Un algorithme génétique pour l'ordonnancement robuste: application au problème du flow shop hybride

Tarek Chaari

Thèse

en co-tutelle

pour obtenir le grade de

Docteur de l'Université de Valenciennes et du Hainaut-Cambrésis

Discipline : Automatique

Spécialité : Automatique, Génie Informatique

et le grade de

Docteur de la Faculté des Sciences Économiques et de Gestion de Sfax

Discipline : Méthodes Quantitatives

présentée et soutenue publiquement le 11 Mars 2010 par

Tarek CHAARI

Un algorithme génétique pour l'ordonnancement robuste : application au problème du flow shop hybride

Jury

Henri PIERREVAL	Pr. à l'Institut Français de Mécanique Avancée, Clermont-Ferrand	Président
Patrick SIARRY	Pr. à l'Université Paris-Est Créteil Val de Marne	Rapporteur
Talel LADHARI	Pr. agrégé habilité à diriger des recherches à l'École Supérieure des Sciences Économiques et Commerciales de Tunis	Rapporteur
Jacques TEGHEM	Pr. à la Faculté Polytechnique de Mons	Examinateur
Taïcir LOUKIL	Pr. à la Faculté des Sciences Économiques et de Gestion de Sfax	Directrice de thèse
Damien TRENTESAUX	Pr. à l'Université de Valenciennes et du Hainaut-Cambrésis	Co-directeur
Sondes CHAABANE	Maître de conférences à l'Université de Valenciennes et du Hainaut-Cambrésis	Co-encadrante

À mes parents,

à mon épouse Imen,

à ma chère petite fille Yasmine.

Remerciements

Ce n'est pas par tradition que cette page figure au préambule de ce mémoire, mais c'est plutôt un devoir moral et une reconnaissance sincère qui me pousse à la faire. Je serais en effet ingrat si je n'exprime pas ma reconnaissance et ma gratitude à tous ceux, de prés ou de loin, qui ont facilité ma tâche et m'ont permis de mener à bien ce travail.

Ce travail est issu d'une collaboration entre le Laboratoire d'Automatique, Mécanique, Informatique industrielles et Humaines " L.A.M.I.H " de l'université de Valenciennes et du Hainaut-Cambrésis et l'unité de recherche Gestion Industrielle et Aide à la Décision " G.I.A.D " de la Faculté des Sciences Économiques et de Gestion de Sfax en Tunisie. Je tiens à remercier Mme Taïcir Loukil, M. Habib Chabchoub, M. Damien Trentesaux et M. Christian Tahon qui ont permis cette co-tutelle, en espérant que celle-ci marque le début d'une collaboration des plus fructueuses entre les deux laboratoires. Je remercie tous les membres des laboratoires pour leur accueil et les moments agréables passés durant ma thèse.

Je tiens tout particulièrement à exprimer toute ma gratitude et mes vifs remerciements à mes professeurs Taïcir Loukil et Damien Trentesaux qui m'ont accordé leur confiance en me proposant ce sujet de thèse et pour leur aide avec leurs expériences et savoir faire. Je les prie de croire à ma respectueuse estime et ma sincère reconnaissance pour leurs conseils qui m'ont été très précieux et indispensables.

Je ne saurais oublier de remercier Mlle Sondes Chaabane pour sa collaboration durant ce travail de recherche avec ses connaissances dans ce domaine de recherche. Qu'elle me permet de l'exprimer l'assurance de ma gratitude et de mon profond respect.

Mes remerciements vont également à tous les membres de jury pour avoir accepté d'évaluer cette thèse.

Un grand Merci à mes parents qui m'ont donné la chance de poursuivre mes études et qui m'ont appris à surpasser les moments difficiles, à ma précieuse femme Imen et ma mignonne fille Yasmine pour leur patience éternelle.

