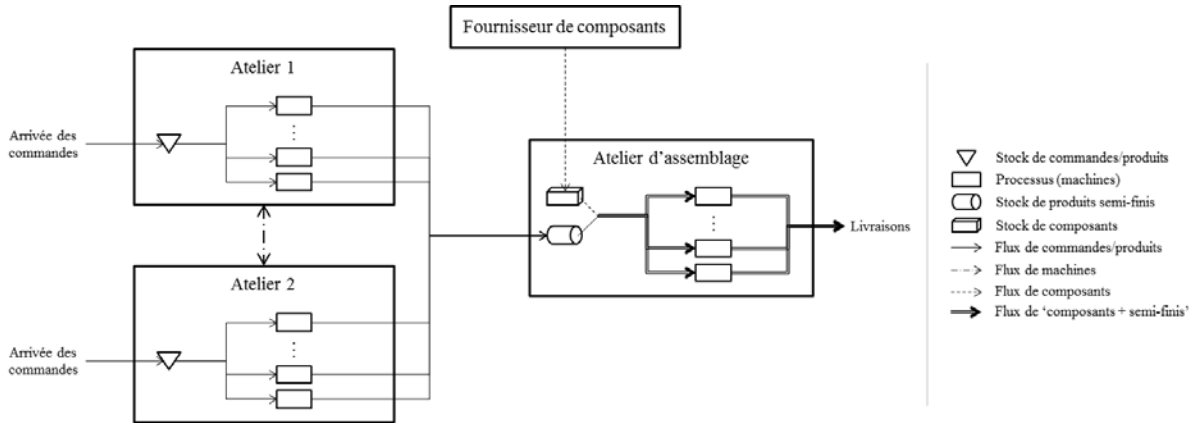

FIGURE 5.18 – Schéma simplifié d'un système à ressources partagées et incluant l'approvisionnement

5.4.2 Description du système

5.4.2.1 Éléments constituant le système

Ce nouveau système de production possède deux gammes principales de produits, $Produit_A$ et $Produit_B$, chacune subissant respectivement des opérations dans l'$Atelier_A$ et l'$Atelier_B$. Ces ateliers restent comme dans l'exemple précédent dédiés à une gamme de produit exclusive. Nous gardons l'indice $w \in W = \{A; B\}$ pour faire référence à la gamme de produit ou désigner l'atelier associé. Le système compte un total de $Cap = 7$ machines polyvalentes pouvant produire le $Produit_A$ ou le $Produit_B$ suivant l'atelier où elles se trouvent. Les temps de traitement suivent une distribution exponentielle dont la moyenne est de 15 minutes, et ce indépendamment de la machine et de la gamme de produit : $\mu_A = \mu_B = \frac{1}{15}$.

Le processus de fabrication d'un produit est ici agrémenté d'une étape supplémentaire. Ainsi, un produit de gamme w, après traitement sur une machine de l'$Atelier_w$, subit en plus une opération d'assemblage. Cette dernière est réalisée dans un troisième atelier dédié (à l'assemblage) dont la capacité est toujours suffisante et qui peut traiter aussi bien le $Produit_A$ que le $Produit_B$. Il arrive cependant que les composants à assembler ne soient pas toujours disponibles en quantités nécessaires. Pour être accomplie, l'opération d'assemblage requiert un produit semi-fini (sortant de l'un des deux ateliers en amont) ainsi que la quantité exacte de composants nécessaire à ce dernier.

En fait, chacune des gammes principales de produits compte ici un certain nombre de variantes. Ces dernières diffèrent uniquement dans l'assemblage par leur nombre de composants qui peut varier de 1 à 25, le plus courant étant de 10.

Le système possède une zone de stockage pouvant accueillir un maximum de $SC_{Max} = 43500$ unités de ces composants dont la fabrication est sous-traitée à un fournisseur unique. Ce dernier travaille selon deux politiques d'approvisionnement possibles. La première, dite

périodique et notée $Politique_{\Delta}$, se base sur une livraison à des intervalles de temps fixes, ici une fois par semaine ($\Delta_{reappro} = 7 \cdot 24h$). Dans ce cas, la commande stipulant la quantité exacte de composants à livrer doit être passée 1 heure avant la livraison ($\delta F_{\Delta} = 1h$). Cette quantité est calculée par la formule $SC_{Max} - SC_t$ où SC_t est la quantité en stock au moment de la commande. La deuxième politique d'approvisionnement se base quant à elle sur un seuil d'alerte. Elle est dite à point de commande et est notée $Politique_{Seuil}$. Ce seuil, établi ici à 5500 unités, déclenche une demande de réapprovisionnement visant à remplir de nouveau la zone de stockage. Ainsi, au moins 38000 unités seront commandées à chaque fois. Cette valeur peut aller jusqu'à 38024 lorsque l'on a 5501 unités en stock et que le produit déclenchant le réapprovisionnement en a sollicité le maximum possible (donc 25). Puisqu'ici le fournisseur contrôle moins la situation (il n'a pas de prévision sur la date de livraison), la commande doit être passée 1 jour à l'avance ($\delta F_S = 24h$).

Ces deux politiques sont basées sur celles couramment utilisées pour la gestion des stocks. Dans la littérature, la $Politique_{\Delta}$ est notée (T, S) alors que la $Politique_{Seuil}$ est notée (s, S). La première permet plus de maîtrise du processus comme un tout alors que la deuxième a tendance à être moins onéreuse mais les deux peuvent avoir des performances équivalentes selon la définition de la période et du seuil de réapprovisionnement [Giard 2003].

Le but dans ce système est de rester le plus rentable possible en termes de coûts de stockage. L'objectif G prend alors en compte les différents coûts engendrés par les produits, que ce soit avant tout traitement, durant ou après la première étape de leur processus de fabrication. Considéré comme un coût fixe, l'assemblage n'est quant à lui pas pris en compte. En revanche, les coûts concernant le stockage de composants le sont. Le capital immobilisé par la détention d'un produit semi-fini est 10 fois plus élevé que celui avant et durant le traitement par l'$Atelier_A$ ou l'$Atelier_B$. Ce dernier est quant à lui de 1 mais reste 10000 fois plus élevé que celui pour le stockage des composants. La fonction objectif est donc donnée par la minimisation de ces coûts moyens pour la période d'étude :

$$Minimiser\ z = \mathbb{E}[\overline{wip_A} + \overline{wip_B} + 10 \cdot \overline{wip_{SF}} + 10^{-4} \cdot \overline{wip_{comp}}] \tag{5.10}$$

où $\overline{wip_w}$ correspond aux encours moyens de la gamme w jusqu'à la fin de leur traitement dans l'$Atelier_w$, $\overline{wip_{SF}}$ au nombre moyen de semi-finis en attente de composants pour l'assemblage et $\overline{wip_{comp}}$ au nombre moyen de composants en stock. Aucune différentiation n'est faite quant au capital immobilisé selon les gammes principales et/ou leurs variantes. Notons que $\overline{wip_w}$ est lié à la bonne répartition des machines tandis que les deux autres le sont à une bonne gestion du stock.

5.4.2.2 Adaptabilité du système

Pour permettre de mieux atteindre les objectifs énoncés, l'aspect adaptatif du système concerne d'une part le nombre de machines sur l'$Atelier_A$ et l'$Atelier_B$ et d'autre part la politique de réapprovisionnement à adopter qui peut également varier. Nous avons cette

fois-ci deux objets modifiables ($J = 2$), chacun ayant un seul attribut variable ($K_1 = K_2 = 1$). Ainsi, $O = \{o_1; o_2\}$, où o_1 représente le groupement des ateliers à l'exception de celui d'assemblage et o_2 le stock de composants. Pour pouvoir tirer profit de la flexibilité du système, nous comptons sur deux types de changement ($L = 2$) : $C = \{c_1; c_2\}$. Le premier, c_1, représente le déplacement de machines. Le second, c_2, représente quant à lui le changement de politique de réapprovisionnement.

Cette description sera détaillée dans ce qui suit. Il en sera de même pour les deux variables de décision ($N = 2$) qui découlent du problème d'adaptation associé.

5.4.2.2.1 Répartition de machines

Concentrons-nous d'abord sur le premier objet, à savoir le groupement des ateliers, pour lequel les observations sont les mêmes que pour l'exemple précédent. Elles seront reprises ici mais de façon plus succincte. Le fait d'avoir un unique objet qui réunit l'$Atelier_A$ et l'$Atelier_B$ est explicable mais surtout cohérent avec le problème traité : nous ne nous intéressons pas ici à la position précise de chaque machine au sein d'un atelier mais uniquement à leur appartenance à un atelier ou à l'autre. Un déplacement de machine est donc fait entre les ateliers, ces derniers étant modifiés par la même occasion.

Le vecteur d'attributs variables correspondant est $A_1 = \langle a_{1,1} \rangle$ où $a_{1,1}$ représente la répartition des machines. Si l'on considère son instanciation dans le temps, $a_{1,1,t} = \langle NbM_{A,t}; NbM_{B,t} \rangle$, où chaque composante $NbM_{w,t}$ correspond au nombre de machines à l'instant t dans un $Atelier_w$ donné. Par ailleurs, il doit y avoir à tout instant au minimum une machine dans chaque $Atelier_w$ de ce système. Puisque le nombre total de Cap machines est fixe et réparti entre les deux ateliers ($NbM_{A,t} + NbM_{B,t} = Cap = 7, \forall t$), il n'est pas possible d'en avoir plus que $Cap - 1 = 6$ dans chaque $Atelier_w$. Ceci peut être exprimé par la Contrainte 5.11 qui, lorsqu'elle est satisfaite pour un atelier, l'est également pour l'autre :

$$1 \leq NbM_{w,t} \leq 6, \forall w, t \tag{5.11}$$

Nous avons donc toutes les répartitions possibles des machines dans le système : $E_{1,1} = \{\langle 1; 6 \rangle; \langle 2; 5 \rangle; \langle 3; 4 \rangle; \langle 4; 3 \rangle; \langle 5; 2 \rangle; \langle 6; 1 \rangle\}$. Pour pouvoir les exploiter, la variable de décision $d_{1,t}$ est utilisée. Comme décrit dans l'exemple précédent, il s'agit d'un entier qui précise le nombre de machines à déplacer par sa valeur absolue et la direction du déplacement par son signe. Dans notre système, une seule machine peut être déplacée à la fois, ainsi $D_1 = [-1; 1]$: $+1$ représente un déplacement de l'$Atelier_B$ vers l'$Atelier_A$, -1 de l'$Atelier_A$ vers l'$Atelier_B$ alors que 0 est l'action par défaut consistant à ne rien déplacer. La nouvelle répartition $\langle NbM_{A,t}; NbM_{B,t} \rangle := \langle NbM_{A,t} + d_t; NbM_{B,t} - d_t \rangle$ doit satisfaire la Contrainte 5.11.

Les déplacements entre ateliers prennent $H_{durée,1} = 1$ heure et sont ici associés à l'horizon décisionnel $H_{D,1}$ d'ordre tactique. À compter du début d'un déplacement, aucun

autre ne pourra être fait pendant $H_{min,1} = 1$ jour. Une telle valeur de $H_{min,1}$ assure une stabilité minimale au système tout comme le non chevauchement de deux déplacements lorsque le premier a déjà été démarré.

5.4.2.2.2 Politique de réapprovisionnement

Passons maintenant au deuxième objet, le stock de composants, pour lequel plus de précisions devront être données. Nous avons le vecteur d'attributs variables $A_2 = \langle a_{2,1} \rangle$ où $a_{2,1}$ représente la politique de réapprovisionnent, son instanciation dans le temps $a_{2,1,t}$ indiquant celle en cours à l'instant t. Deux seules possibilités sont proposées par le fournisseur, $Politique_{\Delta}$ et $Politique_{Seuil}$. Pour traiter plus facilement cet attribut qualitatif, elles seront représentées ici par des entiers : la première par 1 et la seconde par 2. Ainsi $E_{2,1} = \{1; 2\}$.

Pour tirer profit de cette flexibilité permise par le fournisseur, la variable de décision $d_{2,t}$ est utilisée dans la formulation de notre problème d'adaptation. Nous nous servons ici encore d'un entier pour lequel trois valeurs sont possibles : $+1$ représente un changement vers la $Politique_{Seuil}$ (donc vers 2), -1 vers la $Politique_{\Delta}$ (donc vers 1) et l'action par défaut 0 indique quant à elle de rester sur la politique en cours. La nouvelle politique est obtenue par la formule $a_{2,1,t} =: a_{2,1,t} + d_{2,t}$. Pour que seules les valeurs autorisées soient obtenues, ces actions sont redéfinies comme suit :

$$d_{2,t} := \begin{cases} min\{d_{2,t}; 2 - a_{2,1,t}\} & \text{si } d_{2,t} = 1 \\ max\{d_{2,t}; 1 - a_{2,1,t}\} & \text{si } d_{2,t} = -1 \\ d_{2,t} & \text{si } d_{2,t} = 0 \end{cases} \tag{5.12}$$

Ainsi, les actions pour changer vers une politique donnée reviendront parfois à ne rien faire puisque la politique souhaitée correspond à celle déjà adoptée.

Le temps de mise en place d'un changement de politique implique des procédures administratives mais pas de contraintes au niveau du fonctionnement du système. Il est donc négligeable : $H_{durée,2} = 0$. Il n'en est cependant pas de même pour son temps minimal de validité $H_{min,2}$. Justement et en raison de l'aspect administratif, il ne serait pas pertinent de changer sans arrêt une politique de réapprovisionnement (voir discussion sur la nervosité dans la Section 2.3.1.2). La flexibilité ici offerte ne l'est donc que de façon limitée. Cette limitation est ici exprimée sous la forme d'une durée minimale $H_{min,2}$ lors de chaque contrat établit avec le fournisseur spécifiant la politique pour la livraison de composants. Dans notre cas, $H_{min,2} = 30$ jours, ce qui nous mène à un horizon décisionnel $H_{D,2}$ également d'ordre tactique. Le même type de décision serait cependant d'ordre stratégique si le contrat était plus restrictif (c'est-à-dire $H_{min,2}$ plus élevé).

Avant de détailler le fonctionnement du système d'une façon plus générale, signalons une de ses facettes induite par $H_{durée,2} = 0$. Lorsque le contrat est modifié de la $Politique_{\Delta}$ vers la $Politique_{Seuil}$, le prochain réapprovisionnement est déclenché dès que le seuil

d'alerte est franchi, y compris s'il l'est déjà. Dans ce cas, la commande risque de compter plus de composants que les 38000 habituellement commandés lorsque la $Politique_{Seuil}$ est déjà en place. Lors d'une modification inverse, donc de la $Politique_{Seuil}$ vers la $Politique_{\Delta}$, la date de la prochaine commande est quant à elle calculée par rapport à la date de la commande qui lui précède selon la $Politique_{Seuil}$. Elle peut ici encore être immédiate si la dernière commande a été passée il y a plus longtemps que $\Delta_{reappro}$.

5.4.2.3 Fonctionnement du système

Tous ces éléments étant introduits, passons à la dynamique du système. Celui-ci démarre avec son processus de fabrication vide et le stock de composants rempli à son maximum SC_{Max}. Nous avons au départ 3 machines dans l'$Atelier_A$ et 4 dans l'$Atelier_B$. Le contrat en cours avec le fournisseur désigne la $Politique_{\Delta}$ pour le réapprovisionnement.

Le fonctionnement du système peut être décrit comme suit. Il n'y a pas de prévision sur la demande future et ce sont les commandes arrivant selon un processus de Poisson qui déclenchent la production. Toute commande arrive directement dans l'$Atelier_A$ ou l'$Atelier_B$ qui la concerne. Il n'y a pas de différenciation entre les machines dans un même atelier ni de priorité entre différentes commandes. Les commandes en attente d'une machine libre restent donc dans une file d'attente commune à toutes les machines d'un atelier et sont traitées selon leur ordre d'arrivée (FIFO).

Une fois le traitement initial d'un produit terminé, la machine correspondante est libérée. Ceci déclenche potentiellement le processus de prise de décision quant à l'adaptation ou non du système à cet instant. Si la machine libérée n'est pas sollicitée pour un déplacement, elle pourra alors se charger d'un nouveau produit en attente. Sinon, elle reste inopérable pendant la durée de son déplacement suite auquel elle pourra se charger d'un produit dans son atelier de destination. Notons que la décision de déplacement peut également concerner non pas la machine qui l'a déclenchée mais une machine de l'atelier « voisin ». Par ailleurs, la décision sur la politique de réapprovisionnement est également prise à cette même occasion.

Un produit dont le traitement est fini se dirige vers la prochaine étape du processus, c'est-à-dire vers l'atelier d'assemblage. Le temps de transport est négligeable. En revanche, pour que l'opération puisse démarrer, il lui faut un nombre de composants spécifique non pas à la gamme principale mais à la variante qu'il représente. Si ces composants ne sont pas tous disponibles, le produit reste dans une zone de stockage intermédiaire jusqu'à ce qu'une nouvelle livraison de composants soit faite. Puisque l'assemblage ne représente pas une contrainte au fonctionnement du système, le processus en lui-même n'est pas analysé : une durée symbolique d'une minute lui est attribuée suite à laquelle le produit est considéré comme livré au client.

C'est donc entre les deux étapes du processus de fabrication que les décisions d'adaptation peuvent être prises. Autrement dit, lors de la fin du traitement d'un produit par

l'$Atelier_A$ ou l'$Atelier_B$ et avant que son assemblage démarre. Ceci définit l'événement déclencheur pris en compte et partiellement notre stratégie de déclenchement T. Celle-ci concerne les deux types de décision pris en charge simultanément par ce système : le déplacement (ou non) d'une machine et le changement (ou non) de la politique de réapprovisionnement. Notons cependant qu'ils ont tous les deux des restrictions qui leur sont propres. Le premier est soumis à la condition de ne pas avoir un autre déplacement en cours, le deuxième à celle d'avoir franchi la période minimale établie pour le contrat avec le fournisseur. Ainsi, même si les deux décisions sont prises en simultané, une seule est parfois effective et donc effectuée. Si une condition n'est pas vérifiée à l'instant t, la décision spécifique qu'elle concerne sera annulée pour cet instant en lui associant la valeur par défaut de 0 (ne rien faire). Si en revanche aucune des conditions n'est vérifiée, la prise de décision sera « inhibée » : elle n'aura donc pas lieu car l'occurrence de l'événement déclencheur est sujette à la vérification d'au moins une des conditions préalables pour définir complètement T.

En accord avec la description du système, les deux décisions possibles cherchent à minimiser le coût total que représentent les encours et le stock de composants (objectifs dans G évalués par leur valeur moyenne sur une période d'étude). Elles seront toutes les deux prises par une logique décisionnelle unique centralisée (structure S). Cette logique sera trouvée ici par notre approche de programmation génétique linéaire basée sur la simulation (mécanisme de changement M). Elle se base sur le nombre de produits $nq_{w,t}$ à l'instant t dans la file d'attente de chaque $Atelier_w$ et éventuellement sur le nombre de machines $NbM_{A,t}$ à cet instant (donc indirectement sur $NbM_{B,t}$). Elle a également accès à l'écart-type du nombre de composants utilisés par produit. Noté $varComp_t$ et calculé à partir des 100 dernières commandes entrées dans le système (historique), il nous donne un indicateur sur la variabilité dans la demande par composants. Ainsi, $nq_{A,t}$, $nq_{B,t}$, $NbM_{A,t}$ et $varComp_t$ constituent notre vecteur de variables d'état S_t, partie dynamique de notre ensemble d'informations I.

Nous venons de montrer que notre cadre conceptuel et notre formalisation sont appropriés à des systèmes d'où découle un problème « multi-décision ».

5.4.2.4 Variation de la demande

Pour illustrer notre approche, nous avons introduit dans notre modèle de simulation des variations au niveau de l'arrivée des commandes au cours du temps. Ces variations auxquelles le système devrait s'adapter sont régies par l'évolution de la moyenne d'un processus de Poisson. Plus précisément, les changements des durées moyennes inter-arrivées ($dia_w = \frac{1}{\lambda_w}$) peuvent être représentés en termes de gammes principales ($Produit_A$ et $Produit_B$) ou de leurs variantes (besoin de composants). Deux *patterns* seront étudiées : $Pattern_{3M,6S}$ et $Pattern_{6M,3S}$ pour lesquelles les valeurs des dia_w seront indiquées en minutes.

Ici, les demandes pour les $Produit_A$ et $Produit_B$ s'inversent littéralement : au départ nous avons $dia_A = 5,95$ et $dia_B = 4,3$, puis c'est le contraire et ainsi de suite tout au long de la période d'étude. Ces changements faits tous les 3 mois pour la $Pattern_{3M,6S}$ et tous les 6 mois pour la $Pattern_{6M,3S}$ (Figure 5.19) devraient engendrer des besoins de déplacements de machines.

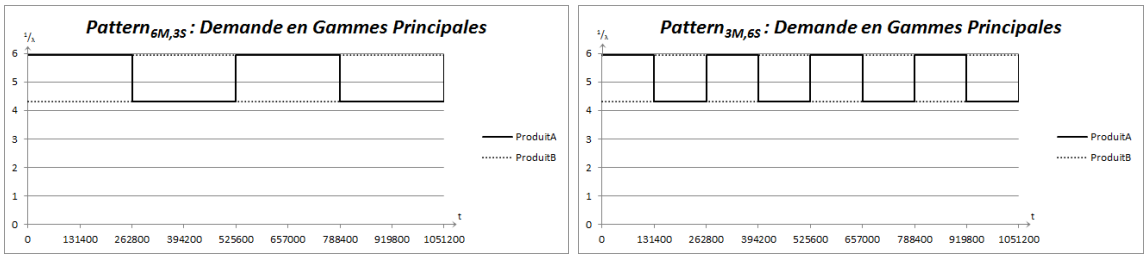

FIGURE 5.19 – *Patterns* utilisées pour la variation des gammes principales dans la simulation

Si maintenant ces demandes sont détaillées selon leur variante, le besoin en composants peut être plus au moins « dispersé ». La proportion des variantes fait que deux situations se présentent. Dans la première, 75% des commandes requièrent 10 composants alors que les 25% restant en requièrent entre 1 et 25. Dans la seconde, seulement 10% des commandes requièrent 10 composants alors que 90% en requièrent entre 1 et 25. La portion variable est modélisée par une loi triangulaire de mode 10. Au départ le système se trouve dans la première situation. Les changements se font tous les 3 mois pour la $Pattern_{6M,3S}$ et tous les 6 mois pour la $Pattern_{3M,6S}$ (Figure 5.20). Dans la littérature, il a été identifié qu'une $Politique_{Seuil}$ est plus adéquate lorsque le système est confronté à des variations au niveau de la demande [Whybark et Williams 1976, Grasso et Taylor III 1984] tandis que le délai de réapprovisionnement inférieur de la $Politique_{\Delta}$ devrait favoriser cette dernière dans des conditions plus stables. Ainsi, les variations de la proportion des variantes devraient être à l'origine de changements de politique de réapprovisionnement.

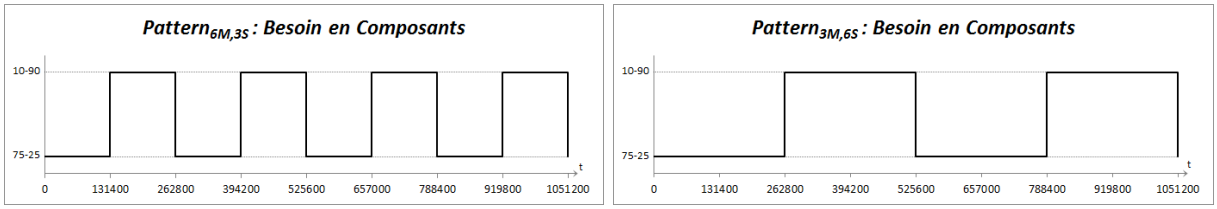

FIGURE 5.20 – *Patterns* utilisées pour la variation du besoin de composants dans la simulation

5.4.3 Conditions d'expérimentations

5.4.3.1 Généralités

Les conditions d'expérimentation sont toujours les mêmes que celles énoncées dans la Section 5.2.3.1. Rappelons que nous avons un apprentissage sur la base de 5 réplications de la simulation, puis la meilleure solution est simulée de nouveau sur 30 réplications.

Comme pour l'exemple précédent, nous définissons pour nos expérimentations une durée de simulation équivalente à 2 ans de fonctionnement du système de production (Section 5.3.3.1). En absence de travaux connus dans la littérature, les comparaisons seront encore une fois faites par rapport à des cas « statique » et « automatique » (Section 5.3.3.3), ces derniers faisant l'objet de la section qui suit.

5.4.3.2 Valeurs de référence pour la comparaison de nos résultats

Nous définissons le cas automatique pour chacune des deux *patterns* utilisées comme suit : le changement dans la moyenne des durées inter-arrivées des demandes tout comme celui dans le besoin de composants sont détectés sans aucun délai et l'adaptation la plus adéquate est mise en œuvre. Celle-ci peut concerner le déplacement de machines et/ou le changement de politique de réapprovisionnement :

- La répartition des machines commence à $\langle 3; 4\rangle$. Avec la $Pattern_{3M,6S}$, elle passe au bout de 3 mois à $\langle 4; 3\rangle$ pour revenir 3 mois plus tard à la situation de départ ($\langle 3; 4\rangle$) et ainsi de suite. La $Pattern_{6M,3S}$ reprend la précédente mais avec un intervalle de 6 mois.
- La politique de réapprovisionnement commence quant à elle avec la $Politique_{\Delta}$. Avec la $Pattern_{3M,6S}$, elle passe au bout de 6 mois vers la $Politique_{Seuil}$ pour revenir 6 mois plus tard vers la $Politique_{\Delta}$ de départ et ainsi de suite. Avec la $Pattern_{6M,3S}$, seul change l'intervalle qui est de 3 mois.

Comme pour l'exemple précédent, la détection d'un changement et la définition de l'adaptation la plus adéquate ont été possibles grâce à la connaissance de la *pattern* utilisée (Figures 5.19 et 5.20) et en ayant recours à la simulation.

Le cas statique quant à lui ne change pas avec la *pattern*. Il correspond à une répartition fixe des machines et à une même politique de réapprovisionnement tout au long de la période d'étude. Puisque nous avons vu pour l'exemple précédent que parmi les différentes possibilités, les plus pertinentes sont celles qui correspondent à une période de la *pattern* utilisée, nous considérerons les 4 combinaisons suivantes : une répartition $\langle 3; 4\rangle$ avec une $Politique_{\Delta}$; $\langle 3; 4\rangle$ avec $Politique_{Seuil}$; $\langle 4; 3\rangle$ avec $Politique_{\Delta}$; $\langle 4; 3\rangle$ avec $Politique_{Seuil}$. Nous avons simulé 30 réplications de 2 ans de fonctionnement du système avec chacune d'entre elles pour en obtenir les performances (Équation 5.10), et ce pour les deux *patterns*. Les résultats sont explicités dans le Tableau 5.29.