Un algorithme génétique pour l'ordonnancement robuste : application au problème du flow shop hybride

Résumé : La plupart des méthodes d'ordonnancement considèrent un environnement déterministe où les données du problème sont connues. Néanmoins, en réalité, plusieurs sortes d'aléas peuvent être rencontrées et l'ordonnancement robuste permet en tenir compte. Dans cette thèse, notre intuition initiale est que, d'une part, un ordonnancement non robuste (déterministe) deviendra rapidement inefficace avec les incertitudes qu'un ordonnancement robuste, et d'autre part, un ordonnancement robuste sera moins efficace qu'un ordonnancement non robuste en l'absence d'incertitudes. Dans ce cadre, nous avons proposé un algorithme génétique pour l'ordonnancement robuste. Un nouveau mécanisme de résolution et un nouveau critère de robustesse permettant de trouver une solution de bonne performance et peu sensible aux incertitudes ont été développés. Une phase expérimentale a été menée, d'une part, pour vérifier l'efficacité de l'algorithme génétique pour l'ordonnancement déterministe, sans tenir compte des incertitudes, et d'autre part, pour valider l'algorithme génétique pour l'ordonnancement robuste par la simulation afin de juger la qualité de la robustesse face aux incertitudes. Nous avons intégré cette approche de robustesse dans une démarche méthodologique générique intégrant des techniques d'optimisation et de simulation pour l'aide au dimensionnement des systèmes de production basé sur des ordonnancements robustes. Les différents modules de la démarche ont été développés sous forme d'un outil d'aide au dimensionnement, dans le cadre d'un cas applicatif réel, celui du bloc opératoire dans le secteur hospitalier.

Mots clés : Ordonnancement, robustesse, efficacité, algorithme génétique, optimisation-simulation, dimensionnement, temps d'exécution incertain.

A genetic algorithm for robust scheduling : application of hybrid flow shop scheduling problem

Abstract : Most of scheduling methods consider a determinist environment where the data of the problem are known. Nevertheless, in reality, several kinds of risks can be considered and the robust scheduling allows taking into account it. In this thesis, our initial assumption is that, non-robust (deterministic) schedules will rapidly become inefficient with uncertainty, while robust schedules, less efficient than the non-robust ones without any uncertainty, will not deteriorate as fast as non-robust schedules do when uncertainty is introduced. In this context, we proposed a genetic algorithm for a robust scheduling. We developed a new mechanism of resolution and a new robustness criterion allowing to find a solution of good performance and little sensitive to the uncertainties. A computational experiment has been performed, first, to verify the efficiency of the genetic algorithm for deterministic scheduling, without taking uncertainties into account, second, to validate the genetic algorithm for robust scheduling using simulation to evaluate the robustness quality in front of uncertainties. We integrate this approach of robustness in a methodological approach based on simulation-optimisation techniques for the help to the sizing manufacturing systems based on robust scheduling. The various modules of this approach were developed as a sizing and design-aided tool applied in a real application case : operating theatres in the hospital sector.

Keywords : Scheduling, robustness, effectiveness, genetic algorithm, optimization-simulation, sizing, uncertain processing times.

Table des matières

Introduction générale

L'ordonnancement consiste à organiser, dans le temps, la réalisation des tâches compte tenu des contraintes de temps et de ressources, afin d'optimiser un ou plusieurs objectifs. Plusieurs méthodes d'ordonnancement ont été développées dans la littérature. La plupart de ces méthodes considèrent un environnement déterministe où les données du problème sont connues : toutes les tâches à ordonnancer et toutes les contraintes sont connues à l'avance et restent inchangés dans le temps. Alors qu'en réalité il existe plusieurs sortes d'aléas qui vont perturber le plan de production, même les industriels sont incapables de fournir des données fiables ou satisfaisantes au problème posé. En plus, les données des problèmes d'ordonnancement sont assorties de valeurs numériques pour présenter l'état du système réel (temps d'exécution, vitesse d'une machine, etc.). Ces données numériques peuvent être mal connues ou être susceptibles de changer, ce qui est une autre source d'incompatibilité possible du modèle avec la réalité. La compatibilité du modèle avec la réalité est sans cesse remise en cause par la définition et la modélisation des informations d'entrée du modèle. Ces informations sont parfois qualifiées comme incertaines, incomplètes ou imprécises.

La prise en compte des incertitudes sur les données du problème a donné lieu à un nouveau champs de recherche appelée *ordonnancement robuste*. Il s'agit toujours d'organiser, dans le temps, la réalisation des tâches compte tenu des contraintes de temps et de ressources, afin d'optimiser un ou plusieurs objectifs, mais en prenant en compte cette fois des perturbations qui peuvent survenir. La solution produite doit alors résister aux perturbations de l'environnement, c'est-à-dire être toujours utilisable en cas d'imprévus,

mais il faut aussi être sûr du maintien des performances, face à un ensemble de perturbations. Autrement dit, notre intuition initiale est que, d'une part, un ordonnancement non robuste deviendra plus rapidement inefficace avec les incertitudes qu'un ordonnancement robuste, et d'autre part, un ordonnancement robuste sera moins efficace qu'un ordonnancement non robuste, en l'absence d'incertitudes. L'application d'une approche basée sur les métaheuristiques permet de déterminer une solution robuste de "bonne" qualité et en temps de calcul "raisonnable".

En effet, l'ordonnancement robuste joue un rôle essentiel dans divers contextes, en informatique, en administration, en production, mais aussi dans le domaine des services, en particulier pour aider au dimensionnement des systèmes hospitaliers. Dans certaines situations, nous disposons d'informations précises sur l'utilisation future d'un système à concevoir, ce qui nous permet d'envisager, dès la phase de conception, un ensemble d'ordonnancements possibles des activités. Dans ce cas, il serait intéressant de baser les choix de conception sur les résultats des ordonnancements, mais à condition qu'ils soient robustes face aux évènements incertains qui pourraient survenir dans le futur. Sans cela, la confiance accordée aux performances des ordonnancements serait faible et le risque en les utilisant en conception serait alors très fort. Ainsi, pour être utilisés à des fins de conception, les ordonnancements proposés doivent être robustes afin de garantir le maintien de leurs performances face aux incertitudes.

Une des applications possibles de notre contribution consiste ainsi à exploiter un ensemble d'ordonnancements robustes basés sur des contraintes et des objectifs d'exploitation future et potentielle, en phase de conception et de dimensionnement. Souvent, ces contraintes (par exemple, les ressources à utiliser) et les objectifs (par exemple, une cadence cible) sont disponibles dès la phase de conception. Ceci revient donc à prendre les décisions relatives à la conception sur la base d'un ensemble d'ordonnancements représentant des scénarios d'utilisation future, mais qui se doivent d'être robustes pour que le concepteur puisse effectivement avoir confiance dans les résultats présentés. A cet effet, nous proposons une approche intégrant des techniques d'optimisation couplées avec des techniques de simulation.

Le contenu de cette thèse est présenté en quatre chapitres.

Le premier chapitre présentera un panorama de la littérature sur l'ordonnancement sous incertitudes, où nous présenterons, en premier lieu, les différents types d'incertitudes et les techniques de leurs modélisations. En deuxième lieu, nous proposerons une nouvelle classification des différentes approches de l'ordonnancement dans un milieu incertain : on distingue les approches proactives, les approches réactives et les approches hybrides, qui sont composées de deux approches : les approches prédictives-réactives et les approches proactives-réactives. Pour chacune de ces approches, nous présenterons ses avantages et inconvénients, ainsi que les méthodes de résolution développées dans la littérature.

Dans le deuxième chapitre, nous proposerons une nouvelle méthode d'ordonnancement sous incertitudes basée sur un algorithme génétique. L'objectif est d'obtenir une solution "robuste" de bonne performance et qui est peu sensible à ces incertitudes. Une description complète du fonctionnement des algorithmes génétiques sera faite. Nous présenterons les opérateurs génétiques participant à l'exploration de l'espace de recherche et les paramètres nécessaires pour la convergence vers de bonnes solutions. Dans la deuxième partie de ce chapitre, nous développons un algorithme génétique pour l'ordonnancement robuste. Nous proposons un nouveau mécanisme de résolution intégré dans l'algorithme génétique pour donner une solution robuste. Cette solution est évaluée par un nouveau critère de robustesse, qui peut être appliqué sur différents problèmes d'ordonnancement. Ce critère est constitué d'une agrégation de deux objectifs permettant de trouver une solution de bonne performance et peu sensible aux incertitudes. Une organisation de type *flow shop* hybride constituera notre cas d'étude.

Dans le troisième chapitre, nous décrirons les résultats de la phase expérimentale qui a été menée. Le but de cette phase est double : d'abord, nous vérifions l'algorithme génétique pour l'ordonnancement déterministe sur des *benchmarks* existant dans la littérature, dont l'objectif est de minimiser seulement la date d'achèvement des travaux (*makespan*), sans tenir compte des incertitudes sur les temps d'exécution. Puis nous validons l'algorithme génétique pour l'ordonnancement robuste par la simulation, pour juger la qualité de la robustesse face aux incertitudes.