	$Pattern_{3M,6S}$	$Pattern_{6M,3S}$
$\langle 3;4\rangle$ - $Politique_{\Delta}$	2222,7077	3435,2455
$\langle 3;4\rangle$ - $Politique_{Seuil}$	2096,5273	3343,4186
$\langle 4;3\rangle$ - $Politique_{\Delta}$	2552,1474	4597,0482
$\langle 4;3\rangle$ - $Politique_{Seuil}$	2410,7416	4453,5859

Tableau 5.29 – Performances pour les cas statiques de répartition de machines et de politique de réapprovisionnement

Parmi ces cas sans changements, nous retenons le « meilleur » pour nos comparaisons qui suivront, à savoir la répartition $\langle 3;4\rangle$ avec la $Politique_{Seuil}$ pour les deux *patterns*. Le Tableau 5.30 reprend ce cas et ajoute les performances obtenues pour le cas automatique que nous utiliserons également. La différence considérable entre les deux cas, et ce indépendamment de la *pattern* en cours, témoigne de l'importance d'une bonne logique décisionnelle pour aider les adaptations.

	$Pattern_{3M,6S}$	$Pattern_{6M,3S}$
Automatique	27,4228	28,1912
Statique	2096,5273	3343,4186

Tableau 5.30 – Performances pour les cas de référence retenus pour le problème de répartition de machines et de politique de réapprovisionnement

Passons maintenant à la définition des éléments nécessaires à une résolution par programmation génétique linéaire.

5.4.3.3 Formulation du problème dans le cadre de la programmation génétique linéaire

Pour nos décisions d'adaptation, nous avons un registre de sortie unique mais à deux dimensions : $T_A = \{d_t\} = \langle d_{1,t}; d_{2,t}\rangle$. D'après leur définition, ces deux variables de décision peuvent prendre leur valeur dans l'intervalle [-1 ; 1] : $T_{CA} = \mathcal{D} = \{[-1;1] \times [-1;1]\}$.

Pour satisfaire la Contrainte 5.11 sur le nombre minimum de machines dans chaque atelier, nous utiliserons le même traitement de correction de l'exemple précédent. L'action $d_{1,t}$ proposée par la logique décisionnelle est ainsi modifiée au besoin comme suit, et ce directement dans notre modèle de simulation :

$$d_{1,t} := \begin{cases} min\{d_{1,t}; 6 - NbM_{A,t}\} & \text{si } d_{1,t} = 1 \\ max\{d_{1,t}; 1 - NbM_{A,t}\} & \text{si } d_{1,t} = -1 \\ d_{1,t} & \text{si } d_{1,t} = 0 \end{cases} \tag{5.13}$$

Les registres mémoire d'entrée sont quant à eux composés des variables d'état explicitées dans la section précédente : $T_V = \{nq_{A,t}; nq_{B,t}; NbM_{A,t}; varComp_t\}$ où la dernière

illustre un exemple d'utilisation de données historiques résumées par un indicateur. Ces variables seront comparées à des seuils pouvant prendre des valeurs dans des intervalles définis par T_{CV}. Les trois premières variables sont entières tandis que la quatrième est réelle. Pour réduire l'espace de recherche, l'intervalle associé à cette dernière est discrétisé de sorte à prendre en compte les valeurs ayant au maximum deux décimales (Section 4.6.1). Nous avons $T_{CV} = \{[2; 50]; [2; 50]; [2; 5]; [1; 5]\}$. Encore une fois, quelques tests seront faits sans inclure la variable d'état $NbM_{A,t}$ associée à la variable de décision $d_{1,t}$.

Nous utilisons des comparateurs d'égalité, de supériorité et d'infériorité. Notons cependant que le test d'égalité n'est pas souvent pertinent pour une variable réelle. De plus, ce n'est en général pas la valeur précise du nombre de produits dans une fille d'attente qui nous intéresse. Ainsi, pour réduire partiellement la redondance et l'espace de recherche, le comparateur d'égalité n'est retenu que pour la variable $NbM_{A,t}$. L'ensemble d'instructions peut alors être décrit comme suit :

- $d_t = \langle 0; 0 \rangle$
- if ($\{nq_{A,t}; nq_{B,t}\}$ {>; <} {[2 ; 50]}) jumpTo *label*
- if ($\{nq_{A,t}; nq_{B,t}\}$ {>; <} {[2 ; 50]}) $d_t = \langle$ {[-1 ; 1]} ; {[-1 ; 1]} $\rangle$
- if ($NbM_{A,t}$ {=; >; <} {[2 ; 5]}) jumpTo *label*
- if ($NbM_{A,t}$ {=; >; <} {[2 ; 5]}) $d_t = \langle$ {[-1 ; 1]} ; {[-1 ; 1]} $\rangle$
- if ($varComp_t$ {>; <} {[1 ; 5]}) jumpTo *label*
- if ($varComp_t$ {>; <} {[1 ; 5]}) $d_t = \langle$ {[-1 ; 1]} ; {[-1 ; 1]} $\rangle$

où la forme que prend l'affectation pure définit « ne rien faire » comme action par défaut pour nos deux variables.

Pour la taille des individus, les valeurs moyenne et maximale varient selon l'activation ou non des deux instructions sur le nombre de machines, ce qui est conditionné par l'utilisation du $Groupe_{QMV}$ ou du $Groupe_{QV}$. Le Tableau 5.31 résume les valeurs minimale et maximale utilisées et valables tout au long de l'évolution ainsi que la moyenne et l'écart-type paramétrant l'initialisation aléatoire selon une distribution normale.

Tailles d'individus	$Groupe_{QV}$	$Groupe_{QMV}$
Minimale	7	7
Maximale	35	70
Moyenne	13	25
Écart-type	5	5

Tableau 5.31 – Paramètres concernant les individus pour le problème de la répartition des machines et de la politique de réapprovisionnement

Les autres paramètres nécessaires au fonctionnement d'un algorithme de programmation génétique linéaire restent les mêmes que pour l'exemple précédent et le Tableau 5.32 qui suit en fait un récapitulatif.

Paramètres	$Config^P$	$Config^M$	$Config^G$
V_{max}	25	50	100
V_{elite}	4	8	15
Immortalité	Non	Non	Non
$NbOp$	25	50	100
P_{max}	100	100	100
P_{stagne}	50	50	50
$V_{tournoi}$	2	2	2

Tableau 5.32 – Configurations utilisées pour le problème de la répartition des machines et de la politique de réapprovisionnement

5.4.4 Résultats

5.4.4.1 Convergence et temps de calcul

Nous avons un total de 6 expérimentations pour chacune des deux *patterns* : 2 configurations pour un individu × 3 configurations pour la PGL. L'évolution de la meilleure solution dans la population selon les différentes expérimentations est présentée dans la Figure 5.21 pour la $Pattern_{3M,6S}$ et dans la 5.22 pour la $Pattern_{6M,3S}$.

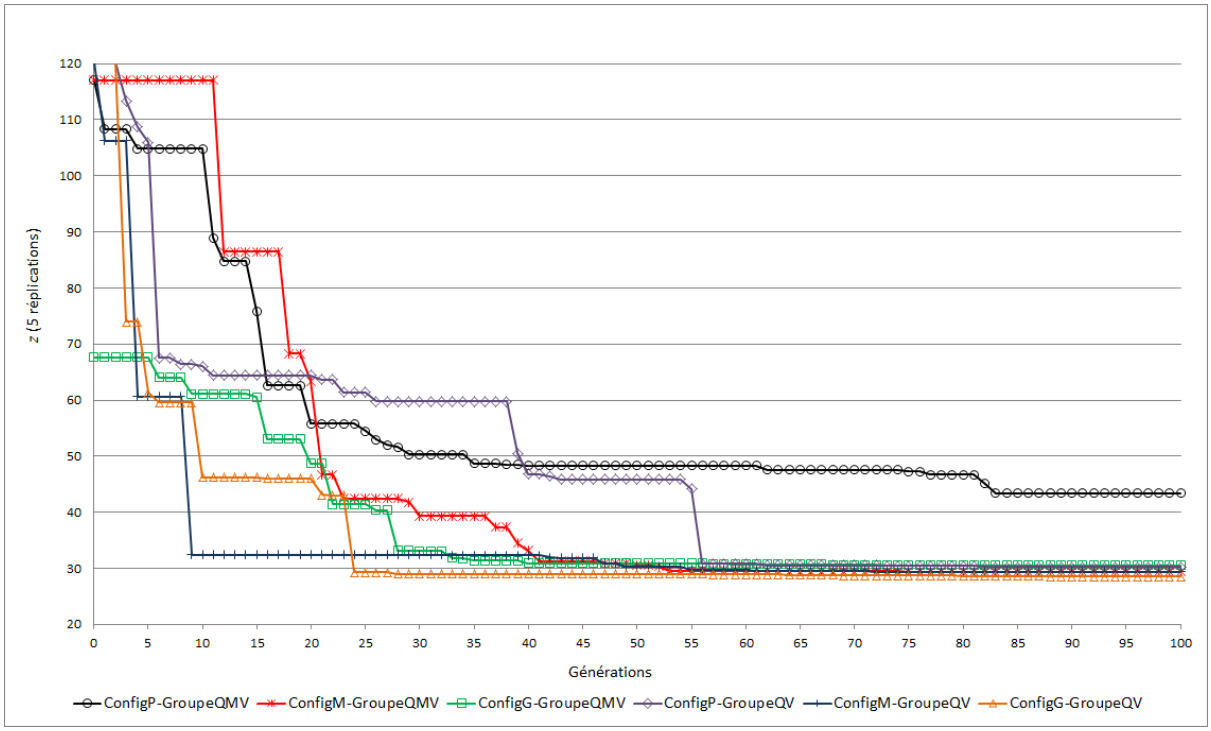

FIGURE 5.21 – Courbes d'apprentissage pour la $Pattern_{3M,6S}$

Nous pouvons constater que l'expérimentation $<Config^G$ - $Groupe_{QMV}>$ (dotée d'une plus grande population et ayant accès à toutes les informations) présente une meilleure

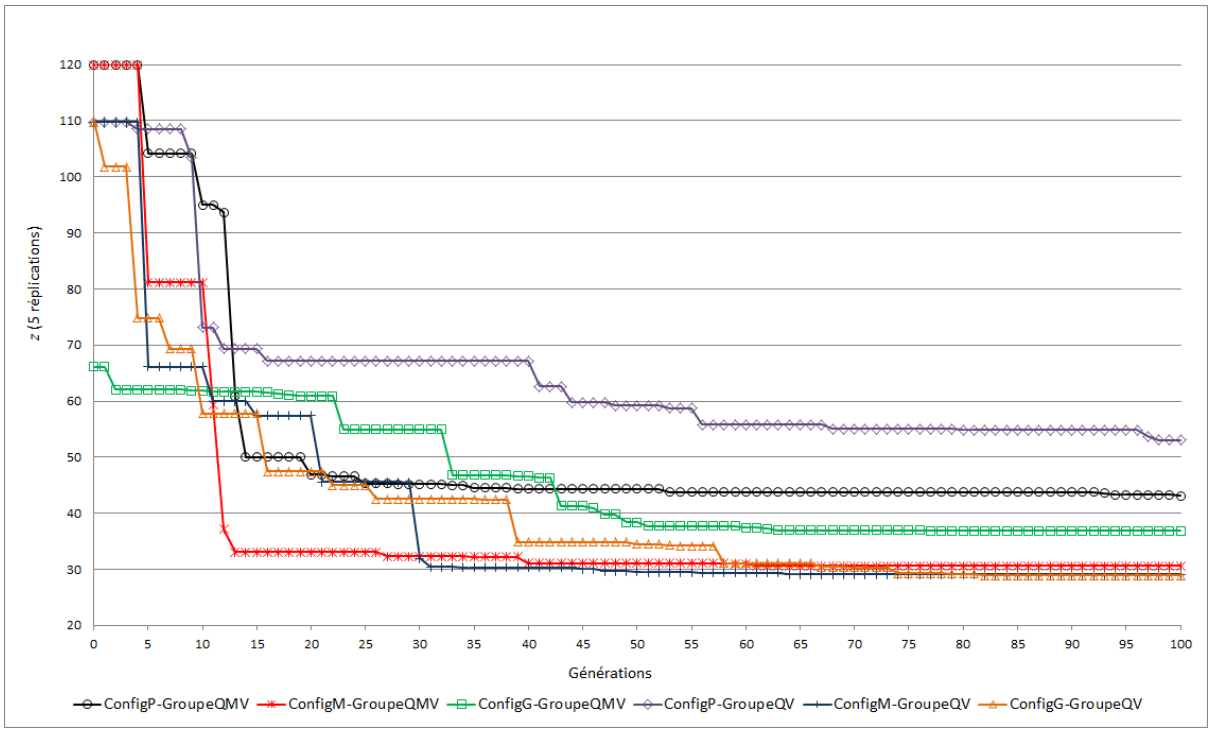

FIGURE 5.22 – Courbes d'apprentissage pour la $Pattern_{6M,3S}$

solution de départ pour les deux scénarios. Les performances finales sont cependant très proches bien qu'une expérimentation soit en retrait pour chaque scénario. Il s'agit de $<Config^P$ - $Groupe_{QMV}>$ pour la $Pattern_{3M,6S}$ et de $<Config^P$ - $Groupe_{QV}>$ pour la $Pattern_{6M,3S}$. Notons également que la convergence de $<Config^P$ - $Groupe_{QV}>$ est plus faible que les autres. Dans tous les cas, nous pouvons encore une fois constater la capacité de notre approche à apprendre.

Le Tableau 5.33 indique le temps de calcul et le nombre d'évaluations faites pour l'obtention des résultats. Les valeurs des colonnes *Temps* sont ici données en jours.

Expérimentations	$Pattern_{3M,6S}$		$Pattern_{6M,3S}$	
	Temps (j)	*NbEval*	*Temps (j)*	*NbEval*
$<Config^P$ - $Groupe_{QV}>$	2,96	4030	2,67	3987
$<Config^M$ - $Groupe_{QV}>$	4,95	6992	6,54	7702
$<Config^G$ - $Groupe_{QV}>$	9,49	14045	9,40	13961
$<Config^P$ - $Groupe_{QMV}>$	3,38	4417	3,13	4136
$<Config^M$ - $Groupe_{QMV}>$	4,40	6453	4,49	6802
$<Config^G$ - $Groupe_{QMV}>$	9,06	13775	9,28	14209

Tableau 5.33 – Temps de calcul et nombre d'évaluations pour les différentes expérimentations avec la PGL pour le problème de répartition de machines et de politique de réapprovisionnement

Il a fallu entre 2,67 jours et 1,36 semaine de calcul hors ligne pour qu'un proces-

sus d'apprentissage soit conclu. D'après le nombre d'évaluations (colonne $NbEval$), ceci correspond à un temps moyen de 61,11 secondes pour l'évaluation de chaque solution. Rappelons que la durée de simulation (2 ans) et le nombre de réplications (5) sont les mêmes que dans l'exemple précédent pour lequel une solution sollicitait en moyenne 25,56 secondes. Ceci confirme donc une constatation de nombreux travaux basés sur la simulation : le temps d'évaluation devient plus important à mesure que le modèle de simulation se complexifie. Toutefois, la prise de décision en ligne à travers la logique décisionnelle obtenue reste largement applicable.

5.4.4.2 Analyse des résultats quantitatifs

Le Tableau 5.34 présente les performances finales après les 30 réplications de la meilleure solution retenue pour chaque combinaison <expérimentation, scénario>. Nous trouverons également la valeur estimée du nombre de déplacements effectués ($NbDep$) et celle du nombre de changements de politique de réapprovisionnement ($NbCP$). Ces valeurs prennent en compte les 2 ans que représente la période d'étude et ne sont donc pas une moyenne par unité de temps.

Expérimentations	$Pattern_{3M,6S}$			$Pattern_{6M,3S}$		
	z	$NbDep$	$NbCP$	z	$NbDep$	$NbCP$
$<Config^{P}$ - $Groupe_{QV}>$	30,9027	56,27	1,00	53,8645	574,57	1,00
$<Config^{M}$ - $Groupe_{QV}>$	30,3141	41,10	1,00	**30,4772**	46,37	1,00
$<Config^{G}$ - $Groupe_{QV}>$	**29,7760**	19,93	1,00	69,9678	13,83	1,00
$<Config^{P}$ - $Groupe_{QMV}>$	45,0530	15,83	14,33	45,1464	351,00	1,00
$<Config^{M}$ - $Groupe_{QMV}>$	31,0370	37,73	9,07	32,1361	92,03	1,00
$<Config^{G}$ - $Groupe_{QMV}>$	32,1344	82,27	6,53	38,0758	223,50	1,00

Tableau 5.34 – Performance estimée et nombre de changements pour les différentes expérimentations avec la PGL pour le problème de répartition de machines et de politique de réapprovisionnement

Notons que pour 75% des expérimentations, un seul changement de politique de réapprovisionnement a été fait. C'est notamment le cas de la meilleure solution pour chaque scénario. Puisqu'au départ c'est la $Politique_{\Delta}$ qui est en place, nous en déduisons que la $Politique_{Seuil}$ est prédominante : une fois celle-ci établie, nous ne revenons plus en arrière. Cette conclusion n'est valable que pour l'exemple spécifique étudié ici, c'est-à-dire pour ce système donné et compte tenu de tous les paramètres nécessaires à la définition des deux politiques.

Pour la $Pattern_{3M,6S}$, le $Groupe_{QV}$ n'utilisant pas d'information sur la répartition courante des machines a obtenu les meilleures performances. Pour la $Pattern_{6M,3S}$, c'est la $Config^{M}$ (de taille intermédiaire). De par ces résultats, confirmés via des tests t de Student, il ne nous est donc pas possible d'identifier une expérimentation qui soit plus

pertinente que les autres, c'est-à-dire, qui produise toujours de meilleurs résultats. Toutefois et dans tous les cas, l'approche semble encore une fois robuste puisque les résultats sont bien plus proches du cas automatique que du statique (revoir Tableau 5.30). Pour nos meilleures solutions obtenues, la différence estimée par rapport au cas automatique est de 8,58% pour la $Pattern_{3M,6S}$ et de 8,11% pour la $Pattern_{6M,3S}$.

5.4.4.3 Analyse de l'arbre de décision généré

Concentrons-nous maintenant sur le contenu des solutions. Pour l'analyser, nous détaillerons la meilleure logique décisionnelle pour la $Pattern_{3M,6S}$ obtenue par l'expérimentation $<Config^G$ - $Groupe_{QV}>$.

Rappelons tout d'abord qu'avec le $Groupe_{QV}$, la variable $NbM_{A,t}$ indiquant la répartition courante des machines n'est jamais analysée. Ainsi, les actions de déplacement doivent toujours être soumises à des vérifications et éventuelles corrections. Celles-ci sont faites directement par notre modèle de simulation et selon l'Équation 5.13. Nous retrouvons dans la Figure 5.23 la version corrigée de cette logique où l'on modifie également les actions correspondant à un changement de politique pour assurer leur validité (selon l'Équation 5.12).

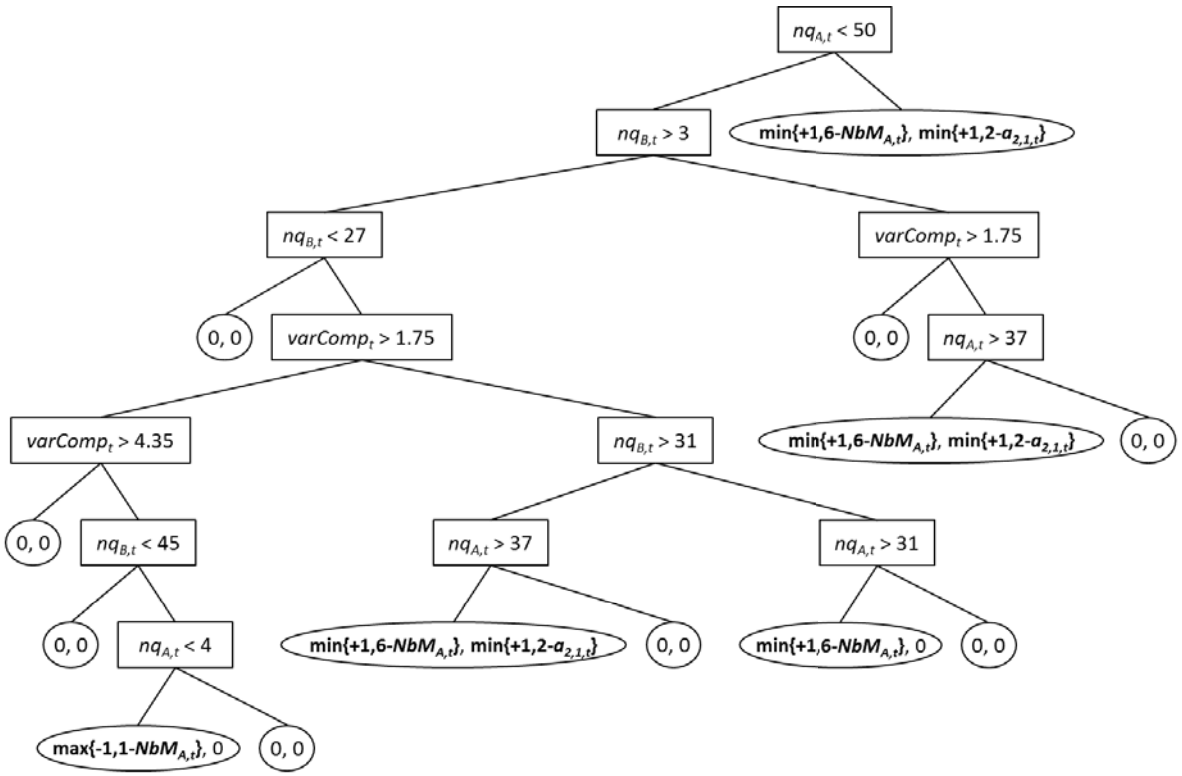

FIGURE 5.23 – Meilleure solution pour le problème ($Pattern_{3M,6S}$)

En remplaçant les variables, les actions et les corrections par leur représentation textuelle, nous aurions mieux montré l'expertise communicable à un décideur. Toutefois, ceci

aurait alourdi nos figures. Rappelons donc que les actions prennent ici la forme d'un vecteur à deux dimensions. La première composante indique le déplacement de l'$Atelier_A$ vers l'$Atelier_B$ avec -1 et le contraire avec +1. La deuxième composante indique quant à elle un changement vers la $Politique_{Seuil}$ avec +1 ou vers la $Politique_{\Delta}$ avec -1. Pour les deux composantes, la valeur 0 correspond à ne rien faire.

Même si seul le code effectif est illustré, le raisonnement est ici plus difficile à comprendre. Toutefois, nous pouvons relever un certain nombre de constatations :

- Seule une circonstance mène à un déplacement de l'$Atelier_A$ vers l'$Atelier_B$, à savoir quand ce dernier est surchargé ($nq_{B,t} \geq 45$) alors que le premier est en surcapacité ($nq_{A,t} < 4$). Ce déplacement est de plus soumis à la condition $1,75 < VarComp_t \leq 4,35$ sur la variabilité dans la demande par composants ;
- Le déplacement inverse est quant à lui envisagé dans beaucoup plus de circonstances. Ce sera le cas à partir du moment où l'$Atelier_A$ a une certaine charge ($nq_{A,t} > 31$) et ce, même si l'$Atelier_B$ est lui aussi chargé. Encore une fois et dans la plupart des cas, des conditions doivent être remplies pour la valeur de $VarComp_t$;
- La variable $nq_{A,t}$ indiquant le nombre de produits dans la file d'attente de l'$Atelier_A$ peut suffire à elle seule au déclenchement d'une décision : lorsque sa valeur est vraiment élevée (≥ 50), la décision proposée est toujours celle de faire un déplacement de l'$Atelier_B$ vers l'$Atelier_A$ et d'établir les réapprovisionnements sur la base de la $Politique_{Seuil}$;
- La décision pour des réapprovisionnements selon la $Politique_{\Delta}$ ne figure pas dans la logique décisionnelle retenue : une fois un contrat signé pour la $Politique_{Seuil}$, on ne revient plus vers la $Politique_{\Delta}$ de départ.

Comme pour l'exemple précédent, différents seuils sont utilisés pour évaluer la charge de nos deux ateliers permettant des réactions pertinentes dans des situations spécifiques. Les déplacements sont déclenchés lorsque cela est nécessaire dans un atelier. De plus, une certaine priorité de l'$Atelier_A$ vis-à-vis de l'$Atelier_B$ a été identifiée. Quant aux politiques de réapprovisionnement, l'absence de décision qui conduirait vers la $Politique_{\Delta}$ peut être interprétée comme la détection par notre algorithme d'une politique qui s'avère toujours plus pertinente (donc meilleure). Notons par ailleurs que la $Politique_{Seuil}$ retenue coïncide avec celle à l'origine du « meilleur cas statique » (Tableau 5.29). Cela appuie la crédibilité de l'expertise construite par la PGL à travers la simulation.

5.4.4.4 Importance relative des décisions vis-à-vis de la performance

Les conclusions que nous venons de dresser nous ont conduits à évaluer l'influence de chaque décision sur la performance finale d'une solution. Pour cela, nous avons testé quatre nouveaux cas de référence, c'est-à-dire sans arbre de décision. Nous avons plus précisément créé, toujours artificiellement, des mélanges entre le cas automatique et les cas statiques. Ainsi, seul un type de variation est détecté automatiquement et sans délai,

faisant l'objet de l'adaptation adéquate. L'autre type de variation n'entraîne quant à lui jamais d'adaptation.

Les deux premiers cas détectent correctement les variations dans le besoin de composants : ils ont une répartition fixe des machines mais changent automatiquement la politique de réapprovisionnement ($Politique_{Auto}$). Les deux autres détectent quant à eux les variations dans les gammes principales : ils conservent donc une politique de réapprovisionnement fixe mais changent automatiquement la répartition des machines ($Repartition_{Auto}$). Les performances obtenues pour 30 réplications de la simulation sont fournies dans le Tableau 5.35 et ce, pour chacune des deux *patterns*.

	$Pattern_{3M,6S}$	$Pattern_{6M,3S}$
$\langle 3;4\rangle - Politique_{Auto}$	2105,7901	3358,2247
$\langle 4;3\rangle - Politique_{Auto}$	2425,1866	4481,7931
$Repartition_{Auto} - Politique_{\Delta}$	105,4395	104,9600
$Repartition_{Auto} - Politique_{Seuil}$	28,8985	28,8027

Tableau 5.35 – Performances pour les cas où l'une des variables reste fixe tandis que l'autre change automatiquement

Nous pouvons ainsi confirmer ce qui a été détecté et correctement intégré dans la logique décisionnelle par notre approche : la meilleure performance dépend plus d'une bonne répartition des machines que du choix d'une politique de réapprovisionnement. Lorsque la politique est changée automatiquement, nous avons un résultat du même ordre de grandeur que ceux des cas statiques (revoir Tableau 5.29). Lorsqu'elle est fixe mais que la répartition change automatiquement, la performance est considérablement améliorée. De plus, nous confirmons également que la $Politique_{Seuil}$ est plus adaptée : lorsqu'on l'utilise tout au long de la simulation avec une répartition adéquate, nous atteignons presque les mêmes performances que celles du cas automatique où la politique change elle aussi automatiquement (différences estimées à 5,38% et 2,17% selon le scénario, revoir Tableau 5.30).

5.5 Bilan

Une discussion peut maintenant être faite sur quelques aspects importants de notre approche, que ce soit de façon générale ou spécifiquement sur la problématique d'adaptation qui nous intéresse. Bien que les réflexions qui suivent découlent de nos expérimentations, certaines les extrapolent.

Tout d'abord, rappelons que les performances des algorithmes de PGL dépendent fortement de l'information comprise dans les ensembles de registres mémoire et d'instructions. Une étude approfondie pour mieux définir ces ensembles devrait donc être conduite avant toute application de la PGL, étude qui reste spécifique à chaque problème. Cela pourrait

par ailleurs faciliter le passage à grande échelle du type d'approche que nous proposons. En effet, une telle étude réalisée à priori et cherchant à ne conserver que les informations les plus pertinentes pourrait atténuer une limitation potentielle de notre approche. Cette limitation serait due au nombre de variables d'état ou de décision devant être gérées par le module d'apprentissage.

Rappelons également que lors de la proposition de notre cadre conceptuel, une première discussion a été faite sur les contraintes auxquelles l'adaptation tout comme le système qu'elle concerne sont soumis. Lors de nos exemples pratiques, nous avons donc cherché à intégrer certaines de ces contraintes. Pour cela, nous avons mis en place un ensemble de règles indiquant, si besoin, comment corriger à posteriori les actions suggérées, et ce directement dans le modèle de simulation. Dans un cas en particulier, nous avons même essayé de gérer les contraintes en définissant une logique décisionnelle composée de sous-parties empêchant leur violation mais cette solution s'est avérée moins performante. Ces traitements ont pu être faits de façon assez simple puisque nous sommes restés sur des cas facilement maîtrisables. Pour le système à flux tiré, les contraintes se résumaient aux nombres minimal et maximal de cartes y circulant. Pour le réseau intra-entreprise, il s'agissait du nombre minimal de machines dans chaque atelier cumulé à l'impossibilité d'en déplacer plus d'une à la fois. Signalons donc que cet aspect relatif aux contraintes est normalement bien plus complexe et doit faire l'objet de recherches supplémentaires. Il peut inclure par exemple des contraintes entre différents types de changement interdépendants. Le but de cette thèse n'était pas de pouvoir les traiter entièrement mais d'en donner un bref aperçu pour en montrer l'importance.

Lors de notre cadre conceptuel, nous avons également évoqué à plusieurs reprises le problème de la nervosité. Nous avons aussi eu l'occasion d'analyser son effet dans un système ConWIP en dehors du cadre de cette thèse. Ce n'est pas notre approche d'apprentissage par simulation qui a été utilisée mais une approche d'optimisation multi-objectif via simulation [Belisário et Pierreval 2014, Belisário *et al.* 2015]. Il n'en demeure pas moins que les résultats confirment un réel intérêt à prendre en compte la nervosité dans la conception de tout système visant l'adaptabilité. Cet aspect peut être traité par différents moyens. Il est possible de l'intégrer directement dans la fonction objectif (G), de façon hiérarchique ou de façon multi-objectif. Il peut également faire partie des informations utilisées lors des prises de décision (I). Enfin, il peut restreindre directement le système via un contrôle antérieur à la validation d'une décision. Ce contrôle peut soit être lié à chaque type de changement spécifique (C via $H_{min,l}$), soit faire partie du mécanisme de changement (M). Bien que ces différentes possibilités puissent être appliquées individuellement, elles ne sont pas pour autant exclusives. Nous avons par ailleurs essayé d'intégrer un certain contrôle de la nervosité dans les applications de ce chapitre, d'une part via le temps minimal $H_{min,l}$ pendant lequel une décision reste valable et d'autre part comme deuxième critère (hiérarchique) de performance. Ces solutions simples se sont avérées plutôt efficaces mais une étude plus approfondie mérite d'être faite, par exemple pour arriver à quantifier les coûts liés à la nervosité.

Nous avons aussi pu effectuer certains tests préliminaires. Tout d'abord, nous avons cherché à apprendre dans des circonstances diverses. Ceci s'est traduit par des expérimentations sur un scénario général combinant plusieurs scénarios particuliers et qui visait à générer de la diversité pour obtenir des solutions plus robustes. Puis, nous avons cherché à prendre en compte les connaissances expertes disponibles. Dans le cas spécifiquement testé, il s'agissait plutôt de connaissances traduisant les contraintes du système. Bien que préliminaires, ces tests nous ont tout de même permis d'identifier des voies de recherche.

Remarquons que nous avons choisi de ne pas faire de réplications pour l'algorithme d'apprentissage, bien que ce dernier soit stochastique. Ce choix est lié au temps de calcul de notre approche qui n'est déjà pas négligeable et qui constitue sa principale limitation. La distribution/parallélisation de notre approche favoriserait la mise en place de ces réplications tout en limitant dans la mesure du possible ce problème de temps de calcul.

Enfin, le passage à grande échelle peut se montrer comme une autre limitation importante du type d'approche que nous proposons. Cette limitation est d'abord due à la quantité d'informations devant être traitée, nous en avons déjà discuté. Elle peut aussi être due à la complexité du système de production à simuler. Par ailleurs et dans notre dernier cas d'étude, nous avons vu un aperçu des différences de temps de calcul engendrées par une légère complexification de notre modèle de simulation. Il est cependant envisageable que cette complexité puisse être partiellement maîtrisée, quand cela se justifie, par ce que l'on appelle la réduction de modèles [Brooks et Tobias 2000, Chwif *et al.* 2006, Thomas *et al.* 2014].

5.6 Conclusion

Dans ce chapitre, nous avons pu concrétiser notre travail par trois cas d'étude qui témoignent de l'applicabilité de notre approche à différents types d'adaptations et/ou de systèmes. Par cette même occasion, nous avons montré la capacité du cadre conceptuel proposé dans le Chapitre 2 à mettre en avant les différents aspects nécessaires à la résolution d'un problème d'adaptation. Il remplit donc son rôle d'aide à la spécification, à l'analyse et à l'exploitation des systèmes de production adaptables.

Nos résultats ont été comparés à des valeurs de référence provenues de la littérature ou créées artificiellement. Ceci a permis d'illustrer l'efficacité et par conséquent l'intérêt de notre approche dans des contextes divers. Celle-ci semble prometteuse car produisant de bons résultats. Toutefois, notre objectif reste surtout d'identifier dans quelle mesure ce type d'approche peut contribuer à l'adaptation des systèmes de production. Il s'agit d'identifier ses apports et ses limites d'après différentes applications.

Tout d'abord, nous avons montré qu'il est possible de conjuguer simulation et apprentissage pour trouver, sans exemple, une logique décisionnelle définissant quand et comment adapter un système de production. Les logiques analysées se sont avérées effi-

caces et surtout cohérentes d'après le raisonnement qu'elles représentent. Notre approche a donc été capable d'extraire des connaissances pertinentes pour la prise de décision dans le cadre de l'adaptation de différents systèmes de production. Ceci montre sa capacité à apprendre.

Quant aux limites de l'approche, elles ne sont à ce jour pas entièrement définies. Nous avons pu mener des discussions qui en ont identifié certaines comme la prise en compte des contraintes ou encore l'espace de recherche qui, au même titre que le temps de calcul, peut être considérable. Cependant, ces limites peuvent être potentiellement franchies car les discussions ont pour la plupart servi à souligner différentes voies de recherche à exploiter.

Conclusion

De nos jours et afin de rester compétitifs, les systèmes de production doivent être en mesure d'adapter au mieux leurs installations et leur organisation pour faire face à un environnement en constante évolution. Ils doivent le faire tout en offrant à leurs gestionnaires la possibilité d'être plus opportunistes.

En effet, les contextes technique, économique et social fluctuent (marché, dysfonctionnements du système, etc.). Par conséquent, nous constatons une recherche d'adaptabilité de plus en plus présente et qui devient même intrinsèque à l'éternelle quête de performance dans le monde industriel. L'état de l'art du Chapitre 1 témoigne du besoin d'outils d'aide à la décision pour supporter ces adaptations qui se doivent d'être rapides et continues. De même, cet état de l'art nous a incités à proposer un cadre conceptuel du domaine et nous a permis de montrer sa généricité puisqu'il recouvre bien la littérature. Détaillé lors du Chapitre 2, il vise à définir les éléments et les processus impliqués lorsque l'on souhaite tirer profit de l'adaptabilité d'un système de production. Il peut ainsi constituer un véritable support aux chercheurs dans ce domaine mais aussi aux industriels souhaitant mieux maîtriser et exploiter la flexibilité offerte par leur système.

L'efficacité de l'exploitation d'un système de production dépend fortement de la maîtrise qu'en ont ses gestionnaires. Or, il s'avère que cette maîtrise est loin d'être suffisante dans le cadre de l'adaptation. Nous nous sommes ainsi concentrés sur la problématique d'extraction de connaissances pouvant aider la prise de décisions dans un tel contexte. Ceci nous a conduits aux méthodes d'apprentissage automatique issues de l'intelligence artificielle.

Nous avons cependant pu constater à travers le Chapitre 3 que l'utilisation des paradigmes de programmation de l'IA ouvrent des possibilités pour construire un système puissant sans pour autant le rendre automatiquement puissant (pour qu'il puisse résoudre des problèmes complexes tout en ne nécessitant qu'un minimum d'efforts de la part de l'utilisateur) [Reddy 1987]. Ainsi, le manque d'expertise et d'exemples pourrait constituer une potentielle barrière. En effet, ces éléments servent à construire des ensembles d'apprentissage souvent nécessaires à ce type d'approche. Leur construction est un problème complexe, surtout lorsque le contexte lui-même complique la tâche de par la taille de l'espace de recherche et le « coût » d'acquisition de données [Huyet 2004]. L'obtention d'ensembles d'apprentissage par simulation a largement été exploitée comme alternative au manque d'expertise, or ceci n'est malheureusement pas toujours possible ou adéquat à un problème donné.

Émerge alors l'idée d'apprendre sans ensemble d'apprentissage, ce qui constitue plus précisément le paradigme d'apprentissage à partir de modèles. La simulation joue ici un rôle crucial car elle est capable de représenter de façon assez réaliste le comportement dynamique et complexe d'un système tout en prenant en compte des données incertaines

et/ou aléatoires. Elle peut ainsi constituer une véritable source de génération de connaissances relatives au fonctionnement d'un système en temps réel.

Cependant, le Chapitre 3 montre que ce potentiel de la simulation reste pour l'instant limité car il a été peu exploité à ce jour. Cette thèse a justement cherché à approfondir l'étude de ce potentiel ainsi qu'à en tester les limites.

Dans ce but, le Chapitre 4 a introduit les principes de notre approche, proposé une formulation du problème puis décrit l'approche en détail. Nous nous sommes penchés sur la technique d'apprentissage de la programmation génétique linéaire. Les particularités concernant cette technique ainsi que celles liées à notre implémentation ont par cette occasion été précisées. Néanmoins, nous avons également insisté sur le fait que d'autres choix auraient pu être faits. En effet, une fois le problème posé, l'idée générale pour le résoudre reste la même et peut donc être facilement adaptée à d'autres techniques et implémentations.

Afin d'évaluer les véritables contributions d'une telle approche ou, autrement dit, dans quelle mesure elle représente une aide intelligente à l'adaptation des systèmes de production, le Chapitre 5 s'est consacré à la concrétiser par des exemples. Ces derniers représentaient des objectifs et/ou contextes différents, ce qui montre que l'approche a vocation à être appliquée à un large spectre de problèmes. Par ailleurs et dans tous les cas étudiés, nous avons pu fournir une aide effective à la prise de décision. Nous avons ainsi vu que cette approche a également vocation à fournir des connaissances pertinentes et réutilisables.

L'extraction de connaissances facilitant l'adaptation des systèmes de production tend à leur fournir des capacités d'auto-adaptation. Les travaux détaillés dans cette thèse ouvrent de nombreuses perspectives de recherche, que ce soit à court, moyen ou long terme.

Une première perspective à très court terme consiste à mieux prendre en compte les différents objectifs au sein d'un système de production en traitant le problème dans une optique multi-objectif [Collette et Siarry 2002, Dimopoulos 2004]. Ceci permettrait notamment d'inclure l'aspect nervosité (fréquence de changements) comme un objectif à part entière. Le logiciel μGP supporte déjà cette fonctionnalité pour laquelle il se base sur une méthode assez reconnue, le NSGA-II. Cependant, des améliorations la concernant sont en cours et seront disponibles dans une prochaine version prévue entre septembre et décembre 2015.

Il existe également à court terme une perspective d'apprentissage dans des circonstances diverses pour obtenir des solutions robustes et non pas adéquates à une situation précise. En effet et bien que ce soit un aspect souvent négligé, il est important de tester la robustesse des outils visant à contrôler l'adaptation d'un système [Cámara *et al.* 2014]. Pour ce faire, une alternative à exploiter serait d'utiliser des scénarios variés et indépendants. Nous entendons par cela obtenir la performance d'une logique décisionnelle non pas sur un scénario unique mais sur plusieurs scénarios en parallèle. Cette voie a déjà

été partiellement exploitée, citons en exemple [Geiger *et al.* 2006]. Ils ont utilisé plusieurs instances pour évaluer une règle de priorité mais ces instances représentaient toutes un même scénario. Le but n'était donc pas d'être robuste à d'autres conditions opératoires mais à des variations au sein d'une même condition, ce que l'on obtient en partie via les réplications de notre modèle de simulation.

Une autre perspective cette fois-ci à moyen terme consiste à améliorer le temps de calcul nécessaire à notre approche. Pour ce faire, nous envisageons l'adaptation de notre méthode pour qu'elle supporte des calculs parallèles et/ou distribués [Tomassini 1999, Maitre 2011, Sudholt 2015]. Ceci suppose une gestion correcte des nombres aléatoires et peut, dans un premier temps, concerner uniquement le module de simulation [Passerat-Palmbach *et al.* 2012, Hill *et al.* 2013]. Une solution envisagée pour l'approche dans sa globalité est d'utiliser OpenMOLE pour la rendre distribuée [Reuillon *et al.* 2013, Reuillon *et al.* 2015]. Dans tous les cas, l'amélioration de cet aspect en favorisera beaucoup d'autres que nous mettons en avant.

Citons tout d'abord l'hybridation avec différentes méthodes permettant par exemple l'analyse de séries chronologiques. Cela nous semble un point intéressant à développer, voire même essentiel. En effet, les décisions d'adaptation dans un système de production ne sont pas prises arbitrairement, y compris celles basées sur une évolution du système surveillée à tout instant. D'une part, nous avons les données historiques. D'autre part, même si des prévisions et/ou plans de production précis et fiables ne sont pas disponibles, il est possible que nous disposions malgré tout d'une certaine visibilité du futur, même la plus minime qui soit. Ceci est même très probable lorsqu'il s'agit de prendre des décisions associées à l'horizon stratégique (long terme). Pour bien analyser ces différents types d'information (historiques et/ou « prévisions »), plusieurs considérations doivent être faites comme le temps de recueil/stockage des données. Puis il faut être capable d'utiliser ces informations au mieux en détectant par exemple les tendances ou les ruptures, de préférence en les traduisant en indicateurs pertinents. Ici, certains travaux en statistique peuvent être d'une grande utilité comme ceux de [Chalmond 1981, Lubes *et al.* 1994, Bertrand et Fleury 2008, Derquenne 2010, Hachicha et Ghorbel 2012, Xanthopoulos et Razzaghia 2014]. Notons que dans cette démarche, un facteur d'oubli peut s'avérer intéressant pour utiliser des longues séries de données tout en attribuant plus d'importance aux informations les plus récentes.

Une hybridation peut d'autre part permettre la mise à jour d'une logique décisionnelle au besoin. Ceci peut notamment être utile lorsque les changements auxquels un système doit faire face sont tels que la logique décisionnelle en vigueur ne s'avère plus adéquate à la situation. Nous pouvons citer quelques travaux dans cette direction (souvent basés sur la théorie statistique des cartes de contrôle). D'un point de vue assez générique et orienté vers les applications réelles, [Pérez-Sánchez *et al.* 2013] ont proposé des réseaux de neurones à topologie dynamique pour accompagner l'évolution des données. Il s'agit d'une méthode d'apprentissage en ligne qui utilise une fonction d'oubli pour que les données récentes soient priorisées, forçant ainsi une adaptation. Intéressés par le contrôle de qualité, [Noyel

et al. 2013] resynchronisent aussi des réseaux de neurones aux données. Dans le cadre du pilotage des systèmes de production, les approches de [Metan *et al.* 2010] et de [Shahzad et Mebarki 2012] comprennent quant à elles une auto-adaptation de l'arbre de décision obtenu déclenchée par un certain nombre de conditions. Il serait ainsi intéressant d'étudier comment intégrer ce type de procédure à notre approche pour éviter un déphasage de la logique décisionnelle obtenue vis-à-vis de la réalité.

Citons également le développement de solutions à structure distribuée/décentralisée. Il s'agit d'une perspective intéressante notamment dans le cadre des systèmes holoniques de plus en plus courants dans la littérature [Leitão 2004, Pach *et al.* 2012, Barbosa 2015, Barbosa *et al.* 2015]. Nous parlons ici de la structure des logiques décisionnelles et non pas de celle de l'approche utilisée pour les obtenir. Il s'agirait donc d'avoir des logiques décisionnelles en parallèle selon le type et/ou niveau de décision. En effet, la structure adoptée est pour l'instant centralisée : une logique décisionnelle unique est responsable de toutes les décisions à prendre simultanément. La structure pourrait cependant être distribuée/décentralisée en définissant une logique décisionnelle spécifique à chaque type de décision à prendre. Dans ce cas, il faudrait étudier comment gérer efficacement l'apprentissage de plusieurs logiques en parallèle. Une solution est de toutes les regrouper en une seule où chaque sous-partie indépendante serait dédiée à un type de décision, au risque d'avoir une faible convergence. L'unicité ne serait donc que symbolique : elle se situerait au niveau de la représentation et non pas de l'utilisation de la logique obtenue qui serait alors décomposée en différentes sous-parties. De même, la structure pourrait être hiérarchique avec des décisions à différents niveaux, local ou global, donc plus ou moins drastiques [Barbosa 2015]. Cela exigerait en contrepartie une connexion entre les différents niveaux pour assurer une cohérence du système. Que ce soit pour le développement d'une structure distribuée/décentralisée et/ou hiérarchique, il faudrait donc étudier comment prendre en compte l'interdépendance qui serait engendrée entre certaines des décisions ou même leur intégralité. Cela nous ramène à la notion de contraintes.

Dans un contexte d'adaptation, les contraintes sont en général nombreuses. Elles peuvent être liées au système en lui-même mais aussi aux types de changement envisagés, que ce soit individuellement ou lorsque ceux-ci sont interdépendants. De par sa complexité, l'intégration/traitement des contraintes constitue une voie de recherche à part entière. Il en est de même pour la notion de « montée en cadence » suivant chaque décision. Cette dernière correspond au temps nécessaire pour qu'une décision puisse voir ses effets apparaître. Elle peut constituer une donnée basée sur la théorie ou une expertise, ou encore un critère à optimiser par l'approche en cours. Étant liée à de nombreuses adaptations pouvant être apportées à un système, elle mériterait une attention particulière. Le couplage entre optimisation et apprentissage pourrait ici par exemple s'avérer intéressant [Dutta 1996].

Soulignons encore le besoin d'investiguer d'une part sur l'adéquation de l'apprentissage à des problèmes combinatoires comme ceux liés à l'agencement et d'autre part sur la prise en compte de connaissances expertes. À ce propos, certaines connaissances peuvent

en effet être disponibles, même à un très faible niveau ou alors incomplètes ou « imparfaites » (car sujettes à un facteur d'incertitude). Quelques travaux les exploitant comme [Ichihashi *et al.* 1996] ont été publiés lorsque l'apprentissage se fait à travers des exemples. Il serait intéressant de pouvoir également en tirer profit pour « assister » le processus d'apprentissage de notre approche. La combinaison entre connaissances expertes et apprises constitue donc une voie de recherche intéressante.

Enfin, une autre perspective intéressante se présente sur le long terme. Bien que nous ayons déjà illustré l'efficacité de notre approche sur différents exemples, la complexité de ces derniers est restée limitée. Ainsi, l'étude de cas plus complexes voire d'un cas réel pourrait valider la pertinence de notre approche à grande échelle ou indiquer plus clairement les difficultés restant à traiter.

Bibliographie

[Aissani *et al.* 2008] Nassima Aissani, Bouziane Beldjilali et Damien Trentesaux. *Use of machine learning for continuous improvement of the real time heterarchical manufacturing control system performances.* International Journal of Industrial and Systems Engineering, vol. 3, no. 4, pages 474–497, 2008.

[Aissani *et al.* 2012] Nassima Aissani, Abdelghani Bekrar, Damien Trentesaux et Bouziane Beldjilali. *Dynamic scheduling for multi-site companies : A decisional approach based on reinforcement multi-agent learning.* Journal of Intelligent Manufacturing, vol. 23, no. 6, pages 2513–2529, 2012.

[Aissani 2010] Nassima Aissani. *Pilotage adaptatif et réactif pour un système de production à flux continu : application à un système de production pétrochimique.* Thèse de doctorat en automatique et informatique, Université de Valenciennes et du Hainaut-Cambrésis (France), Université d'Oran (Algérie), 2010.

[Aldanondo *et al.* 2008] Michel Aldanondo, Elise Vareilles, Meriem Djefel et Claude Baron. *Product and process configuration : A constraint based approach.* In 2008 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM 2008), pages 2010–2014, Singapour, Singapour, 8-11 décembre 2008. IEEE.

[Altiok et Melamed 2007] Tayfur Altiok et Benjamin Melamed. Simulation modeling and analysis with arena. Academic Press, Burlington, Massachusetts, États-Unis, 2007.

[Angelidis *et al.* 2012] Evangelos Angelidis, Daniel Bohn et Oliver Rose. *A simulation-based optimization heuristic using self-organization for complex assembly lines.* In Proceedings of the 2012 Winter Simulation Conference (WSC'12), pages 1231–1240, Berlin, Allemagne, 9-12 décembre 2012. IEEE.

[APICS 1998] The educational Society for Resource Management APICS. Apics dictionary. APICS, The educational Society for Resource Management, Falls Church, Virginie, États-Unis, 9ème édition, 1998.

[Aravind Raj *et al.* 2013] S. Aravind Raj, A. Sudheer, S. Vinodh et G. Anand. *A mathematical model to evaluate the role of agility enablers and criteria in a manufacturing environment.* International Journal of Production Research, vol. 51, no. 19, pages 5971–5984, 2013.

[Araz et Salum 2010] Özlem Uzun Araz et Latif Salum. *A multi-criteria adaptive control scheme based on neural networks and fuzzy inference for DRC manufacturing systems.* International Journal of Production Research, vol. 48, no. 1, pages 251–270, 2010.

[Araz *et al.* 2006] Özlem Uzun Araz, Özgür Eski et Ceyhun Araz. *A Multi-Criteria Decision Making Procedure Based on Neural Networks for Kanban Allocation.* In Jun Wang, Zhang Yi, Jacek M. Zurada, Bao-Liang Lu et Hujun Yin (editeurs), Advances in Neural Networks - ISNN 2006, Third International Symposium on Neural Networks, volume 3973 de *Lecture Notes in Computer Science*, pages 898–905. Springer Berlin Heidelberg, Chengdu, Chine, 28 mai - 1er juin 2006.

[Araz *et al.* 2008] Özlem Uzun Araz, Özgür Eski et Ceyhun Araz. *Determining the parameters of dual-card kanban system : an integrated multicriteria and artificial neural network*

methodology. The International Journal of Advanced Manufacturing Technology, vol. 38, no. 9-10, pages 965–977, 2008.

[Aryanezhad *et al.* 2009] Mir Bahador Aryanezhad, V. Deljoo et Seyed M. Javad Mirzapour Ale Hashem. *Dynamic cell formation and the worker assignment problem : a new model*. The International Journal of Advanced Manufacturing Technology, vol. 41, no. 3-4, pages 329–342, 2009.

[Banzhaf *et al.* 1998] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller et Frank D. Francone. Genetic programming - an introduction : On the automatic evolution of computer programs and its applications. Morgan Kaufmann Publishers ; dpunkt-Verlag, San Francisco, Californie, États-Unis ; Heidelberg, Allemagne, 1998.

[Baqai 2010] Aamer Baqai. *Co-conception des processus d'usinage et des configurations cinématiques d'un système de production reconfigurable*. Thèse de doctorat en génie mécanique, École Nationale Supérieure d'Arts et Métiers, ParisTech, France, 2010.

[Barbosa *et al.* 2011] José Barbosa, Paulo Leitão, Damien Trentesaux et Emmanuel Adam. *Enhancing ADACOR with Biology Insights Towards Reconfigurable Manufacturing Systems*. In 37th Annual Conference on IEEE Industrial Electronics Society (IECON 2011), pages 2746–2751, Melbourne, Victoria, Australie, 7-10 novembre 2011. IEEE.

[Barbosa *et al.* 2012] José Barbosa, Paulo Leitão, Emmanuel Adam et Damien Trentesaux. *Nervousness in Dynamic Self-organized Holonic Multi-agent Systems*. In Javier Bajo Pérez, Miguel A. Sánchez, Philippe Mathieu, Juan M. Corchado Rodríguez, Emmanuel Adam, Alfonso Ortega, María N. Moreno, Elena Navarro, Benjamin Hirsch, Henrique Lopes-Cardoso et Vicente Julián (editeurs), Highlights on Practical Applications of Agents and Multi-Agent Systems, 10th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS'12), volume 156 de *Advances in Intelligent and Soft Computing*, pages 9–17. Springer Berlin Heidelberg, Salamanque, Espagne, 28-30 mars 2012.

[Barbosa *et al.* 2015] José Barbosa, Paulo Leitão, Emmanuel Adam et Damien Trentesaux. *Dynamic self-organization in holonic multi-agent manufacturing systems : The ADACOR evolution*. Computers in Industry, vol. 66, pages 99–111, 2015.

[Barbosa 2015] José Barbosa. *Self-organized and evolvable holonic architecture for manufacturing control*. Thèse de doctorat en automatique et génie informatique, Université de Valenciennes et du Hainaut-Cambrésis, France, 2015.

[Beach *et al.* 2000] R. Beach, A. P. Muhlemann, D. H. R. Price, A. Paterson et J. A. Sharp. *A review of manufacturing flexibility*. European Journal of Operational Research, vol. 122, no. 1, pages 41–57, 2000.

[Beham *et al.* 2008] Andreas Beham, Stephan Winkler, Stefan Wagner et Michael Affenzeller. *A Genetic Programming Approach to Solve Scheduling Problems with Parallel Simulation*. In 2008 IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2008), pages 1–5, Miami, Floride, États-Unis, 14-18 avril 2008. IEEE.

[Bel et Dubois 1985] Gérard Bel et Didier Dubois. *Modélisation et simulation de systèmes automatisés de production*. RAIRO - Automatique, Productique, Informatique Industrielle (APII), vol. 19, pages 3–43, 1985.

[Belisário et Pierreval 2013] Lorena Silva Belisário et Henri Pierreval. *A conceptual framework for analyzing adaptable and reconfigurable manufacturing systems.* In Proceedings of the 5th International Conference on Industrial Engineering and Systems Management (IESM'13), pages 644–650, Rabat, Maroc, 28-30 octobre 2013. IEEE.

[Belisário et Pierreval 2014] Lorena Silva Belisário et Henri Pierreval. *Systèmes à flux tiré réactifs : une approche d'optimisation multiobjectif via simulation.* In 15ème Congrès Annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF 2014), Bordeaux, France, 26-28 février 2014.

[Belisário et Pierreval 2015] Lorena Silva Belisário et Henri Pierreval. *Using genetic programming and simulation to learn how to dynamically adapt the number of cards in reactive pull systems.* Expert Systems with Applications, vol. 42, no. 6, pages 3129–3141, 2015.