Le quatrième chapitre sera présenté en deux parties. Dans la première partie, nous traitons une nouvelle approche dans le cadre de la conception des systèmes de production, celle du problème de dimensionnement, en tenant compte de l'ordonnancement. Un état de l'art sur le problème du dimensionnement des systèmes de production sera développé. Nous présenterons les différentes approches de résolution, telles que les approches analytiques, les approches basées sur l'optimisation, les approches basées sur la simulation et les approches optimisation-simulation. L'objectif de cette partie est de travailler conjointement le dimensionnement et l'ordonnancement, dès la phase de conception. L'application en ordonnancement robuste constitue la seconde phase, puisque la qualité du dimensionnement sera basée sur la qualité des ordonnancements proposés et leur robustesse face aux perturbations.

A cet égard, nous développerons, dans un premier temps, une approche statique (déterministe), qui consiste à dimensionner les machines dans un atelier de production, en tenant compte de la performance et des types d'ordonnancements prévisionnels. Au vu des résultats obtenus, cette approche peut fournir des solutions exactes pour des problèmes de petites tailles.

Dans un deuxième temps, une approche robuste sera développée. Nous proposerons une démarche méthodologique intégrant des techniques d'optimisation et de simulation pour l'aide au dimensionnement des systèmes de production basé sur des ordonnancements robustes. Cette démarche est générique et peut être appliquée aux différents types d'organisation (*flow shop* hybride, *Job shop* flexible, machines parallèles, etc.). Nous développerons les différents modules de la démarche sous forme d'un outil d'aide au dimensionnement, dans le cadre d'un cas applicatif réel, celui du bloc opératoire dans le secteur hospitalier. En effet, la problématique du dimensionnement du bloc opératoire consiste, dans ce cas, à déterminer le nombre de salles d'opération et de lits dans la salle de réveil, en se basant sur des ordonnancements robustes de l'activité chirurgicale. L'objectif est donc de minimiser le coût d'investissement des salles d'opération et des lits dans la salle de réveil et de minimiser la date de fermeture du bloc opératoire. Cette démarche méthodologique permettra d'accompagner le décideur dans son projet de restructuration

ou de réalisation d'un nouveau bloc opératoire.

Enfin, nous concluons notre thèse en présentant un bilan final de notre travail et en ouvrant quelques perspectives de recherches.

Chapitre 1

Ordonnancement sous incertitudes : état de l'art

Sommaire

Dans la littérature, la majorité des approches en ordonnancement considèrent un environnement totalement déterministe. Néanmoins, en réalité, plusieurs sortes d'aléas peuvent être considérés. L'ordonnancement sous incertitudes permet de tenir compte de ce genre de situations.

Dans ce chapitre, nous présenterons, en premier lieu, un état de l'art sur l'ordonnancement sous incertitudes, où nous définissons les différents types d'incertitudes et les techniques de leurs modélisations. En deuxième lieu, nous proposerons une nouvelle classification des différentes approches d'ordonnancement sous incertitudes. Pour chacune de ces approches, nous présenterons ses avantages et inconvénients, ainsi que les méthodes de résolution développées dans la littérature.

1.1 Introduction à l'ordonnancement

L'ordonnancement est la programmation dans le temps de l'exécution d'une série de tâches (ou activités, opérations) sur un ensemble de ressources physiques (humaines et techniques), cherchant à optimiser certains critères, financiers ou technologiques, et en respectant les contraintes de fabrication et d'organisation (GOThA, 2002; Esquirol et Lopez, 1999).Cette définition de l'ordonnancement ne permet pas de voir le nombre important de problèmes que ce domaine comprend. En effet, l'ordonnancement est lié à plusieurs secteurs de recherches et d'activités très variés. En informatique, le choix des tâches à envoyer aux processeurs se modélise comme un problème d'ordonnancement. En production, l'ordonnancement consiste à déterminer les séquences d'opérations à réaliser sur les différentes machines de l'atelier. En gestion de projet, ordonnancer, c'est déterminer les dates d'exécution des activités constituant le projet. Chacun de ces contextes nécessite la détermination des caractéristiques propres des tâches et des ressources, le type des décisions à prendre, les modalités d'exécution des tâches par les ressources, qui déterminent des contraintes sur les décisions et aussi les différents critères à optimiser.