[Belisário *et al.* 2009] Lorena Silva Belisário, Lucas Sirimarco Moreira Guedes, Geraldo Robson Mateus et Mauricio Cardoso de Souza. *Heurísticas para o dimensionamento de lotes e sequenciamento de máquinas.* In Anais do XLI Simpósio Brasileiro de Pesquisa Operacional (SBPO 2009), pages 3289–3298, Porto Seguro, Bahia, Brésil, 1-4 septembre 2009.

[Belisário *et al.* 2015] Lorena Silva Belisário, Nesrine Azouz et Henri Pierreval. *Adaptive ConWIP : analyzing the impact of changing the number of cards.* In Proceedings of the 6th International Conference on Industrial Engineering and Systems Management (IESM'15), Séville, Espagne, 21-23 octobre 2015. IEEE.

[Belluz *et al.* 2015] Jany Belluz, Marco Gaudesi, Giovanni Squillero et Alberto Tonda. *Operator Selection using Improved Dynamic Multi-Armed Bandit.* In Proceedings of the 17th Genetic and Evolutionary Computation Conference (GECCO 2015), pages 1311–1317, Madrid, Espagne, 11-15 juillet 2015.

[Ben Ammar 2014] Oussama Ben Ammar. *Planification des réapprovisionnements sous incertitudes pour les systèmes d'assemblage à plusieurs niveaux.* Thèse de doctorat en génie industriel, École Nationale Supérieure des Mines de Saint-Étienne, France, 2014.

[Benkamoun *et al.* 2013] Nadège Benkamoun, Anne-Lise Huyet et Khalid Kouiss. *Reconfigurable Assembly System configuration design approaches for product change.* In Proceedings of the 5th International Conference on Industrial Engineering and Systems Management (IESM'13), pages 651–658, Rabat, Maroc, 28-30 octobre 2013. IEEE.

[Benkamoun *et al.* 2014] Nadège Benkamoun, Waguih ElMaraghy, Anne-Lise Huyet et Khalid Kouiss. *Architecture Framework for Manufacturing System Design.* Procedia CIRP, vol. 17, pages 88–93, 2014. Variety Management in Manufacturing - Proceedings of the 47th CIRP Conference on Manufacturing Systems.

[Benmammar 2009] Badr Benmammar. *Intelligence Artificielle et Systèmes Multi-Agents*, 2009. Cours en ligne disponible sur : https://hal.inria.fr/cel-00660507/document. Mis en ligne en janvier 2012. Dernier accès le 4 février 2015.

[Bensmaine *et al.* 2013] Abderrahmane Bensmaine, Mohammed Dahane et Lyes Benyoucef. *A non-dominated sorting genetic algorithm based approach for optimal machines selection in reconfigurable manufacturing environment.* Computers & Industrial Engineering, vol. 66, no. 3, pages 519–524, 2013. Special Issue : The International Conferences on Computers and Industrial Enginnering (ICC&IEs) - series 41.

[Berchet 2000] Claire Berchet. *Modélisation pour la simulation d'un système d'aide au pilotage industriel.* Thèse de doctorat en génie industriel, Institut National Polytechnique de Grenoble, France, 2000.

[Berrah 2013] Lamia Berrah. *La quantification de la performance dans les entreprises manufacturières : de la déclaration des objectifs à la définition des systèmes d'indicateurs.* Habilitation à diriger des recherches, Université de Savoie, France, 2013.

[Bertrand et Fleury 2008] Pierre R. Bertrand et Gérard Fleury. *Detecting small shift on the mean by finite moving average.* International Journal of Statistics and Management System, vol. 3, no. 1-2, pages 56–73, 2008.

[Blickle et Thiele 1994] Tobias Blickle et Lothar Thiele. *Genetic Programming and Redundancy.* In Jörn Hopf (editeur), Genetic Algorithms within the Framework of Evolutionary Computation : Proceedings of the KI-94 Workshop, pages 33–38, Sarrebruck, Allemagne, août 1994. Max-Planck-Institut für Informatik (MPI-I-94-241).

[Bolis *et al.* 2001] Enzo Bolis, Christian Zerbi, Pierre Collet, Jean Louchet et Evelyne Lutton. *A GP Artificial Ant for image processing : preliminary experiments with EASEA.* In Julian Miller, Marco Tomassini, Pier Luca Lanzi, Conor Ryan, Andrea G. B. Tettamanzi et William B. Langdon (editeurs), Genetic Programming, Proceedings of the Forth European Conference on Genetic Programming (EuroGP2001), volume 2038 de *Lecture Notes in Computer Science*, pages 246–255. Springer Berlin Heidelberg, Lac de Côme, Italie, 18-20 avril 2001.

[Bollon *et al.* 2004] Jean-Marc Bollon, Maria Di Mascolo et Yannick Frein. *Unified Framework for Describing and Comparing the Dynamics of Pull Control Policies.* Annals of Operations Research, vol. 125, no. 1-4, pages 21–45, 2004.

[Bonabeau et Theraulaz 1994] Eric Bonabeau et Guy Theraulaz. Intelligence collective. Hermès, Paris, France, 1994.

[Bonneau et Proth 1985] Fabrice Bonneau et Jean-Marie Proth. *Application de règles de gestion à un systeme de fabrication : classification des objectifs atteints en vue de leur utilisation.* Rapport de recherche RR-0372 RR-0372, SAGEP (Simulation, analyse et gestion des systèmes de production) - INRIA Lorraine, 1985.

[Borangiu *et al.* 2010] Theodor Borangiu, Silviu Răileanu, Damien Trentesaux et Thierry Berger. *Open manufacturing control with agile reconfiguring of resource services.* Journal of Control Engineering and Applied Informatics, vol. 12, no. 4, pages 10–17, 2010.

[Borangiu *et al.* 2015] Theodor Borangiu, Silviu Răileanu, Thierry Berger et Damien Trentesaux. *Switching mode control strategy in manufacturing execution systems.* International Journal of Production Research, vol. 53, no. 7, pages 1950–1963, 2015.

[Bordoloi *et al.* 1999] Sanjeev K. Bordoloi, William W. Cooper et Hirofumi Matsuo. *Flexibility, adaptability, and efficiency in manufacturing systems.* Production and Operations Management, vol. 8, no. 2, pages 133–150, 1999.

[Borisovsky *et al.* 2013] Pavel A. Borisovsky, Xavier Delorme et Alexandre Dolgui. *Genetic algorithm for balancing reconfigurable machining lines.* Computers & Industrial Engineering, vol. 66, no. 3, pages 541–547, 2013. Special Issue : The International Conferences on Computers and Industrial Enginnering (ICC&IEs) - series 41.

[Boser *et al.* 1992] Bernhard E. Boser, Isabelle M. Guyon et Vladimir N. Vapnik. *A training algorithm for optimal margin classifiers.* In David Haussler (editeur), Proceedings of the Fifth Annual Workshop on Computational Learning Theory (COLT'92), pages 144–152, Pittsburgh, Pennsylvanie, États-Unis, 27-29 juillet 1992. ACM Press, New York, États-Unis.

[Bot 2000] Martijn C. J. Bot. *Improving Induction of Linear Classification Trees with Genetic Programming.* In Darrell Whitley (editeur), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2000), pages 403–410, Las Vegas, Nevada, États-Unis, 8-12 juillet 2000. Morgan Kaufmann Publishers.

[Bousbia et Trentesaux 2002] Salah Bousbia et Damien Trentesaux. *Self-organization in distributed manufacturing control : state-of-the-art and future trends.* In 2002 IEEE International Conference on Systems, Man and Cybernetics (SMC), volume 5. IEEE, 6-9 octobre 2002.

[Brameier et Banzhaf 2001] Markus Brameier et Wolfgang Banzhaf. *A comparison of Linear Genetic Programming and Neural Networks in Medical Data Mining.* IEEE Transactions on Evolutionary Computation, vol. 5, no. 1, pages 17–26, 2001.

[Brameier et Banzhaf 2007] Markus Brameier et Wolfgang Banzhaf. Linear genetic programming. Genetic and Evolutionary Computation. Springer US, 1 édition, 2007.

[Brameier 2004] Markus Brameier. *On Linear Genetic Programming.* Thèse de doctorat en informatique, Universität Dortmund, Allemagne, 2004.

[Breiman *et al.* 1984] Leo Breiman, Jerome H. Friedman, Richard A. Olshen et Charles J. Stone. Classification and regression trees. The Wadsworth statistics/probability series. CRC Press, Boca Raton, Floride, États-Unis, 1984.

[Brennan et Norrie 2003] Robert W. Brennan et Douglas H. Norrie. *Metrics for evaluating distributed manufacturing control systems.* Computers in Industry, vol. 51, no. 2, pages 225–235, 2003.

[Brooks et Tobias 2000] Roger J. Brooks et Andrew M. Tobias. *Simplification in the simulation of manufacturing systems.* International Journal of Production Research, vol. 38, no. 5, pages 1009–1027, 2000.

[Burlat et Campagne 2001] Patrick Burlat et Jean-Pierre Campagne. Performance industrielle et gestion des flux. Traité IC2, Série Productique. Hermès Science, Paris, France, 2001.

[Burlat 2015a] Patrick Burlat, 2015. http://www.conwip.com/. Dernier accès le 3 septembre 2015.

[Burlat 2015b] Patrick Burlat, 2015. http://www.wipsim.fr/. Site mis en ligne en mars 2015. Dernier accès le 3 septembre 2015.

[Burlat 2015c] Patrick Burlat. *Méthodes concrètes de conception de système de pilotage de ligne en mode CONWIP.* Techniques de l'Ingénieur, 2015. Modes de pilotage des flux logistiques.

[Cámara *et al.* 2014] Javier Cámara, Rogério de Lemos, Nuno Laranjeiro, Rafael Ventura et Marco Vieira. *Testing the robustness of controllers for self-adaptive systems.* Journal of the Brazilian Computer Society, vol. 20, no. 1, 2014.

[Campbell 1971] Kenneth L. Campbell. *Scheduling is not the problem.* Production and Inventory Management, vol. 12, no. 2, pages 53–60, 1971.

[Can et Heavey 2012] Birkan Can et Cathal Heavey. *A comparison of genetic programming and artificial neural networks in metamodeling of discrete-event simulation models.* Computers & Operations Research, vol. 39, no. 2, pages 424–436, 2012.

[Cardin et Castagna 2009] Olivier Cardin et Pierre Castagna. *Using online simulation in Holonic manufacturing systems.* Engineering Applications of Artificial Intelligence, vol. 22, no. 7, pages 1025–1033, 2009. Distributed Control of Production Systems.

[Cardin *et al.* 2015] Olivier Cardin, Damien Trentesaux, André Thomas, Pierre Castagna, Thierry Berger et Hind Bril El-Haouzi. *Coupling predictive scheduling and reactive control in manufacturing hybrid control architectures : state of the art and future challenges.* Journal of Intelligent Manufacturing, pages 1–15, 2015.

[Caruana et Niculescu-Mizil 2006] Rich Caruana et Alexandru Niculescu-Mizil. *An Empirical Comparison of Supervised Learning Algorithms.* In Proceedings of the 23rd International Conference on Machine learning (ICML'06), pages 161–168, Pittsburgh, Pennsylvanie, États-Unis, 25-29 juin 2006. ACM, New York, États-Unis.

[Caskey 2001] Kevin R. Caskey. *A manufacturing problem solving environment combining evaluation, search, and generalisation methods.* Computers in Industry, vol. 44, no. 2, pages 175–187, 2001.

[Castagna *et al.* 2001] Pierre Castagna, Nasser Mebarki et Roland Gauduel. *Apport de la simulation comme outil d'aide au pilotage des systèmes de production - exemples d'application.* In 3ème Conférence Francophone de Modélisation et Simulation (MOSIM'01), Troyes, France, 25-27 avril 2001.

[Chalfoun *et al.* 2012] Imad Chalfoun, Khalid Kouiss, Nicolas Bouton et Pascal Ray. *Characterization of a Reconfigurable and Agile Manufacturing System (RAMS).* In 14th International Conference on Modern Information Technology in the Innovation Processes of Industrial Enterprises (MITIP 2012), Budapest, Hongrie, 24-26 octobre 2012.

[Chalfoun *et al.* 2013] Imad Chalfoun, Khalid Kouiss, Anne-Lise Huyet, Nicolas Bouton et Pascal Ray. *Proposal for a Generic Model Dedicated to Reconfigurable and Agile Manufacturing Systems (RAMS).* Procedia CIRP, vol. 7, pages 485–490, 2013. Forty Sixth CIRP Conference on Manufacturing Systems 2013.

[Chalmond 1981] Bernard Chalmond. *Détection d'une rupture de moyenne en un point inconnu d'un processus ARMA.* Revue de Statistique Appliquée, vol. 29, no. 1, pages 7–20, 1981.

[Chan et Spedding 2001] K. K. Chan et T. A. Spedding. *On-line optimization of quality in a manufacturing system.* International Journal of Production Research, vol. 39, no. 6, pages 1127–1145, 2001.

[Charles 2010] Aurélie Charles. *Improving the design and management of agile supply chains : feedback and application in the context of humanitarian aid.* Thèse de doctorat en systèmes industriels, Institut National Polytechnique de Toulouse, France, 2010.

[Chebel-Morello *et al.* 2006] Brigitte Chebel-Morello, Pierre Baptiste et Emmanuel Lereno. *Extraction de règles d'ordonnancement : Aide au paramétrage d'un progiciel d'ordonnancement.* Jounal Européen des Systèmes Automatisés, vol. 40, no. 1, pages 11–32, 2006.

[Chen *et al.* 2014] Songlin Chen, Shuli Wu, Xiaojin Zhang et Hongyan Dai. *An evolutionary approach for product line adaptation.* International Journal of Production Research, vol. 52, no. 20, pages 5932–5944, 2014.

[Christo et Cardeira 2007] Camilo Christo et Carlos Cardeira. *Trends in Intelligent Manufacturing Systems*. In 2007 IEEE International Symposium on Industrial Electronics (ISIE 2007), pages 3209–3214, Vigo, Espagne, 4-7 juin 2007. IEEE.

[Christopher et Peck 2004] Martin Christopher et Helen Peck. *Building the resilient supply chain*. International Journal of Logistics Management, vol. 15, no. 2, pages 1–13, 2004.

[Christopher et Towill 2000] Martin Christopher et Denis R. Towill. *Supply chain migration from lean and functional to agile and customized*. Supply Chain Management : an International Journal, vol. 5, no. 4, pages 206–213, 2000.

[Chwif *et al.* 2006] Leonardo Chwif, Ray J. Paul et Marcos Ribeiro Pereira Barretto. *Discrete event simulation model reduction : A causal approach*. Simulation Modelling Practice and Theory, vol. 14, no. 7, pages 930–944, 2006.

[Claver *et al.* 1997] Jean-François Claver, Jacqueline Gélinier et Dominique Pitt. Gestion des flux en entreprise : modélisation et simulation. Hermès, Paris, France, 1997.

[Coffman 1976] Edward Grady Coffman. Computer and job-shop scheduling theory. John Wiley & Sons, New York, États-Unis, 1976.

[Collette et Siarry 2002] Yann Collette et Patrick Siarry. Optimisation multiobjectif : Algorithmes. Eyrolles, Paris, France, 2002.

[Collins et Watson 1993] Norene Collins et Christine M. Watson. *Introduction to Arena*. In G. W. Evans, M. Mollaghasemi, E. C. Russell et W. E. Biles (editeurs), Proceedings of the 25th Winter Simulation Conference (WSC'93), pages 205–212, Los Angeles, Californie, États-Unis, 1993. ACM, New York, États-Unis.

[Colombo *et al.* 2011] Massimo G. Colombo, Keld Laursen, Mats Magnusson et Cristina Rossi-Lamastra. *Organizing Inter- and Intra-Firm Networks : What is the Impact on Innovation Performance ?* Industry & Innovation, vol. 18, no. 6, pages 531–538, 2011.

[Cornuéjols et Miclet 2003] Antoine Cornuéjols et Laurent Miclet. Apprentissage artificiel : Concepts et algorithmes. Eyrolles, 2ème édition, 2002, deuxième tirage 2003. Avec la participation d'Yves Kodratoff, préface de Tom Mitchell.

[Cortes et Vapnik 1995] Corinna Cortes et Vladimir N. Vapnik. *Support-vector networks*. Machine Learning, vol. 20, no. 3, pages 273–297, 1995.

[Dégrés *et al.* 2008] Ludovic Dégrés, Henri Pierreval et Christophe Caux. *Analysing the impact of market variations in the steel industry : a simulation approach*. Production Planning & Control : The Management of Operations, vol. 19, no. 2, pages 150–159, 2008.

[Deif et ElMaraghy 2006] Ahmed M. Deif et Waguih H. ElMaraghy. *A Control Approach to Explore the Dynamics of Capacity Scalability in Reconfigurable Manufacturing Systems*. Journal of Manufacturing Systems, vol. 25, no. 1, pages 12–24, 2006.

[Deif et ElMaraghy 2007] Ahmed M. Deif et Waguih H. ElMaraghy. *Agile MPC system linking manufacturing and market strategies*. Journal of Manufacturing Systems, vol. 26, no. 2, pages 99–107, 2007. Special Issue : Distributed Control of Manufacturing Systems.

[Derquenne 2010] Christian Derquenne. *Une méthode de segmentation pour le traitement de séries temporelles*. In 42ème Journées de Statistique de la Société Française de Statistique, Marseille, France, 24-28 mai 2010.

[Dimopoulos et Zalzala 2001] Christos Dimopoulos et Ali M.S. Zalzala. *Investigating the use of genetic programming for a classic one-machine scheduling problem.* Advances in Engineering Software, vol. 32, no. 6, pages 489–498, 2001.

[Dimopoulos 2004] Christos Dimopoulos. *A Review of Evolutionary Multiobjective Optimization Applications in the Area of Production Research.* In Proceedings of the 2004 IEEE Congress on Evolutionary Computation (CEC'2004), volume 2, pages 1487–1494, Portland, Oregon, États-Unis, 19-23 juin 2004. IEEE.

[Dolgui et Proth 2006] Alexandre Dolgui et Jean-Marie Proth. Systèmes de production modernes. volume 1 : Conception, gestion et optimisation. Hermès, Paris, France, 1ère édition, 2006.

[Doukidis et Angelides 1994] Georgios I. Doukidis et Marios C. Angelides. *A Framework for Integrating Artificial Intelligence and Simulation.* Artificial Intelligence Review, vol. 8, no. 1, pages 55–85, 1994.

[Dreyfus *et al.* 2004] Gérard Dreyfus, Jean-Marc Martinez, Manuel Samuelides, Mirta B. Gordon, Fouad Badran, Sylvie Thiria et Laurent Hérault. Réseaux de neurones : Méthodologies et applications. Eyrolles, Paris, France, 2ème édition, 2004.

[Ducq 2007] Yves Ducq. *Évaluation de la performance d'entreprise par les modèles.* Habilitation à diriger des recherches, Université Bordeaux 1, France, 2007.

[Dudas *et al.* 2011] Catarina Dudas, Marcus Frantzén et Amos H. C. Ng. *A synergy of multi-objective optimization and data mining for the analysis of a flexible flow shop.* Robotics and Computer-Integrated Manufacturing, vol. 27, no. 4, pages 687–695, 2011. Conference papers of Flexible Automation and Intelligent Manufacturing - Intelligent manufacturing and services.

[Dutta 1996] Amitava Dutta. *Integrating AI and optimization for decision support : A survey.* Decision Support Systems, vol. 18, no. 3-4, pages 217–226, 1996.

[Edwards *et al.* 2004] John S. Edwards, Thanos Alifantis, Robert D. Hurrion, John Ladbrook, Stewart Robinson et A. Waller. *Using a simulation model for knowledge elicitation and knowledge management.* Simulation Modelling Practice and Theory, vol. 12, no. 7-8, pages 527–540, 2004. Simulation in Operational Research.

[Egdorf et Roberts 1988] H.W. Egdorf et Douglas J. Roberts. *Discrete event simulation in the artificial intelligence environment.* In Ranjeet J. Uttamsingh (editeur), AI Papers : Proceedings of the Conference on AI and Simulation, volume 20 de *Simulation Series*, pages 64–68, Orlando, Floride, États-Unis, 18-21 avril 1988. Society for Computer Simulation International, San Diego, Californie, États-Unis.

[El-Bouri et Shah 2006] Ahmed El-Bouri et Pramit Shah. *A neural network for dispatching rule selection in a job shop.* The International Journal of Advanced Manufacturing Technology, vol. 31, no. 3-4, pages 342–349, 2006.

[El Haouzi *et al.* 2008] Hind El Haouzi, André Thomas et Jean-François Pétin. *Contribution to reusability and modularity of manufacturing systems simulation models : Application to distributed control simulation within DFT context.* International Journal of Production Economics, vol. 112, no. 1, pages 48–61, 2008. Special Section on Recent Developments in the Design, Control, Planning and Scheduling of Productive Systems.

[ElMaraghy *et al.* 2012a] Hoda ElMaraghy, T. AlGeddawy, A. Azab et Waguih ElMaraghy. *Change in Manufacturing – Research and Industrial Challenges.* In Hoda A. ElMaraghy (editeur), Enabling Manufacturing Competitiveness and Economic Sustainability - Proceedings of the 4th International Conference on Changeable, Agile, Reconfigurable and Virtual production (CARV2011), pages 2–9. Springer Berlin Heidelberg, Montreal, Canada, 2-5 octobre 2012.

[ElMaraghy *et al.* 2012b] Waguih ElMaraghy, Hoda ElMaraghy, Tetsuo Tomiyama et Laszlo Monostori. *Complexity in engineering design and manufacturing.* CIRP Annals - Manufacturing Technology, vol. 61, no. 2, pages 793–814, 2012.

[ElMaraghy 2006] Hoda A. ElMaraghy. *Flexible and reconfigurable manufacturing systems paradigms.* International Journal of Flexible Manufacturing Systems, vol. 17, no. 4, pages 261–276, 2006.

[Eskandari *et al.* 2011] Hamindreza Eskandari, Ehsan Mahmoodi, Hamed Fallah et Christopher D. Geiger. *Performance analysis of comercial simulation-based optimization packages : OptQuest and Witness Optimizer.* In Proceedings of the 2011 Winter Simulation Conference (WSC'11), pages 2358–2368, Phoenix, Arizona, États-Unis, 11-14 décembre 2011. IEEE.

[Essafi 2010] Mohamed Essafi. *Conception et optimisation d'allocation de ressources dans les lignes d'usinage reconfigurables.* Thèse de doctorat en génie industriel, École Nationale Supérieure des Mines de Saint-Étienne, France, 2010.

[Fent 2001] Thomas Fent. Applications of learning classifier systems for simulating learning organizations, volume 10 de *Fortschrittsberichte Simulation, ARGESIM Report.* ARGESIM/ASIM-Verlag, Vienne, Autriche, 2001.

[Fishwick 1991] Paul A. Fishwick. *Knowledge-Based Simulation.* Expert Systems with Applications, vol. 3, no. 3, page 301, 1991. Special Issue : Knowledge-Based Simulation.

[Fonseca et Navaresse 2002] Daniel J. Fonseca et Daniel Navaresse. *Artificial neural networks for job shop simulation.* Advanced Engineering Informatics, vol. 16, no. 4, pages 241–246, 2002.

[Framinan *et al.* 2003] José M. Framinan, Pedro L. González et Rafael Ruiz-Usano. *The CONWIP production control system : Review and research issues.* Production Planning & Control : The Management of Operations, vol. 14, no. 3, pages 255–265, 2003.

[Framinan *et al.* 2006] José M. Framinan, Pedro L. González et Rafael Ruiz-Usano. *Dynamic card controlling in a Conwip system.* International Journal of Production Economics, vol. 99, no. 1-2, pages 102–116, 2006. Control and Management of Productive systems.

[Frawley 1989] William J. Frawley. *The role of simulation in machine learning research.* In Proceedings of the 22nd Annual Symposium on Simulation (ANSS'89), pages 119–127, Tampa, Floride, États-Unis, mars 1989. IEEE Computer Society Press, Los Alamitos, Californie, États-Unis.

[Frécon et Kazar 2009] Louis Frécon et Okba Kazar. Manuel d'intelligence artificielle. METIS Lyon Tech. Presses Polytechniques et Universitaires Romandes (PPUR), 1ère édition, 2009.

[Fu 1994] Michael C. Fu. *Optimization via simulation : A review.* Annals of Operations Research, vol. 53, no. 1, pages 199–247, 1994.

[Fu 2002] Michael C. Fu. *Optimization for simulation : Theory vs. Practice.* INFORMS Journal on Computing, vol. 14, no. 3, pages 192–215, 2002.

[Fukunaga *et al.* 2012] Alex Fukunaga, Hideru Hiruma, Kazuki Komiya et Hitoshi Iba. *Evolving controllers for high-level applications on a service robot : a case study with exhibition visitor flow control.* Genetic Programming and Evolvable Machines, vol. 13, no. 2, pages 239–263, 2012.

[Garavelli 2003] A. Claudio Garavelli. *Flexibility configurations for the supply chain management.* International Journal of Production Economics, vol. 85, no. 2, pages 141–153, 2003. Supply Chain Management.

[Garces-Perez *et al.* 1996] Jaime Garces-Perez, Dale A. Schoenefeld et Roger L. Wainwright. *Solving Facility Layout Problems Using Genetic Programming.* In John R. Koza, David E. Goldberg, David B. Fogel et Rick L. Riolo (editeurs), Proceedings of the First Annual Conference on Genetic Programming (GP'96), pages 182–190, Stanford, Californie, États-Unis, 28-31 juillet 1996. MIT Press, Cambridge, Massachusetts, États-Unis.

[Garcia-Almanza et Tsang 2006] Alma Lilia Garcia-Almanza et Edward P. K. Tsang. *Simplifying Decision Trees Learned by Genetic Programming.* In Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC 2006), pages 2142–2148, Vancouver, Colombie-Britanique, Canada, 16-21 juillet 2006. IEEE.

[Geiger et Uzsoy 2008] Christopher D. Geiger et Reha Uzsoy. *Learning effective dispatching rules for batch processor scheduling.* International Journal of Production Research, vol. 46, no. 6, pages 1431–1454, 2008.

[Geiger *et al.* 2006] Christopher D. Geiger, Reha Uzsoy et Haldun Aytuğ. *Rapid Modeling and Discovery of Priority Dispatching Rules : An Autonomous Learning Approach.* Journal of Scheduling, vol. 9, no. 1, pages 7–34, 2006.

[Geurts *et al.* 2006] Pierre Geurts, Louis Wehenkel et Florence D'Alché-Buc. *OK3 : Méthode d'arbres à sortie noyau pour la prédiction de sorties structurées et l'apprentissage de noyau.* In Conférence francophone sur l'apprentissage automatique (CAP 2006), Trégastel, France, 22-24 mai 2006. Presses Universitaires de Grenoble.

[Giard 2003] Vincent Giard. Gestion de la production et des flux. Collection Gestion - Série : Production et techniques quantitatives appliqueés à la gestion. Economica, Paris, France, 3ème édition, 2003.

[González-R *et al.* 2011] Pedro L. González-R, José M. Framinan et Rafael Ruiz-Usanu. *A response surface methodology for parameter setting in a dynamic Conwip production control system.* International Journal of Manufacturing Technology and Management, vol. 23, no. 1-2, pages 16–33, 2011.

[González-R *et al.* 2012] Pedro L. González-R, José M. Framinan et Henri Pierreval. *Token-based pull production control systems : an introductory overview.* Journal of Intelligent Manufacturing, vol. 23, no. 1, pages 5–22, 2012.

[Gourgand et Kellert 1991] Michel Gourgand et Patric Kellert. *Conception d'un environnement de modélisation des systèmes de production.* In Actes du 3ème Congrès International de Génie Industriel (CIGI), pages 191–203, Tours, France, 20-22 mars 1991.

[Grasso et Taylor III 1984] Edward T. Grasso et Bernard W. Taylor III. *A simulation-based experimental investigation of supply/timing uncertainty in MRP systems.* International Journal of Production Research, vol. 22, no. 3, pages 485–497, 1984.

[Guion *et al.* 2011] Rémy Guion, Hind El Haouzi et André Thomas. *Etude de la pertinence d'un kanban adaptatif avec des contraintes multicritères : Cas d'une cellule de découpe.* In 4èmes Journées Doctorales / Journées Nationales MACS (JD-JN-MACS), Marseille, France, 6-8 juin 2011.

[Gunasekaran 1999] Angappa Gunasekaran. *Agile manufacturing : A framework for research and development.* International Journal of Production Economics, vol. 62, no. 1-2, pages 87–105, 1999.

[Guneri *et al.* 2009] A. F. Guneri, A. Kuzu et A. Taskin Gumus. *Flexible kanbans to enhance volume flexibility in a JIT environment : a simulation based comparison via ANNs.* International Journal of Production Research, vol. 47, no. 24, pages 6807–6819, 2009.

[Gupta et Al-Turki 1997] Surendra M. Gupta et Yousef A. Y. Al-Turki. *An algorithm to dynamically adjust the number of Kanbans in stochastic processing times and variable demand environment.* Production Planning & Control : The Management of Operations, vol. 8, no. 2, pages 133–141, 1997.

[Gustafson et Hsu 2001] Steven M. Gustafson et William H. Hsu. *Layered Learning in Genetic Programming for a Cooperative Robot Soccer Problem.* In Julian Miller, Marco Tomassini, Pier Luca Lanzi, Conor Ryan, Andrea G. B. Tettamanzi et William B. Langdon (editeurs), Genetic Programming, Proceedings of the Forth European Conference on Genetic Programming (EuroGP2001), volume 2038 de *Lecture Notes in Computer Science*, pages 291–301. Springer Berlin Heidelberg, Lac de Côme, Italie, 18-20 avril 2001.

[Habchi et Berchet 2003] Georges Habchi et Claire Berchet. *A model for manufacturing systems simulation with a control dimension.* Simulation Modelling Practice and Theory, vol. 11, no. 1, pages 21–44, 2003. Modelling and Simulation : Analysis, Design and Optimisation of Industrial Systems.

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

[Hachicha et Ghorbel 2012] Wafik Hachicha et Ahmed Ghorbel. *A survey of control-chart pattern-recognition literature (1991-2010) based on a new conceptual classification scheme.* Computers & Industrial Engineering, vol. 63, no. 1, pages 204–222, 2012.

[Hadeli *et al.* 2006] Karuna Hadeli, Paul Valckenaers, Paul Verstraete, Bart Saint Germain et Hendrik Van Brussel. *A Study of System Nervousness in Multi-agent Manufacturing Control System.* In Sven A. Brueckner, Giovanna Di Marzo Serugendo, David Hales et Franco Zambonelli (editeurs), Engineering Self-Organising Systems - Third International Workshop ESOA 2005, volume 3910 de *Lecture Notes in Computer Science*, pages 232–243. Springer Berlin Heidelberg, Utrecht, Pays-Bas, 25 juillet 2006.

[Hagendorf 2009] Olaf Hagendorf. *Simulation Based Parameter and Structure Optimisation of Discrete Event Systems.* Thèse de doctorat en informatique, Liverpool John Moores University, Royaume-Uni, 2009.

[Hammann et Markovitch 1995] John E. Hammann et Nancy A. Markovitch. *Introduction to Arena.* In C. Alexopoulos, K. Kang, W. R. Lilegdon et D. Goldsman (editeurs), Proceedings of the 27th Winter Simulation Conference (WSC'95), pages 519–523, Arlington, Virginie, États-Unis, 3-6 décembre 1995. IEEE, Washington, D.C., États-Unis.

[Hammoud 2011] Djamila Hammoud. *Apprentissage Automatique dans un Agent.* Thèse de doctorat en informatique, Université Mentouri Constantine, Algérie, 2011.

[Heritier-Pingeon 1991] Christine Heritier-Pingeon. *Une aide à la conception de systèmes de production basée sur la simulation et l'analyse de données.* Thèse de doctorat en informatique et automatique appliquées, Institut National des Sciences Appliquées de Lyon, France, 1991.

[Hill *et al.* 2013] David R. C. Hill, Claude Mazel, Jonathan Passerat-Palmbach et Mamadou Kaba Traoré. *Distribution of random streams for simulation practitioners.* Concurrency and Computation : Practice and Experience, vol. 25, no. 10, pages 1427–1442, 2013.

[Hill 1996] David R. C. Hill. Object-oriented analysis and simulation. Addison-Wesley Longman, Boston, Massachusetts, États-Unis, 1996.

[Holt et Perry 2008] Jon Holt et Simon Perry. Sysml for systems engineering. Professional Applications of Computing Series 7. The Institution of Engineering and Technology (IET), Londres, Royaume-Uni, 2008.

[Hopp et Roof 1998] Wallace J. Hopp et Melanie L. Roof. *Setting WIP Levels with Statistical Throughput Control (STC) in CONWIP Production Lines.* International Journal of Production Research, vol. 36, no. 4, pages 867–882, 1998.

[Hunt 1962] Earl Busby Hunt. Concept learning : An information processing problem. John Wiley & Sons, New York, États-Unis, 1ère édition, 1962.

[Hurrion 1997] Robert D. Hurrion. *An example of simulation optimisation using a neural network metamodel : finding the optimum number of kanbans in a manufacturing system.* Journal of Operational Research Society, vol. 48, no. 11, pages 1105–1112, 1997.

[Huyet et Paris 2004] Anne-Lise Huyet et Jean-Luc Paris. *Synergy between evolutionary optimization and induction graphs learning for simulated manufacturing systems.* International Journal of Production Research, vol. 42, no. 20, pages 4295–4313, 2004.

[Huyet 2004] Anne-Lise Huyet. *Extraction de connaissances pertinentes sur le comportement des systèmes de production : une approche conjointe par optimisation évolutionniste via simulation et apprentissage.* Thèse de doctorat en informatique et productique, Université Blaise Pascal - Clermont-Ferrand II, France, 2004.

[Huyet 2006] Anne-Lise Huyet. *Optimization and analysis aid via data-mining for simulated production systems.* European Journal of Operational Research, vol. 173, no. 3, pages 827–838, 2006.

[Ichihashi *et al.* 1996] H. Ichihashi, T. Shirai, K. Nagasaka et T. Miyoshi. *Neuro-fuzzy ID3 : a method of inducing fuzzy decision trees with linear programming for maximizing entropy and an algebraic method for incremental learning.* Fuzzy Sets and Systems, vol. 81, no. 1, pages 157–167, 1996. Fuzzy Optimization.

[Jaegler et Burlat 2012] Anicia Jaegler et Patrick Burlat. *Carbon friendly supply chains : a simulation study of different scenarios.* Production Planning & Control : The Management of Operations, vol. 23, no. 4, pages 269–278, 2012.

[Jain *et al.* 2013] Ajai Jain, P. K. Jain, Felix T. S. Chan et Shailendra Singh. *A review on manufacturing flexibility.* International Journal of Production Research, vol. 51, no. 19, pages 5946–5970, 2013.

[Jimenez *et al.* 2013] Jose-Fernando Jimenez, Abdhelghani Bekrar, Damien Trentesaux, Jairo R. Montoya-Torres et Paulo Leitão. *State of the Art and Future Trends of Optimality and Adaptability Articulated Mechanisms for Manufacturing Control Systems.* In Proceedings of the 2013 IEEE International Conference on Systems, Man, and Cybernetics (SMC 2013), pages 1265–1270, Manchester, Royaume-Uni, 13-16 octobre 2013. IEEE.

[Kandel et Langholz 1992] Abraham Kandel et Gideon Langholz. Hybrid architectures for intelligent systems. CRC Press, Boca Raton, Floride, États-Unis, 1992.

[Kanso et Berruet 2010] Marwa Kanso et Pascal Berruet. *RMS : une évolution des systèmes manufacturiers.* Techniques de l'Ingénieur, 2010. Méthodes de production.

[Kantschik et Banzhaf 2001] Wolfgang Kantschik et Wolfgang Banzhaf. *Linear-Tree GP and Its Comparison with Other GP Structures.* In Julian Miller, Marco Tomassini, Pier Luca Lanzi, Conor Ryan, Andrea G. B. Tettamanzi et William B. Langdon (editeurs), Genetic Programming, Proceedings of the Forth European Conference on Genetic Programming (EuroGP2001), volume 2038 de *Lecture Notes in Computer Science*, pages 302–312. Springer Berlin Heidelberg, Lac de Côme, Italie, 18-20 avril 2001.

[Kapanoglu et Alikalfa 2011] Muzaffer Kapanoglu et Mete Alikalfa. *Learning IF-THEN priority rules for dynamic job shops using genetic algorithms.* Robotics and Computer-Integrated Manufacturing, vol. 27, no. 1, pages 47–55, 2011.

[Katayama et Bennett 1999] Hiroshi Katayama et David Bennett. *Agility, adaptability and leanness : A comparison of concepts and a study of practice.* International Journal of Production Economics, vol. 60-61, pages 43–51, 1999.

[Kayser 1997] Daniel Kayser. La représentation des connaissances. Hermès, Paris, France, 1997.

[Kelton *et al.* 2015] W. David Kelton, Randall P. Sadowski et Nancy B. Zupick. Simulation with arena. McGraw-Hill Education, 6ème édition, 2015.

[Ketfi 2004] Abdelmadjid Ketfi. *Une Approche Générique pour la Reconfiguration Dynamique des Applications à base de Composants Logiciels.* Thèse de doctorat en informatique, Université Joseph Fourier - Grenoble I, France, 2004.

[Kilmer *et al.* 1994] Robert A. Kilmer, Alice E. Smith et L Shuman. *Neural networks as a metamodeling technique for discrete event stochastic simulation.* In Intelligent Engineering Systems Through Artificial Neural Networks, Proceedings of the Artificial Neural Networks in Engineering Conference (ANNIE'94), volume 4, pages 1141–1146, Saint-Louis, Missouri, États-Unis, 13-16 novembre 1994. ASME Press.

[Kilmer *et al.* 1999] Robert A. Kilmer, Alice E. Smith et Larry J. Shuman. *Computing confidence intervals for stochastic simulation using neural networks metamodels.* Computers & Industrial Engineering, vol. 36, no. 2, pages 391–407, 1999.

[Kleijnen et Wan 2007] Jack P. C. Kleijnen et Jie Wan. *Optimization of simulated systems : OptQuest and alternatives.* Simulation Modelling Practice and Theory, vol. 15, no. 3, pages 354–362, 2007.

[Kleijnen 2015] Jack P. C. Kleijnen. Design and analysis of simulation experiments, volume 230 de *nternational Series in Operations Research & Management Science.* Springer International Publishing, 2ème édition, 2015.

[Kleinau et Thonemann 2004] Peer Kleinau et Ulrich W. Thonemann. *Deriving inventory-control policies with genetic programming.* OR Spectrum, vol. 26, no. 4, pages 521–546, 2004.

[Koren et Shpitalni 2010] Yoram Koren et Moshe Shpitalni. *Design of reconfigurable manufacturing systems.* Journal of Manufacturing Systems, vol. 29, no. 4, pages 130–141, 2010.

[Koren *et al.* 1999] Yoram Koren, Uwe Heisel, Francesco Jovane, Toshimichi Moriwaki, Gunter Pritschow, Galip Ulsoy et Hendrik Van Brussel. *Reconfigurable Manufacturing Systems.* CIRP Annals - Manufacturing Technology, vol. 48, no. 2, pages 527–540, 1999.

[Korugan et Gupta 2014] Aybek Korugan et Surendra M. Gupta. *An adaptive CONWIP mechanism for hybrid production systems.* The International Journal of Advanced Manufacturing Technology, vol. 74, no. 5-8, pages 715–727, 2014.

[Koza *et al.* 1999] John R. Koza, Forrest H. Bennett III, David Andre et Martin A. Keane. Genetic programming iii : Darwinian invention and problem solving. Morgan Kaufmann, San Francisco, Californie, États-Unis, 1999.

[Koza *et al.* 2003] John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu et Guido Lanza. Genetic programming iv : Routine human-competitive machine intelligence. Kluwer Academic Publishers, Norwell, Massachusetts, États-Unis, 2003.

[Koza 1992] John R. Koza. Genetic programming : On the programming of computers by means of natural selection. MIT Press, Cambridge, Massachusetts, États-Unis, 1992.

[Koza 1994] John R. Koza. Genetic programming ii : Automatic discovery of reusable programs. MIT Press, Cambridge, Massachusetts, États-Unis, 1994.

[Lage Junior et Godinho Filho 2010] Muris Lage Junior et Moacir Godinho Filho. *Variations of the kanban system : Literature review and classification.* International Journal of Production Economics, vol. 125, no. 1, pages 13–21, 2010.

[Lamotte 2006] Florent Frizon de Lamotte. *Proposition d'une approche haut niveau pour la conception , l'analyse et l'implantation des systèmes reconfigurables.* Thèse de doctorat en automatique et informatique industrielle, Université de Bretagne Sud, France, 2006.

[Langdon et Nordin 2001] William B. Langdon et Peter Nordin. *Evolving Hand-Eye Coordination for a Humanoid Robot with Machine Code Genetic Programming.* In Julian Miller, Marco Tomassini, Pier Luca Lanzi, Conor Ryan, Andrea G. B. Tettamanzi et William B. Langdon (editeurs), Genetic Programming, Proceedings of the Forth European Conference on Genetic Programming (EuroGP2001), volume 2038 de *Lecture Notes in Computer Science*, pages 313–324. Springer Berlin Heidelberg, Lac de Côme, Italie, 18-20 avril 2001.

[Law 2014] Averill M. Law. Simulation modeling and analysis. McGraw-Hill, 5ème édition, 2014.

[Le Moigne 1994] Jean-Louis Le Moigne. La théorie du système général : Théorie de la modélisation. Collection Les Classiques du Réseau Intelligence de la Complexité. Presses universitaires de France, Paris, France, 4ème édition, 1994.

[Le Pallec Marand *et al.* 2013] Léo Le Pallec Marand, Yo Sakata, Daisuke Hirotani, Katsumi Morikawa et Katsuhiko Takahashi. *An Adaptive Kanban and Production Capacity Control Mechanism.* In Christos Emmanouilidis, Marco Taisch et Dimitris Kiritsis (editeurs), Advances in Production Management Systems : Competitive Manufacturing for Innovative Products and Services, volume 397 de *IFIP Advances in Information and Communication Technology*, pages 452–459. Springer Berlin Heidelberg, Rhodes, Grèce, 24-26 septembre 2013.

[LeCroy *et al.* 1996] Kenneth LeCroy, Mansooreh Mollaghasemi et Michael Georgiopoulos. *Application of neural networks and simulation modeling in manufacturing system design.* In IEEE Southcon/96 Conference Record, pages 322–326, Orlando, Floride, États-Unis, 25-27 juin 1996. IEEE.

[Lee *et al.* 2013] Loo Hay Lee, Ek Peng Chew, Peter I. Frazier, Qing-Shan Jia et Chun-Hung Chen. *Advances in simulation optimization and its applications.* IIE Transactions, vol. 45, no. 7, pages 683–684, 2013.

[Lee 2004] Hau L. Lee. *The Triple-A Supply Chain.* Harvard Business Review, vol. 82, no. 10, pages 102–113, 2004.

[Lee 2008] Key K. Lee. *Fuzzy rule generation for adaptive scheduling in a dynamic manufacturing environment.* Applied Soft Computing, vol. 8, no. 4, pages 1295–1304, 2008. Soft Computing for Dynamic Data Mining.

[Leitão *et al.* 2012] Paulo Leitão, José Barbosa et Damien Trentesaux. *Bio-inspired multi-agent systems for reconfigurable manufacturing systems.* Engineering Applications of Artificial Intelligence, vol. 25, no. 5, pages 934–944, 2012.

[Leitão 2004] Paulo Jorge Pinto Leitão. *An Agile and Adaptive Holonic Architecture for Manufacturing Control.* Thèse de doctorat en automatique industrielle, Universidade do Porto, Portugal, 2004.

[Leitão 2008] Paulo Leitão. *Self-organization in manufacturing systems : challenges and opportunities.* In Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW 2008), pages 174–179, Venise, Italie, 20-24 octobre 2008. IEEE.

[Lemaître 1981] Pierre Lemaître. La décision. Collection Formaction. Les Editions d'Organisation, Paris, France, 1981.

[Lereno *et al.* 2001] Emmanuel Lereno, Brigitte Morello et Pierre Baptiste. *Système d'aide au paramétrage d'un logiciel en ordonnancement.* In 3ème Conférence Francophone de Modélisation et Simulation (MOSIM'01), pages 363–369, Troyes, France, 25-27 avril 2001.

[Li *et al.* 2007] Der-Chiang Li, Chih-Sen Wu, Tung-I Tsai et Yao-San Lina. *Using mega-trend-diffusion and artificial samples in small data set learning for early flexible manufacturing system scheduling knowledge.* Computers & Operations Research, vol. 34, no. 4, pages 966–982, 2007.

[Liao 2005] Shu-Hsien Liao. *Expert system methodologies and applications—a decade review from 1995 to 2004.* Expert Systems with Applications, vol. 28, no. 1, pages 93–103, 2005.

[Lin *et al.* 2006] Ching-Torng Lin, Hero Chiu et Yi-Hong Tseng. *Agility evaluation using fuzzy logic.* International Journal of Production Economics, vol. 101, no. 2, pages 353–368, 2006.

[Lin *et al.* 2008] Jung-Yi Lin, Hao-Ren Ke, Been-Chian Chien et Wei-Pang Yang. *Classifier design with feature selection and feature extraction using layered genetic programming.* Expert Systems with Applications, vol. 34, no. 2, pages 1384–1393, 2008.

[Lubes *et al.* 1994] H. Lubes, J. M. Masson, E. Servat, J. E. Paturel, B. Kouame et J. F. Boyer. *Caractérisation de fluctuations dans une série chronologique par application de tests statistiques. Étude bibliographique.* Programme iccare, rapport n°3, Office de la recherche scientifique et technique outre-mer (ORSTOM), Montpellier, France, 1994.

[Macías-Escrivá *et al.* 2013] Frank D. Macías-Escrivá, Rodolfo Haber, Raul del Toro et Vicente Hernandez. *Self-adaptive systems : A survey of current approaches, research challenges and applications.* Expert Systems with Applications, vol. 40, no. 18, pages 7267–7279, 2013.

[Maes 2009] Francis Maes. *Apprentissage dans les Processus de Décision Markoviens pour la Prédiction Structurée. Applications à l'étiquetage de séquences, la transformation d'arbres et l'apprentissage dans les problèmes de recherche combinatoire.* Thèse de doctorat en informatique, Université Pierre et Marie Curie (UPMC), France, 2009.

[Mahboub 2011] Karim Mahboub. *Modélisation des processus émotionnels dans la prise de décision.* Thèse de doctorat en informatique, Université du Havre, France, 2011.

[Maitre 2011] Ogier Maitre. *GPGPU for Evolutionary Algorithms.* Thèse de doctorat en informatique, Université de Strasbourg, France, 2011.

[Manupati *et al.* 2013] V. K. Manupati, Rohit Anand, J. J. Thakkar, Lyes Benyoucef, Fausto P. Garsia et M. K. Tiwari. *Adaptive production control system for a flexible manufacturing cell using support vector machine-based approach.* The International Journal of Advanced Manufacturing Technology, vol. 67, no. 1-4, pages 969–981, 2013.

[Markham *et al.* 1998] Ina S. Markham, Richard G. Mathieu et Barry A. Wray. *A rule induction approach for determining the number of kanbans in a just-in-time production system.* Computers & Industrial Engineering, vol. 34, no. 4, pages 717–727, 1998.

[Markham *et al.* 2000] Ina S. Markham, Richard G. Mathieu et Barry A. Wray. *Kanban setting through artificial intelligence : a comparative study of artificial neural networks and decision trees.* Integrated Manufacturing Systems, vol. 11, no. 4, pages 239–246, 2000.

[McCullen *et al.* 2006] Peter McCullen, Richard Saw, Martin Christopher et Denis Towill. *The F1 Supply Chain : Adapting the Car to the Circuit - the Supply Chain to the Market.* Supply Chain Forum : an International Journal, vol. 7, no. 1, pages 14–23, 2006.

[Meinadier 1998] Jean-Pierre Meinadier. Ingénierie et intégration des systèmes. Hermès, Paris, France, 1998.

[Mesarović *et al.* 1980] Mihajlo D. Mesarović, Dušan Macko et Yasuhiko Takahara. Théorie des systèmes hiérarchiques à niveaux multiples. Economica, Paris, France, 1980.

[Metan et Sabuncuoglu 2005] Gokhan Metan et Ihsan Sabuncuoglu. *A simulation based learning meachanism for scheduling systems with continuous control and update structure.* In M. E. Kuhl, N. M. Steiger, F. B. Armstrong et J. A. Joines (editeurs), Proceedings of the 37th Winter Simulation Conference (WSC'05), pages 2148–2156, Orlando, Floride, États-Unis, 4 décembre 2005. IEEE.

[Metan *et al.* 2010] Gokhan Metan, Ihsan Sabuncuoglu et Henri Pierreval. *Real time selection of scheduling rules and knowledge extraction via dynamically controlled data mining.* International Journal of Production Research, vol. 48, no. 23, pages 6909–6938, 2010.

[Meuleau 1996] Nicolas Meuleau. *Le dilemme entre exploration et exploitation dans l'apprentissage par renforcement : Optimisation adaptative des modèles de décision multi-états.* Thèse de doctorat en informatique, Université de Caen, France, 1996.

[Michalski et Tecuci 1994] Ryszard S. Michalski et Gheorghe Tecuci. Machine learning : A multistrategy approach, volume IV de *Machine Learning.* Morgan Kaufmann Publishers, San Francisco, Californie, États-Unis, 1994.

[Michalski *et al.* 1993] R. S. Michalski, J.G. Carbonell, T. Mitchell et Y. Kodratoff. Apprentissage symbolique : une approche de l'intelligence artificielle. Éditions Cépadues, Toulouse, 1993.

[Miller et Smith 2006] Julian F. Miller et Stephen L. Smith. *Redundancy and Computational Efficiency in Cartesian Genetic Programming.* IEEE Transactions on Evolutionary Computation, vol. 10, no. 2, pages 167–174, 2006.

[Miller et Thomson 2000] Julian F. Miller et Peter Thomson. *Cartesian Genetic Programming.* In Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian Miller, Peter Nordin et Terence C. Fogarty (editeurs), Genetic Programming, Proceedings of the Third European Conference on Genetic Programming (EuroGP2000), volume 1802 de *Lecture Notes in Computer Science*, pages 121–132. Springer Berlin Heidelberg, Édimbourg, Écosse, Royaume-Uni, 15-16 avril 2000.

[Mirdamadi 2009] Samieh Mirdamadi. *Modélisation du processus de pilotage d'un atelier en temps réel à l'aide de la simulation en ligne couplée à l'exécution.* Thèse de doctorat en systèmes industriels, Institut National Polytechnique de Toulouse, France, 2009.

[Mitchell 1997] Tom M. Mitchell. Machine learning. McGraw-Hill, 1997.

[Mittal et Falkenheiner 1990] Sanjay Mittal et Brian Falkenheiner. *Dynamic Constraint Satisfaction Problems.* In Proceedings of the 8th National Conference on Artificial Intelligence, pages 25–32, Boston, Massachusetts, États-Unis, 29 juillet - 3 août 1990. AAAI Press.

[Mittal et Frayman 1989] Sanjay Mittal et Felix Frayman. *Towards a generic model of configuration tasks.* In Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI'89), volume 2, pages 1395–1401, Détroit, Michigan, États-Unis, 20-25 août 1989.

[Mollaghasemi *et al.* 1998] Mansooreh Mollaghasemi, Kenneth LeCroy et Michael Georgiopoulos. *Application of neural networks and simulation modeling in manufacturing system design.* Interface, vol. 28, no. 5, pages 100–114, 1998.

[Monden 1981] Yasuhiro Monden. *Adaptable Kanban System Helps Toyota Maintain Just-In-Time Production.* Industrial Engineering, vol. 13, no. 5, pages 29–46, 1981.

[Monostori *et al.* 2000] László Monostori, Botond Kádár, Zsolt János Viharos, István Mezgár et Péter Stefán. *Combined Use of Simulation and AI/Machine Learning Techniques in Designing Manufacturing Processes and Systems.* Journal for Manufacturing Science and Production, vol. 3, no. 2-4, pages 111–117, 2000.

[Monostori 2003] László Monostori. *AI and machine learning techniques for managing complexity, changes and uncertainties in manufacturing.* Engineering Applications of Artificial Intelligence, vol. 16, no. 4, pages 277–291, 2003. Intelligent Manufacturing.

[Montana 1995] David J. Montana. *Strongly Typed Genetic Programming.* Evolutionary Computation, vol. 3, no. 2, pages 199–230, 1995.

[Montanari 1974] Ugo Montanari. *Networks of constraints : Fundamental Properties and Application to Picture Processing.* Information Sciences, vol. 7, pages 95–132, 1974.

[Morgan et Sonquist 1963] James N. Morgan et John A. Sonquist. *Problems in the Analysis of Survey Data, and a Proposal.* Journal of the American Statistical Association, vol. 58, no. 302, pages 415–434, 1963.

[Mouelhi-Chibani et Pierreval 2010] Wiem Mouelhi-Chibani et Henri Pierreval. *Training a neural network to select dispatching rules in real time.* Computers & Industrial Engineering, vol. 58, no. 2, pages 249–256, 2010. Scheduling in Healthcare and Industrial Systems.

[Mouelhi-Chibani 2009] Wiem Mouelhi-Chibani. *Apprentissage autonome de réseaux de neurones pour le pilotage en temps réel des systèmes de production basé sur l'optimisation via simulation.* Thèse de doctorat en informatique, Université Blaise Pascal - Clermont-Ferrand II, France, 2009.

[Mouelhi et Pierreval 2007] Wiem Mouelhi et Henri Pierreval. *Le concept d'apprentissage autonome et son application au pilotage des systèmes de production.* In 7ème Congrès International de Génie Industriel (CIGI), Trois-Rivières, Québec, Canada, 5-8 juin 2007.

[Mouelhi *et al.* 2007] Wiem Mouelhi, Anne-Lise Huyet et Henri Pierreval. *Combining simulation and artificial neural networks : an overview.* In 6th EUROSIM Congress on Modelling and Simulation, Ljubljana, Slovénie, 9-13 septembre 2007.

[Mrabet *et al.* 2001] Wiem Mrabet, Moncef Tajina, Mohamed Ben Ahmed et Damien Trentesaux. *Système interactif d'aide à la reconfiguration des systèmes de production (SIAR).* In 3ème Conférence Francophone de Modélisation et Simulation (MOSIM'01), pages 515–521, Troyes, France, 25-27 avril 2001.

[Muller et Gaertner 1997] Pierre-Alain Muller et Nathalie Gaertner. Modélisation objet avec uml. Eyrolles, 1ère édition, 1997.

[Murthy 1998] Sreerama K. Murthy. *Automatic Construction of Decision Trees from Data : A Multi-Disciplinary Survey.* Data Mining and Knowledge Discovery, vol. 2, no. 4, pages 345–389, 1998.

[National Research Council 1998] National Research Council. Visionary manufacturing challenges for 2020. National Academies Press, Washington, D.C., États-Unis, 1998.

[Naylor *et al.* 1999] J. Ben Naylor, Mohamed M. Naim et Danny Berry. *Leagility : Integrating the lean and agile manufacturing paradigms in the total supply chain.* International Journal of Production Economics, vol. 62, no. 1-2, pages 107–118, 1999.

[Nilsson 1980] Nils John Nilsson. Principles of artificial intelligence. Tioga Publishing Company, Palo Alto, Californie, États-Unis, 1980.

[Nordin et Banzhaf 1995] Peter Nordin et Wolfgang Banzhaf. *Complexity Compression and Evolution.* In Larry J. Eshelman (editeur), Proceedings of the 6th International Conference on Genetic Algorithms (ICGA-95), pages 310–317, Pittsburgh, Pennsylvanie, États-Unis, 15-19 juillet 1995. Morgan Kaufmann.

[Nordin 1994] Peter Nordin. *A Compiling Genetic Programming System that Directly Manipulates the Machine Code.* In Jr. Kenneth E. Kinnear (editeur), Advances in Genetic Programming, chapitre 14, pages 311–331. MIT Press, Cambridge, Massachusetts, États-Unis, 1994.

[Noyel *et al.* 2013] Mélanie Noyel, Philippe Thomas, Patrick Charpentier, André Thomas et Thomas Brault. *Implantation of an on-line quality process monitoring.* In Proceedings of the 5th International Conference on Industrial Engineering and Systems Management (IESM'13), pages 163–168, Rabat, Maroc, 28-30 octobre 2013. IEEE.

[O'Keefe 1986] Robert O'Keefe. *Simulation and expert systems - A taxonomy and some examples.* Simulation, vol. 46, no. 1, pages 10–16, 1986.

[Oliver 1993] Jonathan J. Oliver. *Decision Graphs – An extension of Decision Trees.* In Proceedings of the Fourth International Workshop on Artificial Intelligence and Statistics, pages 343–350, Fort Lauderdale, Floride, États-Unis, janvier 1993.

[Ören 1994] Tuncer I. Ören. *Artificial intelligence in simulation.* Annals of Operations Research, vol. 53, no. 1, pages 287–319, 1994.

[Osório 1998] Fernando Santos Osório. *INSS : un système hybride neuro-symbolique pour l'apprentissage automatique constructif.* Thèse de doctorat en informatique, Institut National Polytechnique de Grenoble (INPG), France, 1998.

[Otto et Otto 2014] Alena Otto et Christian Otto. *How to design effective priority rules : Example of simple assembly line balancing.* Computers & Industrial Engineering, vol. 69, pages 43–52, 2014.

[Pach *et al.* 2012] Cyrille Pach, Abdelghani Bekrar, Nadine Zbib, Yves Sallez et Damien Trentesaux. *An effective potential field approach to FMS holonic heterarchical control.* Control Engineering Practice, vol. 20, no. 12, pages 1293–1309, 2012.

[Passerat-Palmbach *et al.* 2012] Jonathan Passerat-Palmbach, David R. C. Hill et Claude Mazel. *Pseudo-random streams for distributed and parallel stochastic simulations on GP-GPU.* Journal of Simulation, vol. 6, no. 3, page 141–151, 2012.

[Pegden *et al.* 1995] Claude Dennis Pegden, Robert E. Shannon et Randall P. Sadowski. Introduction to simulation using siman. McGraw-Hill, New York, États-Unis, 2 édition, 1995.

[Pérez-Sánchez *et al.* 2013] Beatriz Pérez-Sánchez, Oscar Fontenla-Romero, Bertha Guijarro-Berdiñas et David Martínez-Rego. *An online learning algorithm for adaptable topologies of neural networks.* Expert Systems with Applications, vol. 40, no. 18, pages 7294–7304, 2013.

[Pickardt *et al.* 2010] Christoph W. Pickardt, Jürgen Branke, Torsten Hildebrandt, Jens Heger et Bernd Scholz-Reiter. *Generating dispatching rules for semiconductor manufacturing to minimize weighted tardiness.* In Proceedings of the 2010 Winter Simulation Conference (WSC'10), pages 2504–2515, Baltimore, Maryland, États-Unis, 5-8 décembre 2010. IEEE.

[Pickardt *et al.* 2013] Christoph W. Pickardt, Torsten Hildebrandt, Jürgen Branke, Jens Heger et Bernd Scholz-Reiter. *Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems.* International Journal of Production Economics, vol. 145, no. 1, pages 66–77, 2013.

[Pierreval et Huntsinger 1992] Henri Pierreval et Ralph C. Huntsinger. *An investigation on neural network capabilities as simulation metamodels.* In Proceedings of the 1992 Summer Computer Simulation Conference, pages 413–417. Society for Computer Simulation, 1992.

[Pierreval et Ralambondrainy 1990] Henri Pierreval et Henri Ralambondrainy. *A Simulation and Learning Technique for Generating Knowledge about Manufacturing Systems Behavior.* Journal of Operational Research Society, vol. 41, no. 6, pages 461–474, 1990.

[Pierreval et Tautou 1997] Henri Pierreval et Laure Tautou. *Using evolutionary algorithms and simulation for the optimization of manufacturing systems.* IIE Transactions, vol. 29, no. 3, pages 181–189, 1997.

[Pierreval *et al.* 2013] Henri Pierreval, Antoine Daures, Thomas Both, S. Szimzack, Pedro L. González-R et José M. Framinan. *A Simulation Optimization Approach for Reactive*

ConWIP Systems. In 8th EUROSIM Congress on Modelling and Simulation (EUROSIM'13), pages 415–420, Cardiff, Pays de Galles, Royaume-Uni, 10-13 septembre 2013. IEEE.

[Pierreval 1992a] Henri Pierreval. *Expert System for Selecting Priority Rules in Flexible Manufacturing Systems.* Expert Systems with Applications, vol. 5, no. 1-2, pages 51–57, 1992.

[Pierreval 1992b] Henri Pierreval. *Training a Neural Network by Simulation for Dispatching Problems.* In Proceedings of the Third International Conference on Computer Integrated Manufacturing, pages 332–336, New York, United States, 20-22 mai 1992. IEEE.

[Pierreval 2006] Henri Pierreval. *Simulation combinée discret/continu : étude du cas d'une fonderie.* In 6ème Conférence Francophone de Modélisation et Simulation (MOSIM'06), pages 824–832, Rabat, Maroc, 3-5 avril 2006.

[Piramuthu *et al.* 1993] Selwyn Piramuthu, Narayan Raman, Michael J. Shaw et Sang Chan Park. *Integration of simulation modeling and inductive learning in an adaptive decision support system.* Decision Support Systems, vol. 9, no. 1, pages 127–142, 1993. Model Management Systems.

[Pitiot *et al.* 2013] Paul Pitiot, Michel Aldanondo, Elise Vareilles, Paul Gaborit, Meriem Djefel et Sabine Carbonnel. *Concurrent product configuration and process planning, towards an approach combining interactivity and optimality.* International Journal of Production Research, vol. 51, no. 2, pages 524–541, 2013.

[Pitiot *et al.* 2014] Paul Pitiot, Michel Aldanondo et Elise Vareilles. *Concurrent product configuration and process planning : Some optimization experimental results.* Computers in Industry, vol. 65, no. 4, pages 610–621, 2014.

[Poli *et al.* 2008] Riccardo Poli, William B. Langdon et Nicholas F. McPhee. A field guide to genetic programming. http://lulu.com, 2008. Avec des contributions de Hohn R. Koza.

[Prakash et Chin 2014] Joshua Prakash et Jeng Feng Chin. *Modified CONWIP systems : a review and classification.* Production Planning & Control : The Management of Operations, vol. 26, no. 4, pages 296–307, 2014.

[Pujo et Brun-Picard 2002] Patrick Pujo et D. Brun-Picard. *Pilotage sans plan prévisionnel, ni ordonnancement préalable.* In Patrick Pujo et Jean-Paul Kieffer (editeurs), Méthodes du pilotage des systèmes de production, Traité IC2, Série Productique, pages 129–162. Hermès, Paris, France, 2002.

[Pujo et Kieffer 2002] Patrick Pujo et Jean-Paul Kieffer. *Concepts fondamentaux du pilotage des systèmes de production.* In Patrick Pujo et Jean-Paul Kieffer (editeurs), Fondements du pilotage des systèmes de production, Traité IC2, Série Productique, pages 25–50. Hermès, Paris, France, 2002.

[Pujo *et al.* 2006] Patrick Pujo, Massimo Pedetti et Norbert Giambiasi. *Formal DEVS modelling and simulation of a flow-shop relocation method without interrupting the production.* Simulation Modelling Practice and Theory, vol. 14, no. 7, pages 817–842, 2006.

[Pujo 2001] Patrick Pujo. *Déménagement d'une ligne de production manufacturière sans arrêt de l'activité de production.* In 3ème Conférence Francophone de Modélisation et Simulation (MOSIM'01), pages 493–499, Troyes, France, 25-27 avril 2001.

[Quinlan 1979] J. Ross Quinlan. *Discovering rules by induction from large collections of examples.* In D. Michie (editeur), Expert Systems in the Micro-Electronic Age, pages 168–201. Edinburgh University Press, Édimbourg, Écosse, Royaume-Uni, 1979.

[Quinlan 1986] J. Ross Quinlan. *Induction of Decision Trees.* Machine Learning, vol. 1, no. 1, pages 81–106, 1986.

[Quinlan 1987] J. Ross Quinlan. *Simplifying Decision Trees.* International Journal of Man-Machine Studies, vol. 27, no. 3, pages 221–234, 1987.

[Rakotomalala 1997] Ricco Rakotomalala. *Graphes d'induction.* Thèse de doctorat en informatique, Université Claude Bernard - Lyon I, France, 1997.

[Rakotomalala 2005] Ricco Rakotomalala. *Arbres de Décision.* Revue Modulad, vol. 33, pages 163–187, 2005.

[Reaidy 2003] Jihad Reaidy. *Etude et mise en œuvre d'une Architecture d'Agents en Réseau dans les Systèmes Dynamiques Situés : Pilotage des Systèmes de Production Complexes.* Thèse de doctorat en génie industriel, Université de Savoie, France, 2003.

[Reddy 1987] Ramana Reddy. *Epistemology of knowledge based simulation.* Simulation, vol. 48, no. 4, pages 162–166, 1987.

[Rees *et al.* 1987] Loren Paul Rees, Patrick R. Philipoom, Bernard W. Taylor III et Philip Y. Huang. *Dynamically Adjusting the Number of Kanbans in a Just-in-Time Production System Using Estimated Values of Leadtime.* IIE Transactions, vol. 19, no. 2, pages 199–207, 1987.

[Ren *et al.* 2013] Shenggang Ren, Longwei Wang, Wei Yang et Feng Wei. *The effect of external network competence and intrafirm networks on a firm's innovation performance : The moderating influence of relational governance.* Innovation : Management, Policy & Practice, vol. 15, no. 1, pages 17–34, 2013.

[Reuillon *et al.* 2013] Romain Reuillon, Mathieu Leclaire et Sebastien Rey-Coyrehourcq. *OpenMOLE, a workflow engine specifically tailored for the distributed exploration of simulation models.* Future Generation Computer Systems, vol. 29, no. 8, pages 1981–1990, 2013. Including Special sections : Advanced Cloud Monitoring Systems & The fourth IEEE International Conference on e-Science 2011 - e-Science Applications and Tools & Cluster, Grid, and Cloud Computing.

[Reuillon *et al.* 2015] Romain Reuillon, Mathieu Leclaire et Jonathan Passerat-Palmbach. *Model Exploration Using OpenMOLE - a workflow engine for large scale distributed design of experiments and parameter tuning.* In 2015 IEEE International Conference on High Performance Computing & Simulation (HPCS 2015), Amsterdam, Pays-Bas, 20-24 juin 2015.

[Rockwell Software 2004] Rockwell Software. *OptQuest for Arena : User's Guide*, 2004.

[Rockwell Software 2007a] Rockwell Software. *Arena User's Guide*, 2007.

[Rockwell Software 2007b] Rockwell Software. *Arena Variables Guide*, 2007.

[Saenz de Ugarte 2009] Benoît Saenz de Ugarte. *Aide à la prise de décision en temps réel dans un contexte de production adaptative.* Thèse de doctorat en génie industriel, École Polytechnique de Montréal, Canada, 2009.

[Sahin *et al.* 2012] Seda Sahin, Mehmet R. Tolun et Reza Hassanpour. *Hybrid expert systems : A survey of current approaches and applications.* Expert Systems with Applications, vol. 39, no. 4, pages 4609–4617, 2012.

[Salerno 2009] Mario Sergio Salerno. *Reconfigurable organisation to cope with unpredictable goals.* International Journal of Production Economics, vol. 122, no. 1, pages 419–428, 2009. Transport Logistics and Physical Distribution - Interlocking of Information Systems for International Supply and Demand Chains Management - ICPR19.

[Sanchez *et al.* 2011] Ernesto Sanchez, Massimiliano Schillaci et Giovanni Squillero. Evolutionary optimization : the μgp toolkit. Springer US, 1ère édition, 2011.

[Sanchez *et al.* 2012] Ernesto Sanchez, Giovanni Squillero et Alberto Tonda. Industrial applications of evolutionary algorithms, volume 34 de *Intelligent Systems Reference Library.* Springer Berlin Heidelberg, 1ère édition, 2012.

[Santini et Tettamanzi 2001] Massimo Santini et Andrea Tettamanzi. *Genetic Programming for Financial Time Series Prediction.* In Julian Miller, Marco Tomassini, Pier Luca Lanzi, Conor Ryan, Andrea G. B. Tettamanzi et William B. Langdon (editeurs), Genetic Programming, Proceedings of the Forth European Conference on Genetic Programming (EuroGP2001), volume 2038 de *Lecture Notes in Computer Science*, pages 361–370, Lac de Côme, Italie, 18-20 avril 2001. Springer Berlin Heidelberg.

[Seebacher et Winkler 2013] Gottfried Seebacher et Herwig Winkler. *A Citation Analysis of the Research on Manufacturing and Supply Chain Flexibility.* International Journal of Production Research, vol. 51, no. 11, pages 3415–3427, 2013.

[Shahabudeen et Sivakumar 2008] P. Shahabudeen et G. D. Sivakumar. *Algorithm for the design of single-stage adaptive kanban system.* Computers & Industrial Engineering, vol. 54, no. 4, pages 800–820, 2008.

[Shahzad et Mebarki 2012] Atif Shahzad et Nasser Mebarki. *Data mining based job dispatching using hybrid simulation-optimization approach for shop scheduling problem.* Engineering Applications of Artificial Intelligence, vol. 25, no. 6, pages 1173–1181, 2012.

[Shahzad 2011] Muhammad Atif Shahzad. *Une Approche Hybride de Simulation-Optimisation Basée sur la Fouille de Données pour les Problèmes d'ordonnancement.* Thèse de doctorat en automatique et génie informatique, Université de Nantes, France, 2011.

[Sharabi et Sipper 2006] Shai Sharabi et Moshe Sipper. *GP-sumo : Using genetic programming to evolve sumobots.* Genetic Programming and Evolvable Machines, vol. 7, no. 3, pages 211–230, 2006.

[Shavlik et Towell 1989] Jude W. Shavlik et Geoffrey G. Towell. *An approach to combining explanation based and neural learning algorithms.* Connection Science, vol. 1, no. 3, pages 231–253, 1989.

[Sherehiy *et al.* 2007] Bohdana Sherehiy, Waldemar Karwowski et John K. Layer. *A review of enterprise agility : Concepts, frameworks, and attributes.* International Journal of Industrial Ergonomics, vol. 37, no. 5, pages 445–460, 2007.

[Shin *et al.* 2009] Moonsoo Shin, Jungtae Mun et Mooyoung Jung. *Self-evolution framework of manufacturing systems based on fractal organization.* Computers & Industrial Engineering, vol. 56, no. 3, pages 1029–1039, 2009. Intelligent Manufacturing and Logistics.

[Shin *et al.* 2012] Moonsoo Shin, Kwangyeol Ryu et Mooyoung Jung. *Reinforcement learning approach to goal-regulation in a self-evolutionary manufacturing system.* Expert Systems with Applications, vol. 39, no. 10, pages 8736–8743, 2012.

[Shiue et Guh 2006] Yeou-Ren Shiue et Ruey Shiang Guh. *Learning-based multi-pass adaptive scheduling for a dynamic manufacturing cell environment.* Robotics and Computer-Integrated Manufacturing, vol. 22, no. 3, pages 203–216, 2006.

[Sivakumar et Shahabudeen 2008] G. D. Sivakumar et P. Shahabudeen. *Design of multi-stage adaptive kanban system.* The International Journal of Advanced Manufacturing Technology, vol. 38, no. 3-4, pages 321–336, 2008.

[Sotskov *et al.* 2013] Yuri N. Sotskov, Omid Gholami et Frank Werner. *Solving a job-shop scheduling problem by an adaptive algorithm based on learning.* In 7th IFAC Conference on Manufacturing Modelling, Management and Control, volume 1, pages 1352–1357, Saint-Pétersbourg, Russie, 19-21 juillet 2013.

[Spearman *et al.* 1990] Mark L. Spearman, David L. Woodruff et Wallace J. Hopp. *CONWIP : a pull alternative to kanban.* International Journal of Production Research, vol. 28, no. 5, pages 879–894, 1990.

[Squillero 2005] Giovanni Squillero. *MicroGP - An Evolutionary Assembly Program Generator.* Genetic Programming and Evolvable Machines, vol. 6, no. 3, pages 247–263, 2005.

[Sudholt 2015] Dirk Sudholt. *Parallel Evolutionary Algorithms.* In Janusz Kacprzyk et Witold Pedrycz (editeurs), Springer Handbook of Computational Intelligence, chapitre 46, pages 929–959. Springer Berlin Heidelberg, 2015.

[Sun *et al.* 2009] Koun-Tem Sun, Yi-Chun Lin, Cheng-Yen Wu et Yueh-Min Huang. *An application of the genetic programming technique to strategy development.* Expert Systems with Applications, vol. 36, no. 3, Part 1, pages 5157–5161, 2009.

[Surbier *et al.* 2013] Laurène Surbier, Gülgün Alpan et Eric Blanco. *A comparative study on production ramp-up : state-of-the-art and new challenges.* Production Planning & Control : The Management of Operations, vol. 25, no. 15, pages 1264–1286, 2013.

[Sutton et Barto 1998] Richard S. Sutton et Andrew G. Barto. Reinforcement learning : An introduction. MIT Press, Cambridge, Massachusetts, États-Unis, 1998.

[Swafford *et al.* 2006] Patricia M. Swafford, Soumen Ghosh et Nagesh Murthy. *The antecedents of supply chain agility of a firm : Scale development and model testing.* Journal of Operations Management, vol. 24, no. 2, pages 170–188, 2006.

[Tahon 2003] Christian Tahon. Évaluation des performances des systèmes de production. Traité IC2, Série Productique. Hermès, Paris, France, 2003.

[Takahashi et Nakamura 1999a] Katsuhiko Takahashi et Nobuto Nakamura. *Applying a Neural Network to the Adaptive Control for JIT Production Systems.* In Proceedings of the 1999 IEEE International Conference on Control Applications, volume 2, pages 1648–1653, Kohala Coast, Hawaii, 22-27 août 1999. IEEE.

[Takahashi et Nakamura 1999b] Katsuhiko Takahashi et Nobuto Nakamura. *Reacting JIT ordering systems to the unstable changes in demand.* International Journal of Production Research, vol. 37, no. 10, pages 2293–2313, 1999.

[Takahashi et Nakamura 2002] Katsuhiko Takahashi et Nobuto Nakamura. *Decentralized reactive Kanban system.* European Journal of Operational Research, vol. 139, no. 2, pages 262–276, 2002. EURO XVI : O.R. for Innovation and Quality of Life.

[Takahashi *et al.* 2004] Katsuhiko Takahashi, Katsumi Morikawa et Nobuto Nakamura. *Reactive JIT ordering system for changes in the mean and variance of demand.* International Journal of Production Economics, vol. 92, no. 2, pages 181–196, 2004.

[Takahashi *et al.* 2010] Katsuhiko Takahashi, Katsumi Morikawa, Daisuke Hirotani et Takeshi Yoshikawa. *Adaptive Kanban control systems for two-stage production lines.* International Journal of Manufacturing Technology and Management, vol. 20, no. 1-4, pages 75–93, 2010.

[Talibi *et al.* 2013] Zakia Talibi, Hind Bril El Haouzi et André Thomas. *The relevance study of adaptive kanban in a multicriteria constraints context using data-driven simulation method.* In Proceedings of the 5th International Conference on Industrial Engineering and Systems Management (IESM'13), pages 169–175, Rabat, Maroc, 28-30 octobre 2013. IEEE.

[Tamani *et al.* 2008] Karim Tamani, Reda Boukezzoula et Georges Habchi. *Pilotage flou distribué et supervisé pour la régulation des flux de production.* In Rencontres Francophones sur la Logique Floue et ses Applications (LFA 2008), pages 94–101, Lens, France, 16-17 octobre 2008.

[Tamani *et al.* 2009] Karim Tamani, Reda Boukezzoula et Georges Habchi. *Intelligent distributed and supervised flow control methodology for production systems.* Engineering Applications of Artificial Intelligence, vol. 22, no. 7, pages 1104–1116, 2009. Distributed Control of Production Systems.

[Tardif et Maaseidvaag 2001] Valerie Tardif et Lars Maaseidvaag. *An adaptive approach to controlling kanban systems.* European Journal of Operational Research, vol. 132, no. 2, pages 411–424, 2001. Data Envelopment Analysis.

[Tekin et Sabuncuoglu 2004] Eylem Tekin et Ihsan Sabuncuoglu. *Simulation optimization : A comprehensive review on theory and applications.* IIE Transactions, vol. 36, no. 11, pages 1067–1081, 2004.

[Teller et Veloso 1996] Astro Teller et Manuela Veloso. *PADO : A New Learning Architecture for Object Recognition.* In K. Ikeuchi et Manuela Veloso (editeurs), Symbolic Visual Learning, pages 81–116. Oxford University Press, New York, États-Unis, 1996.

[Teredesai *et al.* 2001] Ankur Teredesai, J. Park et Venugopal Govindaraju. *Active Handwritten Character Recognition Using Genetic Programming.* In Julian Miller, Marco Tomassini, Pier Luca Lanzi, Conor Ryan, Andrea G. B. Tettamanzi et William B. Langdon (editeurs), Genetic Programming, Proceedings of the Forth European Conference on Genetic Programming (EuroGP2001), volume 2038 de *Lecture Notes in Computer Science*, pages 371–379, Lac de Côme, Italie, 18-20 avril 2001. Springer Berlin Heidelberg.

[Terrenoire 1970] Michel Terrenoire. *Un modèle mathématique de processus d'interrogation : les pseudoquestionnaires.* Thèse de doctorat en sciences mathématiques, Université Joseph-Fourier - Grenoble I, France, 1970.

[Thagard et Millgram 1995] Paul Thagard et Elijah Millgram. *Inference to the best plan : A coherence theory of decision.* In Ashwin Ram et David B. Leake (editeurs), Goal-Driven

Learning, chapitre 19, pages 439–454. MIT Press, Cambridge, Massachusetts, États-Unis, 1995.

[Tharumarajah 2003] Ambalavanar Tharumarajah. *A self-organising view of manufacturing enterprises.* Computers in Industry, vol. 51, no. 2, pages 185–196, 2003. Virtual Enterprise Management.

[Thiel 1996] D. Thiel. *Analysis of the behaviour of production systems using continuous simulation.* International Journal of Production Research, vol. 34, no. 11, pages 3227–3251, 1996.

[Thomas *et al.* 2011] Philippe Thomas, André Thomas et Marie-Christine Suhner. *A neural network for the reduction of a product-driven system emulation model.* Production Planning & Control : The Management of Operations, vol. 22, no. 8, pages 767–781, 2011.

[Thomas *et al.* 2014] Philippe Thomas, Marie-Christine Suhner et André Thomas. *Réduction et génération automatique de modèle Arena© en exploitant des arbres de régression : cas d'une scierie.* In 10ème Conférence Francophone de Modélisation, Optimisation et Simulation (MOSIM'14), Nancy, France, 5-7 novembre 2014.

[Tomassini 1999] Marco Tomassini. *Parallel and Distributed Evolutionary Algorithms : A Review.* In Kaisa Miettinen, Marko M. Mäkelä, Pekka Neittaanmäki et Jacques Périaux (editeurs), Evolutionary Algorithms in Engineering and Computer Science, pages 113–133. John Wiley & Sons, 1999.

[Topçu 2014] Okan Topçu. *Adaptive decision making in agent-based simulation.* Simulation : Transactions of the Society for Modeling and Simulation International, vol. 90, no. 7, pages 815–832, 2014.

[Trentesaux et Tahon 1995] Damien Trentesaux et Christian Tahon. *DPACS : a self-adaptative production activity control structure.* In 1995 INRIA/IEEE Symposium on Emerging Technologies and Factory Automation (ETFA'95), volume 2, pages 543–551, Paris, France, 10-13 octobre 1995. IEEE.

[Trentesaux *et al.* 1998] Damien Trentesaux, Neville Moray et Christian Tahon. *Integration of the human operator into responsive discrete production management systems.* European Journal of Operational Research, vol. 109, no. 2, pages 342–361, 1998.

[Trentesaux *et al.* 2013] Damien Trentesaux, Cyrille Pach, Abdelghani Bekrar, Yves Sallez, Thierry Berger, Thérèse Bonte, Paulo Leitão et José Barbosa. *Benchmarking flexible job-shop scheduling and control systems.* Control Engineering Practice, vol. 21, no. 9, pages 1204–1225, 2013.

[Trentesaux 1996] Damien Trentesaux. *Conception d'un système de pilotage distribué, supervisé et multicritère pour les systèmes automatisés de production.* Thèse de doctorat en automatique et productique, Institut National Polytechnique de Grenoble, France, 1996.

[Trentesaux 2002] Damien Trentesaux. *Pilotage hétérarchique des systèmes de production.* Habilitation à diriger des recherches, Université de Valenciennes et du Hainaut-Cambrésis, France, 2002.

[Tsai et Ghoshal 1998] Wenpin Tsai et Sumantra Ghoshal. *Social Capital and Value Creation : The Role of Intrafirm Networks.* The Academy of Management Journal, vol. 41, no. 4, pages 464–476, 1998.

[Tsang 1993] Edward Tsang. Foundations of constraints satisfaction. Academic Press Limited, Londres, Royaume-Uni, 1993.

[Tunstel et Jamshidi 1996] Edward Tunstel et Mo Jamshidi. *On Genetic Programming of Fuzzy Rule-Based Systems for Intelligent Control*. International Journal of Intelligent Automation & Soft Computing, vol. 2, no. 3, pages 273–284, 1996.

[Usman *et al.* 2013] Zahid Usman, R. I. M. Young, Nitishal Chungoora, Claire Palmer, Keith Case et J. A. Harding. *Towards a formal manufacturing reference ontology*. International Journal of Production Research, vol. 51, no. 22, pages 6553–6572, 2013.

[Utgoff 1989] Paul E. Utgoff. *Incremental Induction of Decision Trees*. Machine Learning, vol. 4, no. 2, pages 161–186, 1989.

[Van Brussel *et al.* 1999] Hendrik Van Brussel, Luc Bongaerts, Jo Wyns, Paul Valckenaers et Tony Van Ginderachter. *A Conceptual Framework for Holonic Manufacturing : Identification of Manufacturing Holons*. Journal of Manufacturing Systems, vol. 18, no. 1, pages 35–52, 1999.

[Vareilles *et al.* 2008] Elise Vareilles, Michel Aldanondo, Paul Gaborit, Guillaume Auriol, Samuel Rochet et Claude Baron. *A Prospective Proposal for Coupling Product and Project Configuration with Constraints*. In 2nd International Conference on Software, Knowledge, Information Management and Applications (SKIMA 2008), pages 148–153, Kathmandou, Népal, 18-21 mars 2008.

[Von Neumann et Morgenstern 1944] John Von Neumann et Oskar Morgenstern. The theory of games and economic behaviour. Princeton University Press, 1944.

[Wang et Shi 2013] Long-Fei Wang et Le-Yuan Shi. *Simulation Optimization : A Review on Theory and Applications*. Acta Automatica Sinica, vol. 39, no. 11, pages 1957–1968, 2013.

[Wang *et al.* 2005] K.-J. Wang, J. C. Chen et Y.-S. Lin. *A hybrid knowledge discovery model using decision tree and neural network for selecting dispatching rules of a semiconductor final testing factory*. Production Planning & Control : The Management of Operations, vol. 16, no. 7, pages 665–680, 2005.

[Whybark et Williams 1976] D. Clay Whybark et J. Gregg Williams. *Material requirements planning under uncertainty*. Decision Sciences, vol. 7, no. 4, pages 595–606, 1976.

[Wiendahl et Breithaupt 1999] Hans-Peter Wiendahl et Jan-Wilhelm Breithaupt. *Modelling and controlling the dynamics of production systems*. Production Planning & Control : The Management of Operations, vol. 10, no. 4, pages 389–401, 1999.

[Wilson 1998] William H. Wilson. *The AI Dictionary*, 1998. Disponible sur : http://www.cse.unsw.edu.au/∼billw/aidict.html. Dernière mise à jour le 25 juin 2012. Dernier accès le 15 février 2015.

[Wray *et al.* 1997] Barry A. Wray, Terry R. Rakes et Loren Paul Rees. *Neural network identification of critical factors in a dynamic just-in-time kanban environment*. Journal of Intelligent Manufacturing, vol. 8, no. 2, pages 83–96, 1997.

[Wu et Wysk 1989] Szu-Yung David Wu et Richard A. Wysk. *An application of discrete-event simulation to on-line control and scheduling in flexible manufacturing*. International Journal of Production Research, vol. 27, no. 9, pages 1603–1623, 1989.

[Xanthopoulos et Razzaghia 2014] Petros Xanthopoulos et Talayeh Razzaghia. *A Weighted Support Vector Machine Method for Control Chart Pattern Recognition.* Computers & Industrial Engineering, vol. 70, pages 134–149, 2014.

[Yeom et Park 2012] Kiwon Yeom et Ji Hyung Park. *Morphological approach for autonomous and adaptive systems based on self-reconfigurable modular agents.* Future Generation Computer Systems, vol. 28, no. 3, pages 533–543, 2012.

[Yildirim *et al.* 2006] Mehmet Bayram Yildirim, Tarik Cakar, Ufuk Doguc et Jose Ceciliano Meza. *Machine number, priority rule, and due date determination in flexible manufacturing systems using artificial neural networks.* Computers & Industrial Engineering, vol. 50, no. 1-2, pages 185–194, 2006.

[Zambrano Rey *et al.* 2013] Gabriel Zambrano Rey, Marco Carvalho et Damien Trentesaux. *Cooperation Models Between Humans and Artificial Self-Organizing Systems : motivations, issues and perspectives.* In 6th International Symposium on Resilient Control Systems (ISRCS 2013), pages 156–161, San Francisco, Californie, États-Unis, 13-15 août 2013. IEEE.

[Zambrano Rey *et al.* 2014a] Gabriel Zambrano Rey, Abdelghani Bekrar, Vittaldas Prabhu et Damien Trentesaux. *Coupling a genetic algorithm with the distributed arrival-time control for the JIT dynamic scheduling of flexible job-shops.* International Journal of Production Research, vol. 52, no. 12, pages 3688–3709, 2014.

[Zambrano Rey *et al.* 2014b] Gabriel Zambrano Rey, Thérèse Bonte, Vittaldas Prabhu et Damien Trentesaux. *Reducing myopic behavior in FMS control : A semi-heterarchical simulation-optimization approach.* Simulation Modelling Practice and Theory, vol. 46, pages 53–75, 2014. Simulation-Optimization of Complex Systems : Methods and Applications.

[Zambrano Rey 2014] Gabriel Zambrano Rey. *Reducing Myopic Behavior in FMS Control : A Semi-Heterarchical Simulation-Optimization Approach.* Thèse de doctorat en automatique et génie informatique, Université de Valenciennes et du Hainaut-Cambrésis, France, 2014.

[Zeigler 1976] Bernard P. Zeigler. Theory of modeling and simulation. Wiley Interscience, New York, États-Unis, 1 édition, 1976.

[Zhang 2011] David Z. Zhang. *Towards theory building in agile manufacturing strategies - Case studies of an agility taxonomy.* International Journal of Production Economics, vol. 131, no. 1, pages 303–312, 2011. Innsbruck 2008.

[Zhao et De Souza 2001] Zhenying Zhao et Robert De Souza. *Fuzzy rule learning during simulation of manufacturing resources.* Fuzzy Sets and Systems, vol. 122, no. 3, pages 469–485, 2001.

[Zighed et Rakotomalala 2000] Djamel Abdelkader Zighed et Ricco Rakotomalala. Graphes d'induction : apprentissage et data mining. Hermès, Paris, France, 2000.

Annexe A

Usage conjoint de l'IA et la simulation

Les caractéristiques et potentiels de la simulation décrits précédemment ont fait ressortir pour certains chercheurs des similarités avec l'intelligence artificielle. Pour [Ören 1994], les deux sont basées sur la recherche et seules diffèrent la nature de l'espace de recherche et la façon dont la recherche est effectuée. Pour [Doukidis et Angelides 1994], cela ressort du point de vue méthodologique puisque les deux essaient de modéliser la réalité en représentant des objets et leurs interactions afin de résoudre des problèmes et prendre des décisions. Ils explorent les avantages potentiels de faire profiter à l'un les concepts de l'autre et comment leur intégration peut être mutuellement utile.

En ce qui concerne les systèmes experts plus précisément, [Fishwick 1991] signale leur lien avec la simulation par le fait que les connaissances expertes reflètent souvent des phénomènes dépendant du temps, même si cette connaissance est généralement sous une forme « brute » telle que le langage naturel ou les règles. Il remarque lui aussi que les domaines peuvent être utiles l'un à l'autre et ce en comblant les lacunes entre des approches qualitatives et quantitatives.

[O'Keefe 1986] met en évidence trois points sur la simulation et les systèmes experts :

- Leur orientation commune pour fournir un modèle informatique qui aide à la décision ;
- Le fait qu'ils essaient souvent de modéliser l'incertitude inhérente à un système ;
- Leur croissant besoin d'être à la fois directement exploitables par l'utilisateur mais aussi d'accéder à d'autres logiciels et d'interagir avec (par exemple, des bases de données).

De plus, tous deux sont basés sur une représentation modulaire du système menée par un mécanisme d'inférence qui fonctionne avec n'importe quel nombre de modules. Ces derniers peuvent, en théorie, être présentés dans un ordre quelconque et sont entièrement indépendants les uns des autres. Toutefois, cela est souvent difficile à réaliser en pratique. Selon l'auteur, les similarités se trouvent aussi au niveau de la représentation entre les événements conditionnels d'une simulation à événements discrets et les règles de production dominantes en intelligence artificielle.

Ces fortes similarités méthodologiques ont inspiré des chercheurs à proposer des taxonomies permettant de mieux exploiter le couplage de ces domaines.

A.1 Taxonomies de couplage

[O'Keefe 1986] a proposé une taxonomie reconnue pour la combinaison de systèmes experts et de modèles de simulation. Elle compte quatre grandes catégories dont trois donnent lieu à deux variantes de conception (Figure A.1). Chaque variante (ou architecture) sera par la suite désignée par un numéro.

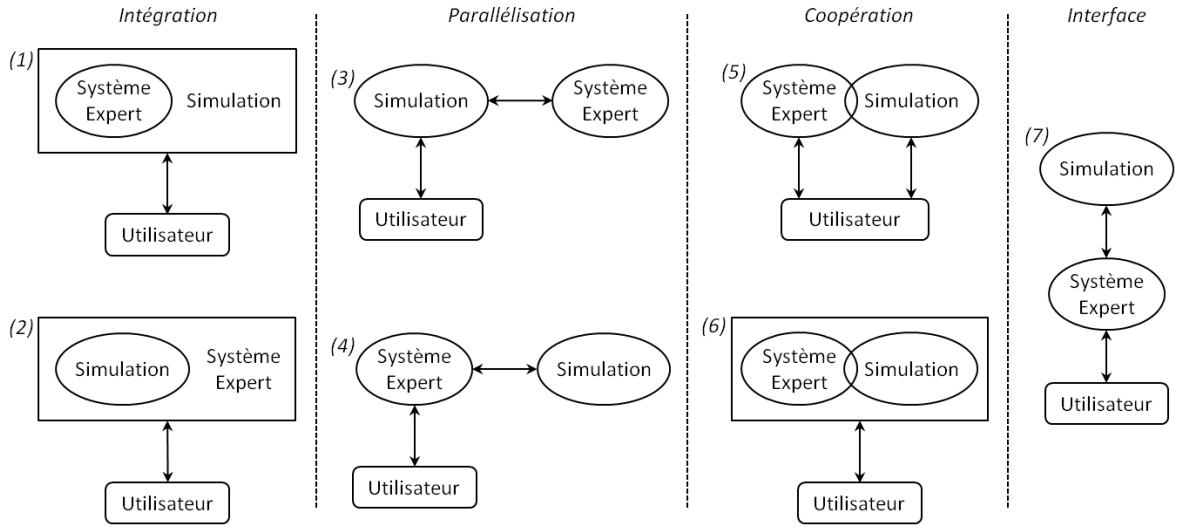

FIGURE A.1 – Taxonomie d'[O'Keefe 1986]

Selon l'auteur, la façon la plus évidente de procéder est d'intégrer l'un dans l'autre :

(1) Dans un sens, la simulation utilise des connaissances (en format de règles) qui sont codées mais gardées dans une base de connaissances (donc un système expert) plutôt que directement dans le code de la simulation ;

(2) Dans l'autre, la simulation peut être nécessaire à un système expert pour obtenir quelques résultats ou surtout pour mettre à jour la valeur des variables dépendantes du temps.

La deuxième possibilité est d'avoir le système expert et le modèle de simulation en parallèle. Ils peuvent ainsi être conçus, développés et mis en œuvre en tant que des logiciels séparés. Cependant, ces derniers peuvent interagir :

(3) Dans un sens, lorsqu'un système expert existe déjà pour une partie des décisions à prendre dans un système complexe à modéliser, il est utilisé en tant que mécanisme de contrôle pour la simulation : la simulation l'interroge, donc elle y a accès plutôt que de ré-encoder ses propres règles ;

(4) Dans l'autre, le système expert exécute et utilise des résultats de simulation pour être testé (plutôt que dans un environnement réel). Il peut être validé s'il contrôle ou répond de façon adéquate au modèle de simulation et que ce dernier est lui-même valide. La simulation peut ensuite être remplacée par le système réel [Monostori *et al.*

2000]. Des applications peuvent concerner par exemple le contrôle de processus en temps réel.

[O'Keefe 1986] signale qu'avec l'architecture (4) le temps de développement est réduit et les tests peuvent être plus compréhensibles. Selon [Doukidis et Angelides 1994], les architectures (1) et (3) peuvent aussi conduire à un développement plus rapide.

L'utilisateur de (1) à (4) a accès direct à un seul outil, mais les deux peuvent aussi être utilisés ensemble pour effectuer une tâche. La troisième possibilité consiste alors à leur faire partager des données et coopérer (5). Ce sera le cas par exemple lorsqu'un système expert est un outil qui aide l'utilisateur avec un modèle fini (expérimentation et analyse, évitement d'une mauvaise utilisation) ou dans son développement (conseils dans le choix du langage de simulation). Pour cela, le système expert contient des informations sur le modèle et le domaine [Monostori *et al.* 2000]. Les deux outils peuvent aussi être entourés par un autre logiciel utilisé directement par l'utilisateur, ceci constituant une autre variante (6).

Enfin, le modèle de simulation et le système expert peuvent être liés de façon à ce que l'on ait une interface intelligente (7). Le système expert, placé entre l'utilisateur et la simulation, génère les instructions ou code nécessaires pour utiliser cette dernière (en suivant un dialogue avec l'utilisateur), interprète ses résultats et les explique. Cela permet de générer et d'exploiter indirectement et plus facilement un modèle de simulation.

La taxonomie d'[O'Keefe 1986] est reprise dans [Monostori *et al.* 2000] pour le couplage de la simulation non pas avec un système expert mais avec différentes techniques intelligentes (des systèmes basés sur la connaissance), ce qui inclut, entre autres, les réseaux de neurones et les systèmes flous. Les auteurs s'intéressent plus précisément à la combinaison de l'apprentissage et de la simulation. Ils signalent d'une part que l'apprentissage peut améliorer la performance de toutes ces différentes architectures et d'autre part que la simulation peut servir à générer des exemples pour l'apprentissage. À ce propos et d'après [Frawley 1989], l'apprentissage est basé sur l'expérience et, malgré les différentes classes d'algorithme, il dépend crucialement de la simulation pour lui fournir cette expérience.

[Ören 1994], quant à lui, propose une taxonomie basée sur deux critères, à savoir le rôle de l'intelligence artificielle et l'accès direct ou non de l'utilisateur à chacun des outils. Il se concentre cependant sur les apports de l'IA à la simulation, et donc sur la simulation basée sur ou assistée par l'IA. Il s'agit des situations où l'IA génère le comportement ou tout simplement lorsqu'elle aide au développement/gestion/expérimentation de la simulation. Il y a cependant de fortes ressemblances avec la proposition d'[O'Keefe 1986] et, comme [Monostori *et al.* 2000], l'auteur signale que l'apprentissage peut être appliqué aux différents types d'architecture, ce qui leur donne la capacité de mettre à jour leur base de connaissances. Il identifie l'apprentissage par simulation comme étant un domaine de recherche prometteur où la simulation est utilisée en tant que technique de génération de connaissances empiriques à base de modèles pour l'apprentissage. Pour cela, des modules peuvent concevoir, exécuter et surveiller les expérimentations de simulation alors

que d'autres vont interpréter les résultats pour en apprendre davantage sur la structure ou le comportement du système étudié.

A.2 Bilan

Sur le plan théorique, quelques travaux proposent des schémas et des explications pour différentes façons possibles de connecter des modules d'intelligence artificielle à la simulation. La différence entre certaines variantes reste parfois très subtile et l'identification de l'architecture adoptée peut varier suivant comment le problème est présenté. Cependant, les voies principales sont identifiées.

Nous pouvons également souligner que, même si des taxonomies sont proposées et que l'apprentissage au sein des différentes architectures est évoqué, les auteurs ne considèrent pas le problème d'élaboration du système expert. Or, c'est un processus parsemé de difficultés. Ainsi, le rôle de la simulation spécifiquement dans le cadre de l'apprentissage mériterait plus d'attention.

Sur le plan pratique et du point de vue de ce qui a été fait dans le domaine des systèmes de production, la Section 3.4.2 nous permet de constater que les différents couplages ne sont pas forcément exploités.

Par rapport à l'approche proposée dans cette thèse, la logique décisionnelle n'est pas directement codée au sein de l'environnement de simulation et doit donc doit provenir d'une base de données externe. Il faut faire en sorte que la simulation puisse y accéder et l'interpréter. Cela correspondrait au cas (3) de la typologie de couplage entre un « système expert » et la simulation (revoir Figure A.1) où cette dernière a accès à une base de règles existante. Notons cependant que si nous considérons non pas cette interaction précise mais notre approche comme un tout, nous sommes plutôt dans le cas (4) : un « système expert » est utilisé pour contrôler la simulation dont les résultats servent à évaluer ledit système et ainsi à guider sa mise à jour. Ce n'est donc pas tout à fait la simulation qui se sert des règles mais c'est celui qui les a conçu qui se sert de la simulation pour s'assurer qu'elles soient adéquates [Doukidis et Angelides 1994, Ören 1994].

Annexe B

Compléments relatifs à l'usage d'Arena

Arena est un produit édité par Rockwell Automation, et plus précisément par sa branche Rockwell Software. C'est un logiciel de simulation utilisant Siman comme langage intégré. La création de modèles avec Arena est surtout graphique et visuelle et il n'est pas nécessaire d'écrire de lignes de codes, bien que ceci reste possible. Parmi les particularités d'Arena, on trouve sa capacité à communiquer avec certains programmes externes. Au-delà des fonctions de base pour lire et écrire des données dans des fichiers, il supporte notamment l'utilisation de fichiers DLL (pour *dynamic link library*) permettant l'ajout de code utilisateur externe sous forme de bibliothèque dynamique.

Des « fonctions » codées par l'utilisateur sont intéressantes notamment pour exécuter certains calculs externes dont les résultats sont ensuite importés à la simulation. Ces calculs peuvent avoir lieu au début et à la fin de chaque réplication, voire même pendant la simulation. Ils peuvent utiliser plusieurs données provenant de la simulation telles que :

- Ses variables globales ;
- Les attributs des entités qui circulent ;
- Les informations sur les ressources et moyens de stockage ;
- Des « variables système » comme « TNOW » et « TFIN » qui indiquent respectivement la date courante et la date de fin de simulation pour la réplication en cours.

Pour que la connexion entre un modèle Arena et un fichier DLL soit bien établie, il faut également que les données de la simulation puissent être identifiées de façon unique. Ceci se fait par leur type (variable, ressource, file d'attente, attribut, système) et leur numéro.

Des fichiers auxiliaires sont fournis avec le logiciel Arena pour supporter une telle utilisation : « msuserc.dsw », « userc.cpp », « smsim.lib », « simlib.h », « msuserc.def », « stdafx.h ». C'est le fichier « userc.cpp » qui doit être adapté selon les besoins. Il fournit des fonctions prédéfinies qui permettent la communication entre le fichier DLL et le modèle de simulation. La définition des fonctions d'accès aux données est quant à elle fournie par « simlib.h ». Décrivons brièvement les deux fonctions de « userc.cpp » que nous avons exploitées dans le cadre de cette thèse :

- « cevent » : Elle définit la logique associée à chaque événement de type n, où n lui est passé en paramètre et correspond au numéro du bloc « Event » du modèle Arena déclenché par le passage d'une entité ;

- « cwrap » : Elle définit la logique de fin de réplication. Nous l'utilisons uniquement après la dernière réplication.

Une fois le fichier modifié et compilé, la DLL résultante est par défaut nommée « msuserc.dll » bien que ce nom soit modifiable. À noter qu'elle doit être générée de préférence par des compilateurs Microsoft (version 5.0 ou supérieure) supportés par Arena pour éviter des problèmes de compatibilité. Pour que le modèle Arena puisse utiliser la bibliothèque générée, il suffit de la saisir parmi les paramètres d'exécution. Voici la procédure : sur l'onglet « Run » de la barre d'outils, option « Setup », onglet « Run Control », cocher la case « Load User-coded .DLL ». Ceci permet de spécifier le nom de la DLL dans le champ correspondant. Il peut être nécessaire de fournir le chemin complet de la DLL si celle-ci ne se trouve pas dans le même répertoire que le modèle de simulation. Une façon alternative de l'indiquer est à travers le bloc « Replicate » avec le champ « DLL Name in Double Quotes ».

Le logiciel peut être utilisé pour exécuter directement un modèle de simulation, offrant dans ce cas la possibilité d'utiliser des animations graphiques. Il peut également être utilisé de façon « externe » en une ligne de commande (application console) ou à travers un programme qui y fait appel. C'est alors le processeur Siman (« siman.exe ») qui lit et exécute le modèle. Pour ce faire, il faut compiler le modèle de simulation initialement au format « .doe », ce qui va générer automatiquement un fichier exécutable par Siman au format « .p ». Des options peuvent être ajoutées en paramètre d'exécution. Il faut en particulier spécifier le nom du fichier DLL, même si celui-ci a déjà été indiqué lors de la construction du modèle comme expliqué auparavant. À noter que pour cette forme d'exécution, le chemin utilisé doit être sous sa forme courte car les noms de fichiers longs ne sont pas pris en charge par les processeurs de ligne de commande. Considérons un modèle Arena et une DLL nommés respectivement « MonModele.p » et « MaDLL.dll », placés tous les deux dans un même répertoire. Nous pourrions lancer une exécution à partir de ce répertoire avec la commande suivante : « siman MonModele.p -uMaDLL.dll ».

Annexe C

Benchmarks

Nous détaillons dans cet annexe les deux exemples de *benchmarks* que nous avons développés afin d'étudier l'adéquation et l'efficacité de nos choix d'implémentation, à savoir le couplage entre μGP et Arena.

C.1 *Benchmark* 1 : Affectation des produits aux machines

Pour cet exemple, nous avons un système de production mono-produit, c'est-à-dire où un seul type de produit est pris en charge. Les commandes sont unitaires (une commande = un produit) et arrivent toutes les 7,5 minutes.

Le système contient deux machines en parallèle : M_{10} et M_{40}. Elles diffèrent quant au temps nécessaire pour le traitement d'un produit : 10 minutes pour la machine M_{10} et 40 minutes pour M_{40}. Chaque machine a sa propre file d'attente.

Une fois une commande arrivée dans le système, il faut tout de suite l'affecter à l'une des deux machines. Si la machine sélectionnée est libre, le traitement commence immédiatement, sinon elle est placée dans la file d'attente correspondante gérée selon l'ordre d'arrivée des commandes (FIFO).

La décision concerne alors uniquement l'affectation des commandes. Elle doit être prise en vue d'équilibrer la charge entre les deux machines. Soient $nq_{M10,t}$ le nombre de produits dans la file d'attente de la machine M_{10} à l'instant t et $nq_{M40,t}$ l'équivalent pour la machine M_{40}. La charge d'une machine est calculée par le nombre de produits dans sa file d'attente multiplié par le temps de traitement d'un produit. Nous avons donc $cg_{M10,t} = 10 \cdot nq_{M10,t}$ et $cg_{M40,t} = 40 \cdot nq_{M40,t}$. Nous cherchons ainsi à ce que $cg_{M10,t}$ soit au plus proche de $cg_{M40,t}$. Autrement dit, la fonction objectif peut être énoncée comme suit :

$$Minimiser\ z = |nq_{M10,t} - 4 \cdot nq_{M40,t}| \tag{C.1}$$

Toutes les données sont déterministes de sorte que le résultat soit connu à l'avance. En effet, une règle d'affectation évidente consiste à affecter 4 produits à la machine M_{10} pour chaque produit affecté à la M_{40}. En cas d'égalité des charges, si nous priorisons par

exemple l'affectation à la machine la plus lente, nous nous attendons à un programme qui se rapproche de :

if $(nq_{M40,t} = 0)$ $d_t = M_{40}$
if $(ratioM10M40_t < 4)$ $d_t = M_{10}$
$d_t = M_{40}$

où $ratioM10M40_t = \frac{nq_{M10,t}}{nq_{M40,t}}$ ou plus précisément $\frac{nq_{M10,t}}{max\{nq_{M40,t};0,1\}}$ pour éviter une division par zéro. En outre, $d_t = M_{10}$ et $d_t = M_{40}$ représentent respectivement une affectation aux machines M_{10} et M_{40}.

Le système a été simulé sur 10080 minutes, soit l'équivalent d'une semaine de fonctionnement.

Nous avons testé 2 configurations pour la PGL combinées à 4 configurations pour un individu, ce qui nous donne 8 expérimentations.

Nous considérons les instructions suivantes :

I0. $d_t = M_{10}$

I1. if $(\{nq_{M10,t}\,;\, nq_{M40,t}\} = 0)$ jumpTo *label*

I2. if $(\{nq_{M10,t}\,;\, nq_{M40,t}\} = 0)$ $d_t = \{M_{10}\,;\, M_{40}\}$

I3. if $(ratioM10M40_t\ \{>\,;\,<\}\ \{[0{,}2\,;\,5]\})$ jumpTo *label*

I4. if $(ratioM10M40_t\ \{>\,;\,<\}\ \{[0{,}2\,;\,5]\})$ $d_t = \{M_{10}\,;\, M_{40}\}$

I5. if $(\{cg_{M10,t}\,;\, cg_{M40,t}\}\ \{=\,;\,>\,;\,<\}\ \{cg_{M10,t}\,;\, cg_{M40,t}\})$ jumpTo *label*

I6. if $(\{cg_{M10,t}\,;\, cg_{M40,t}\}\ \{=\,;\,>\,;\,<\}\ \{cg_{M10,t}\,;\, cg_{M40,t}\})$ $d_t = \{M_{10}\,;\, M_{40}\}$

où leur activation ou non en définit deux groupes détaillés dans le Tableau C.1.

Instructions	$Groupe^{Ratio}$	$Groupe^{Charge}$
I0	oui	oui
I1	oui	oui
I2	oui	oui
I3	oui	
I4	oui	
I5		oui
I6		oui

Tableau C.1 – Instructions activées pour les individus pour le premier exemple de *benchmark*

Chacun des groupes $Groupe^{Ratio}$ et $Groupe^{Charge}$ peut être combiné à deux autres groupes qui précisent les tailles plus ou moins strictes des individus comme l'indique le Tableau C.2. Ces tailles font référence au nombre d'instruction dans une logique décisionnelle (Section 4.8.2.1).

Tailles d'individus	$Taille^-$	$Taille^+$
Minimale	3	2
Maximale	10	20
Moyenne	5	5
Écart-type	2	5

Tableau C.2 – Paramètres concernant les individus pour le premier exemple de *benchmark*

Les 4 configurations résultantes pour un individu sont récapitulées dans le Tableau C.3. La dernière colonne indique par exemple que la configuration $Inst_{Charge^+}$ est une combinaison du $Groupe^{Charge}$ à la $Taille^+$.

	$Inst_{Ratio^-}$	$Inst_{Ratio^+}$	$Inst_{Charge^-}$	$Inst_{Charge^+}$
$Groupe^{Ratio}$	x	x		
$Groupe^{Charge}$			x	x
$Taille^-$	x		x	
$Taille^+$		x		x

Tableau C.3 – Paramètres concernant les individus pour le premier exemple de *benchmark*

Quant aux configurations de la PGL, elles diffèrent par rapport à la taille de population V_{max} ainsi qu'aux deux autres paramètres qui en dépendent : $V_{elite} = \lceil 0,1 \cdot V_{max} \rceil$ et $NbOp = V_{max}$.

Paramètres	$Config^{15}$	$Config^{30}$
V_{max}	15	30
V_{elite}	2	3
Immortalité	Non	Non
$NbOp$	15	30
z_{min}	0	0
P_{max}	1000	1000
$V_{tournoi}$	2	2

Tableau C.4 – Configurations de la PGL utilisées pour le premier exemple de *benchmark*

Les notations sont décrites dans la Section 4.8.2.3, et ce à l'exception de z_{min}. Ce dernier indique qu'ici, le critère d'arrêt est une valeur cible pour le *fitness*, à savoir $z_{min} = 0$. Pour assurer la fin de l'exécution, nous utilisons tout de même le critère $P_{max} = 1000$ plafonnant le nombre maximum de générations, mais ce dernier s'avérera inutile.

En effet, toutes les expérimentations ont atteint $z = z_{min} = 0$. Le Tableau C.5 indique le temps de calcul (en secondes), le nombre de générations P_{fin} ainsi que le nombre d'évaluations $NbEval$ nécessaires à chaque expérimentation pour atteindre l'objectif escompté. Par ailleurs, puisqu'une valeur de z entre 0 et 4 revient à une charge équilibrée

entre les machines, nous indiquons ces mêmes informations lorsque la performance cible est $z'_{min} = 4$, plus facile à atteindre.

	z_{min}			z'_{min}		
Expérimentation	*Temps (s)*	P_{fin}	*NbEval*	*Temps (s)*	P_{fin}	*NbEval*
$<Config^{15}$ - $Inst_{Ratio^-}>$	571	52	991	384	35	663
$<Config^{15}$ - $Inst_{Ratio^+}>$	31	3	58	31	3	58
$<Config^{15}$ - $Inst_{Charge^-}>$	55	7	105	8	0	15
$<Config^{15}$ - $Inst_{Charge^+}>$	258	26	471	8	0	15
$<Config^{30}$ - $Inst_{Ratio^-}>$	37	2	71	37	2	71
$<Config^{30}$ - $Inst_{Ratio^+}>$	609	25	1004	16	0	30
$<Config^{30}$ - $Inst_{Charge^-}>$	15	0	30	15	0	30
$<Config^{30}$ - $Inst_{Charge^+}>$	85	5	168	15	0	30

Tableau C.5 – Résultats pour le premier exemple de *benchmark*

Ces résultats indiquent la pertinence du logiciel qui non seulement a pu trouver la logique décisionnelle recherchée mais l'a surtout fait dans des temps très raisonnables.

C.2 *Benchmark* 2 : Affectation des produits aux machines sujette à pénalisation

Ce deuxième exemple est une adaptation du précédent. Le système de production prend maintenant en charge deux types de produit que nous noterons simplement A et B. Les commandes restent unitaires et arrivent toutes les 7,5 minutes pour les produits de type A et toutes les 4,5 minutes pour ceux de type B.

Une troisième machine M_5 est ajoutée au système. Les machines M_5, M_{10} et M_{40} fonctionnent en parallèle. Elles diffèrent quant à leur mode opératoire et leur temps de traitement. Les machines M_{10} et M_{40} prennent respectivement 10 et 40 minutes pour traiter un produit de type A pour lequel elles sont spécialisées. Elles peuvent tout de même traiter un produit de type B mais sur un temps de traitement pénalisé de 10000 minutes. Pour la machine M_5, c'est le contraire : sa spécialisation fait que le temps opératoire pour un produit de type B est de 5 minutes alors qu'il est de 10000 minutes pour un produit de type A.

Encore une fois, chaque machine a sa propre file d'attente qui fonctionne par ordre d'arrivée. L'affectation d'une commande à une des trois machines est faite immédiatement dès son arrivée. L'objectif reste celui d'équilibrer la charge entre les machines M_{10} et M_{40} mais tout en faisant attention à ce que les machines traitent uniquement le type de produit pour lequel elles sont spécialisées.

Nous avons toujours l'information sur le nombre de produits dans la file d'attente de

chaque machine à tout instant t : $nq_{M5,t}$, $nq_{M10,t}$, $nq_{M40,t}$. Les données sont ici encore déterministes. Mais puisque les temps de traitement peuvent être pénalisés selon le type de produit, la charge des machines M_{10} et M_{40} est une estimation optimiste : elle est toujours calculée comme avant, ce qui suppose que seuls des produits de type A y seront affectés. La mauvaise affectation influence cependant le nombre de commandes livrées de sorte que la fonction objectif puisse maintenant être énoncée comme suit :

$$Maximiser\ z = NbLivre - (|nq_{M10,t} - 4 \cdot nq_{M40,t}|) \tag{C.2}$$

où $NbLivre$ indique le nombre total de produits livrés, tous types confondus.

Ici, même si la valeur optimale de z ne peut pas être connue à l'avance, la logique décisionnelle, elle, peut l'être. La règle d'affectation évidente consiste à affecter tout produit de type B à la machine M_5. Pour le type A, on procède comme dans l'exemple précédent : 4 produits affectés à la machine M_{10} pour 1 seul affecté à M_{40}. Nous nous attendons donc à un programme qui se rapproche de :

if $(TypeProduit_t = B)$ $d_t = M_5$
if $(nq_{M40,t} = 0)$ $d_t = M_{40}$
if $(ratioM10M40_t < 4)$ $d_t = M_{10}$
$d_t = M_{40}$

où $TypeProduit_t \in \{A; B\}$ indique le type du produit devant être affecté à l'instant t.

Le système a également été simulé sur 10080 minutes de fonctionnement.

Nous avons testé cette fois-ci 2 configurations pour la PGL combinées à 3 configurations pour un individu, ce qui nous donne 6 expérimentations.

Nous considérons les instructions suivantes :

I0. $d_t = M_{10}$
I1. if $(TypeProduit_t = \{A; B\})$ jumpTo *label*
I2. if $(TypeProduit_t = \{A; B\})$ $d_t = \{M_5\,;\, M_{10}\,;\, M_{40}\}$
I3. if $(\{nq_{M5,t}\,;\, nq_{M10,t}\,;\, nq_{M40,t}\} = 0)$ jumpTo *label*
I4. if $(\{nq_{M5,t}\,;\, nq_{M10,t}\,;\, nq_{M40,t}\} = 0)$ $d_t = \{M_5\,;\, M_{10}\,;\, M_{40}\}$
I5. if $(ratioM10M40_t\ \{>\,;\,<\}\ \{[0{,}2\,;\,5]\})$ jumpTo *label*
I6. if $(ratioM10M40_t\ \{>\,;\,<\}\ \{[0{,}2\,;\,5]\})$ $d_t = \{M_5\,;\, M_{10}\,;\, M_{40}\}$
I7. if $(\{cg_{M5,t}\,;\, cg_{M10,t}\,;\, cg_{M40,t}\}\ \{=\,;\,>\,;\,<\}\ \{cg_{M5,t}\,;\, cg_{M10,t}\,;\, cg_{M40,t}\})$ jumpTo *label*
I8. if $(\{cg_{M5,t}\,;\, cg_{M10,t}\,;\, cg_{M40,t}\}\ \{=\,;\,>\,;\,<\}\ \{cg_{M5,t}\,;\, cg_{M10,t}\,;\, cg_{M40,t}\})$ $d_t = \{M_5\,;\, M_{10}\,;\, M_{40}\}$
I9. if $(\{nq_{M5,t}\,;\, nq_{M10,t}\,;\, nq_{M40,t}\}\ \{>\,;\,<\}\ \{nq_{M5,t}\,;\, nq_{M10,t}\,;\, nq_{M40,t}\})$ jumpTo *label*
I10. if $(\{nq_{M5,t}\,;\, nq_{M10,t}\,;\, nq_{M40,t}\}\ \{>\,;\,<\}\ \{nq_{M5,t}\,;\, nq_{M10,t}\,;\, nq_{M40,t}\})$ $d_t = \{M_5\,;\, M_{10}\,;\, M_{40}\}$

Instructions	$Inst^{Ratio}$	$Inst^{RatioFile}$	$Inst^{Charge}$
I0	oui	oui	oui
I1	oui	oui	oui
I2	oui	oui	oui
I3	oui	oui	oui
I4	oui	oui	oui
I5	oui	oui	
I6	oui	oui	
I7			oui
I8			oui
I9		oui	
I10		oui	

Tableau C.6 – Instructions activées pour les individus pour le deuxième exemple de *benchmark*

où leur activation ou non en définit trois groupes détaillés dans le Tableau C.6.

Les tailles des individus sont ici définies indépendamment du groupe comme indiqué dans le Tableau C.7.

Tailles d'individus	Valeur
Minimale	3
Maximale	15
Moyenne	7
Écart-type	2

Tableau C.7 – Paramètres concernant les individus pour le deuxième exemple de *benchmark*

Les configurations de la PGL sont basées sur celles de l'exemple précédent mais cette fois-ci en utilisant le nombre de générations sans amélioration P_{stagne} comme critère d'arrêt associé à P_{max}.

Paramètres	$Config^{15}$	$Config^{30}$
V_{max}	15	30
V_{elite}	2	3
Immortalité	Non	Non
$NbOp$	15	30
P_{max}	100	100
P_{stagne}	50	50
$V_{tournoi}$	2	2

Tableau C.8 – Configurations de la PGL utilisées pour le deuxième exemple de *benchmark*

Le Tableau C.9 indique le temps de calcul (en minutes), le nombre de générations

ainsi que le nombre d'évaluations nécessaires à chaque expérimentation pour arriver à leur performance finale z.

Expérimentation	z	*Temps (min)*	P_{fin}	*NbEval*
$<Config^{15}$ - $Inst_{Ratio}>$	3270	13,40	58	1197
$<Config^{15}$ - $Inst_{RatioFile}>$	3273	16,75	71	1473
$<Config^{15}$ - $Inst_{Charge}>$	3267	14,45	76	1358
$<Config^{30}$ - $Inst_{Ratio}>$	3273	47,32	100	4497
$<Config^{30}$ - $Inst_{RatioFile}>$	3273	33,58	79	3331
$<Config^{30}$ - $Inst_{Charge}>$	3218	30,83	71	2975

Tableau C.9 – Résultats pour le deuxième exemple de *benchmark*

Seule l'expérimentation $<Config^{30}$ - $Inst_{Ratio}>$ a atteint le nombre maximum de générations, toutes les autres étant arrêtées par le nombre maximum de générations sans amélioration. Toutes les expérimentations ont atteint des performances équivalentes, à l'exception de $<Config^{30}$ - $Inst_{Charge}>$, en léger retrait. Ces résultats confirment encore une fois que nos choix sont compatibles avec nos objectifs.

Annexe D

Complément relatif à la mise en place des connexions nécessaires à notre approche d'extraction de connaissances

---

Les échanges entre l'utilisateur et les logiciels μGP et Arena sont représentés par un diagramme de séquence qui, pour plus de lisibilité, a été découpé en deux : le second diagramme reprend en détails l'appel de la méthode "evaluerFichier(Fichier)" du premier. Il intègre donc la composante DLL implémentée en langage C++.

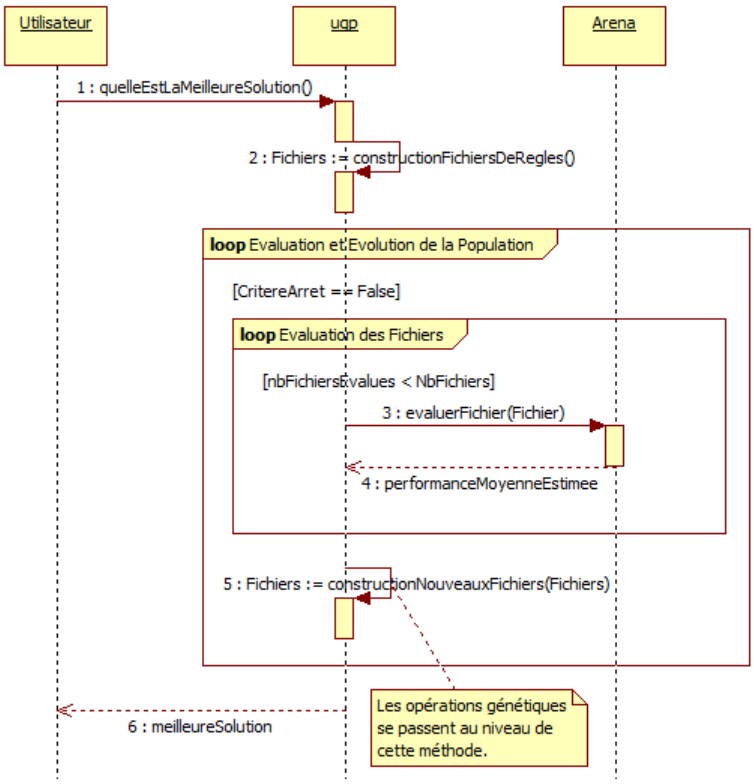

Figure D.1 – Diagramme de séquence résumant le fonctionnement de notre approche : Interactions utilisateur-logiciels

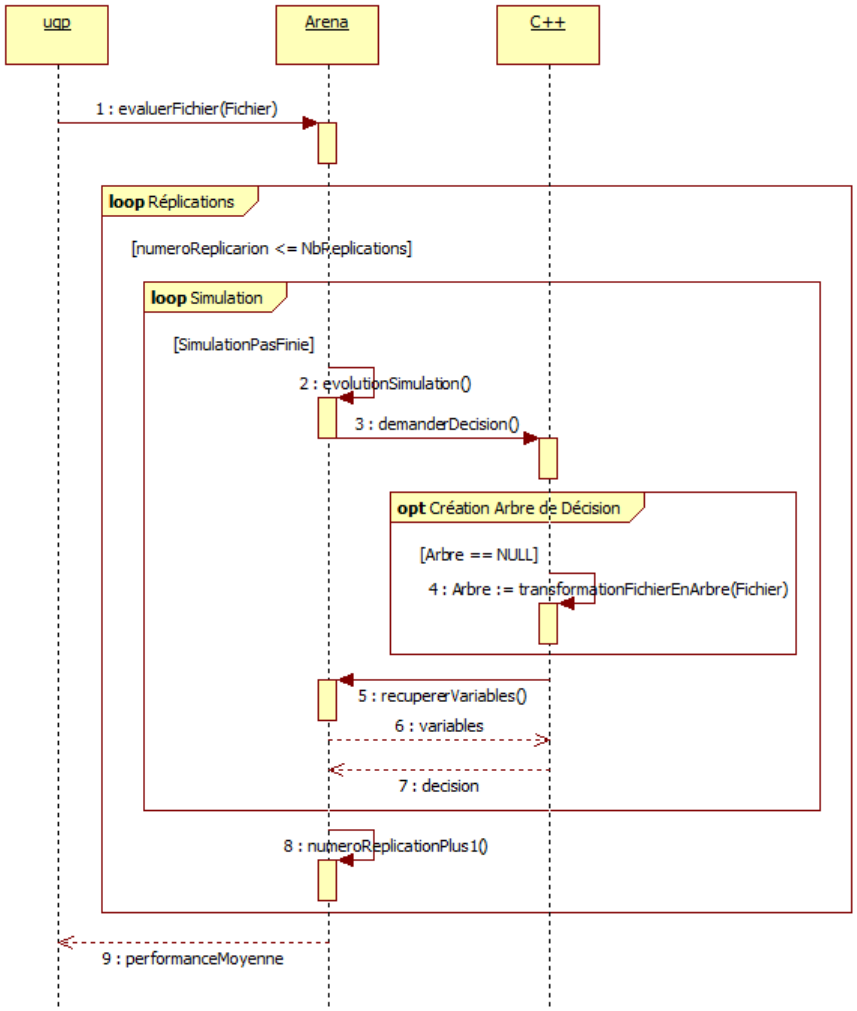

FIGURE D.2 – Diagramme de séquence résumant l'évaluation par simulation dans notre approche : Interactions entre logiciels

Annexe E

Complément relatif au traitement des contraintes pour le problème du ConWIP

Nous avons évoqué dans cette thèse la présence d'un certain nombre de contraintes liées à l'adaptation d'un système. Pour le système ConWIP traité, il y avait notamment la Contrainte 5.1 sur le nombre de cartes qui y circulent. Pour satisfaire cette contrainte, nous avons cité une alternative. Il s'agit de la définition d'un sous-ensemble à chaque instant t contenant seulement les actions valides à cet instant :

$$D_{1,t} = [max\{K_{Min} - a_{1,1,t}; -d_{max}\}; min\{K_{Max} - a_{1,1,t}; d_{max}\}] \tag{E.1}$$

de sorte que $d_t \in D_{1,t} \subseteq D_1$.

L'attribut $a_{1,1,t}$ prendrait alors sa valeur dans le sous-ensemble $E_{1,1,t} \subseteq E_{1,1}$:

$$E_{1,1,t} = [max\{K_{Min}; a_{1,1,t} - d_{max}\}; min\{K_{Max}; a_{1,1,t} + d_{max}\}] \tag{E.2}$$

Ce raisonnement développé pour notre exemple de système ConWIP pourrait être adapté à certains types de contraintes avec les précautions nécessaires.

Ainsi, pour l'exemple de la répartition de machines (Section 5.3.3.4), nous aurions eu : $D_{1,t} = [max\{MMin_A - NbM_{A,t}; -1\}; min\{MMax_A - NbM_{A,t}; 1\}]$ ainsi que le sous-ensemble $E_{1,1,t} \subseteq E_{1,1}$ correspondant.

Nous ne prétendons toutefois pas que cette solution soit toujours applicable. Elle reste limitée, le traitement de contraintes s'avérant souvent bien plus complexe.

Annexe F

Configurations de la PGL utilisées pour le problème du ConWIP

Paramètres	$Config^{--}$	$Config^{++}$	$Config^{-+}$	$Config^{+-}$
V_{max}	20	30	20	30
$V_{elite} = 0,1 \cdot V_{max}$	2	3	2	3
Immortalité	Non	Non	Non	Non
$NbOp = V_{max}$	20	30	20	30
P_{max}	50	100	100	50
$P_{stagne} = 0,5 \cdot P_{max}$	25	50	50	25
$V_{tournoi}$	2	2	2	2

Tableau F.1 – Groupe de test 1 : Population et critères d'arrêt

Paramètres	$Config^{++1}$	$Config^{++}$	$Config^{++3}$	$Config^{++4}$	$Config^{++\Delta}$
V_{max}	30	30	30	30	30
V_{elite}	3	3	3	3	3
Immortalité	Non	Non	Non	Non	Non
$NbOp$	30	30	30	30	30
P_{max}	100	100	100	100	100
P_{stagne}	50	50	50	50	50
$V_{tournoi}$	1	2	3	4	[2 ; 4]
Paramètres	$Config^{--1}$	$Config^{--}$	$Config^{--3}$	$Config^{--4}$	$Config^{--\Delta}$
V_{max}	20	20	20	20	20
V_{elite}	2	2	2	2	2
Immortalité	Non	Non	Non	Non	Non
$NbOp$	20	20	20	20	20
P_{max}	50	50	50	50	50
P_{stagne}	25	25	25	25	25
$V_{tournoi}$	1	2	3	4	[2 ; 4]

Tableau F.2 – Groupe de test 2 : Sélection par tournoi

Paramètres	$Config^{-+}$	$Config^{M30}$	$Config^{M40}$	$Config^{I10}$	$Config^{I15}$	$Config^{I20}$
V_{max}	20	20	20	20	20	20
V_{elite}	2	2	2	2	2	2
Immortalité	Non	Non	Non	Oui	Oui	Oui
$NbOp$	20	30	40	10	15	20
P_{max}	100	100	100	100	100	100
P_{stagne}	50	50	50	50	50	50
$V_{tournoi}$	2	2	2	2	2	2

Tableau F.3 – Groupe de test 3 : Immortalité (et nombre d'opérations)

Paramètres	$Config^{*+}$	$Config^{*+I}$
V_{max}	100	100
V_{elite}	15	15
Immortalité	Non	Oui
$NbOp$	100	50
P_{max}	100	100
P_{stagne}	50	50
$V_{tournoi}$	2	2

Tableau F.4 – Groupe de test 4 : Immortalité (pour une population de 100 individus)

Paramètres	$Config^{-+}$	$Config^{++}$	$Config^{*+}$
V_{max}	20	30	100
V_{elite}	2	3	15
Immortalité	Non	Non	Non
$NbOp$	20	30	100
P_{max}	100	100	100
P_{stagne}	50	50	50
$V_{tournoi}$	2	2	2

Tableau F.5 – Groupe de test 5 : Scénario mixte

Annexe G

Exemple de sortie générée par μGP

Nous trouvons ci-dessous une logique décisionnelle telle que générée par le logiciel de programmation génétique linéaire μGP. Elle correspond au programme donnant origine à l'arbre de décision de la Figure 5.11 avant toute simplification ou correction.

if $(FSL_t \geq 7)$ $d_t = -2$
if $(FSL_t > 6)$ $d_t = 1$
if $(FSL_t \geq 7)$ $d_t = -2$
if $(FSL_t > 6)$ $d_t = 1$
if $(FSL_t > 2)$ $d_t = -1$
if $(FSL_t > 7)$ jumpTo nGXMP
if $(U_t < 5)$ jumpTo nGXM2
nGXMP : if $(FSL_t \geq 7)$ $d_t = -2$
if $(U_t > 0)$ jumpTo nGXMR
nGXMR : if $(FSL_t \geq 7)$ jumpTo nGXNB
if $(U_t > 0)$ jumpTo nGXMT
nGXMT : if $(FSL_t \geq 7)$ jumpTo nGXND
if $(FSL_t \geq 5)$ $d_t = 2$
if $(FSL_t > 7)$ jumpTo nGXMZ
if $(FSL_t > 2)$ $d_t = -1$
if $(FSL_t \geq 6)$ jumpTo nGXM4
if $(U_t < 4)$ $d_t = 2$
nGXMZ : if $(FSL_t < 1)$ $d_t = 2$
nGXM2 : if $(FSL_t > 7)$ jumpTo nGXM6
if $(FSL_t \leq 1)$ jumpTo nGXNB
nGXM4 : if $(U_t < 5)$ jumpTo nGXM5
nGXM5 : if $(FSL_t < 1)$ $d_t = 1$
nGXM6 : if $(FSL_t > 3)$ jumpTo nGXNL
if $(U_t > 0)$ jumpTo nGXNS
if $(FSL_t \geq 6)$ jumpTo nGXNF
nGXNB : if $(FSL_t \geq 7)$ $d_t = -2$
if $(FSL_t > 7)$ jumpTo nGXNG
nGXND : if $(FSL_t \leq 1)$ jumpTo nGXNK
if $(U_t > 0)$ $d_t = -2$
nGXNF : if $(FSL_t < 1)$ $d_t = 2$

nGXNG : if $(FSL_t > 7)$ jumpTo nGXNI
if $(FSL_t \geq 6)$ jumpTo nGXNM
nGXNI : if $(U_t < 4)$ $d_t = 2$
if $(FSL_t < 1)$ $d_t = 2$
nGXNK : if $(FSL_t > 7)$ jumpTo nGXNO
nGXNL : if $(U_t > 0)$ jumpTo nGXNM
nGXNM : if $(FSL_t \geq 7)$ jumpTo nGXN2
if $(FSL_t \leq 1)$ jumpTo nGXNU
nGXNO : if $(FSL_t \leq 1)$ jumpTo nGXNV
if $(U_t > 0)$ $d_t = -2$
if $(U_t < 5)$ jumpTo nGXNR
nGXNR : if $(FSL_t < 1)$ $d_t = 1$
nGXNS : if $(FSL_t > 3)$ jumpTo nGXN2
if $(U_t > 0)$ jumpTo nGXN2
nGXNU : if $(FSL_t > 7)$ jumpTo nGXNW
nGXNV : if $(U_t < 4)$ $d_t = 2$
nGXNW : if $(FSL_t > 7)$ jumpTo nGXN2
if $(FSL_t > 7)$ jumpTo nGXNZ
if $(U_t > 1)$ $d_t = 1$
nGXNZ : if $(U_t > 1)$ $d_t = 1$
nGXN2 : $d_t = 0$

Titre : Contribution de l'apprentissage par simulation à l'auto-adaptation des systèmes de production

Résumé : Pour rester performants et compétitifs, les systèmes de production doivent être capables de s'adapter pour faire face aux changements tels que l'évolution de la demande des clients. Il leur est essentiel de pouvoir déterminer quand et comment s'adapter (capacités, etc.). Malheureusement, de tels problèmes sont connus pour être difficiles. Les systèmes de production étant complexes, dynamiques et spécifiques, leurs gestionnaires n'ont pas toujours l'expertise nécessaire ni les prévisions suffisantes concernant l'évolution de leur système. Cette thèse vise à étudier la contribution que peut apporter l'apprentissage automatique à l'auto-adaptation des systèmes de production. Dans un premier temps, nous étudions la façon dont la littérature aborde ce domaine et en proposons un cadre conceptuel dans le but de faciliter l'analyse et la formalisation des problèmes associés. Ensuite, nous étudions une stratégie d'apprentissage à partir de modèles qui ne nécessite pas d'ensemble d'apprentissage. Nous nous intéressons plus précisément à une nouvelle approche basée sur la programmation génétique linéaire visant à extraire des connaissances itérativement à partir d'un modèle de simulation pour déterminer quand et quoi faire évoluer. Notre approche est implémentée à l'aide d'Arena et μGP. Nous l'appliquons à différents exemples qui concernent l'ajout/retrait de cartes dans un système à flux tiré, le déménagement de machines ou encore le changement de politique de réapprovisionnement. Les connaissances qui en sont extraites s'avèrent pertinentes et permettent de déterminer en continu comment chaque système peut s'adapter à des évolutions. De ce fait, elles peuvent contribuer à doter un système d'une forme d'intelligence. Exprimées sous forme d'un arbre de décision, elles sont par ailleurs facilement communicables à un gestionnaire de production. Les résultats obtenus montrent ainsi l'intérêt de notre approche tout en ouvrant de nombreuses voies de recherche.

Mots-clés : Systèmes de production, auto-adaptation, aide à la décision, extraction de connaissances, apprentissage automatique, simulation, programmation génétique linéaire.

Title : Simulation-based machine learning for the self-adaptation of manufacturing systems

Abstract : Manufacturing systems must be able to continuously adapt their characteristics to cope with the different changes that occur along their life, in order to remain efficient and competitive. These changes can take the form of the evolution of customers demand for instance. It is essential for these systems to determine when and how to adapt (e.g., through changes in capacities). Unfortunately, such issues are known to be difficult. As manufacturing systems are complex, dynamic and specific in nature, their managers do not always have all the necessary expertise nor accurate enough forecasts on the evolution of their system. This thesis aims at studying the possible contributions of machine learning to the self-adaptation of manufacturing systems. We first study how the literature deals with self-adaptation and we propose a conceptual framework to facilitate the analysis and the formalization of the associated problems. Then, we study a learning strategy relying on models, which presents the advantage of not requiring any training set. We focus more precisely on a new approach based on linear genetic programming that iteratively extracts knowledge from a simulation model. Our approach is implemented using Arena and μGP. We show its benefits by applying it to increase/decrease the number of cards in a pull control system, to move machines or to change the inventory replenishment policy. The extracted knowledge is found to be relevant for continuously determining how each system can adapt to evolutions. It can therefore contribute to provide these systems with some intelligent capabilities. Moreover, this knowledge is expressed in the simple and understandable form of a decision tree, so that it can also be easily communicated to production managers in view of their everyday use. Our results thus show the interest of our approach while opening many research directions.

Keywords : Manufacturing systems, self-adaptation, decision support, knowledge extraction, machine learning, simulation, linear genetic programming.