Dans les dernières années, les problèmes d'ordonnancement ont fait l'objet de nombreuses études. En effet, de nombreuses méthodes de résolution ont été développées pour résoudre des problèmes de plus en plus complexes et de plus en plus proches de la réalité.

Plusieurs auteurs (MacCathy et Liu, 1993; Esswein, 2003) classent les méthodes de résolution en trois catégories : les méthodes optimales efficaces, les méthodes optimales énumératives et les méthodes heuristiques.

Les méthodes optimales efficaces garantissent, pour un problème et un critère donné, la détermination d'une solution optimale en un temps de calcul polynomial. De telles méthodes ne sont évidemment disponibles que pour des classes réduites de problèmes d'ordonnancement. Parmi les plus connues, nous citons l'algorithme de Moore et Hodgson (Moore, 1968), l'algorithme de Johnson (Johnson, 1954), etc.

Les méthodes optimales énumératives procèdent, quant à elles, à une énumération

partielle de l'espace de recherche. Théoriquement, leur complexité temporelle est généralement exponentielle, mais en pratique elles fournissent, sur des problèmes de taille moyenne, des solutions optimales en un temps raisonnable. Dans cette classe, on peut distinguer : les procédures par séparation et évaluation, les méthodes de programmation linéaire et les méthodes basées sur la programmation dynamique.

Les méthodes heuristiques sont utilisées pour traiter des problèmes que les méthodes optimales sont incapables de résoudre en un temps de calcul raisonnable. Elles produisent une solution réalisable de bonne qualité, en un temps de calcul relativement court. Ces heuristiques sont classiquement classées en trois grands types : les méthodes de recherche locale (tabou, recuit simulé, algorithmes génétiques, etc.), les algorithmes gloutons et les méthodes de recherche arborescente tronquée.

La plupart de ces méthodes opèrent sous l'hypothèse classique que les données du problème d'ordonnancement sont parfaitement connues à l'avance. Cette hypothèse permet de traiter un problème de façon déterministe. Alors qu'en réalité, les données du problème sont mal connues et il existe plusieurs sortes d'incertitudes, qui vont perturber le plan de production. Ces incertitudes peuvent être des durées opératoires incertaines, une augmentation ou diminution de la capacité des ressources, une modification de l'ordre des tâches, panne d'une machine, etc.

1.2 Ordonnancement sous incertitudes

Les problèmes classiques d'ordonnancement traitent souvent des problèmes réels, en les simplifiant, puis en incorporant petit à petit des contraintes supplémentaires. De façon similaire, la majorité des approches considèrent un environnement totalement déterministe, dans lequel les aléas n'ont pas leur place. Alors qu'en réalité il existe plusieurs sortes d'aléas, qui vont perturber le plan de production, d'autant plus qu'il est souvent plus réaliste de prendre en considération d'autres fonctionnalités que la production, telles que la maintenance (Ruiz *et al.* (2005a) ; Kenne et Gharbib (2004)). D'autre part, les données du problèmes d'ordonnancement sont assorties de valeurs numériques pour présenter l'état du système réel (temps d'exécution, vitesse d'une machine...). Ces données

numériques peuvent être mal connues ou susceptibles de changer, ce qui est une autre source d'incompatibilité possible du modèle avec la réalité.

La compatibilité du modèle avec la réalité est sans cesse remise en cause par la définition et la modélisation des informations d'entrée du modèle. Ces informations sont parfois qualifiées d'incertaines, incomplètes ou imprécises.

1.3 Définition : incertitude, incomplétude, imprécision

D'une façon générale, on peut dire que les connaissances sur un système réel sont souvent imparfaites. Parmi ces imperfections, Bouchon-Meunier (1995) distingue notamment les incertitudes, les imprécisions et les incomplétudes de ces connaissances.

1.3.1 Incertitude

L'incertitude concerne le doute quant à la validité de la connaissance ou le fait de ne pas savoir si une proposition est vraie ou pas (par exemple : *Je crois, mais ce n'est pas sûr*).

1.3.2 Incomplétude

L'incomplétude porte sur l'absence des connaissances ou sur des connaissances partielles sur certaines caractéristiques du système (par exemple : à x temps la machine y est à l'arrêt s'il n'y a aucune perturbation et qu'aucune tâche d'entretien n'est programmée). Ceci concerne surtout les contraintes externes telles que la demande sur un produit ou les prix de la matière première, qui doivent être pris en considération, lors de l'optimisation des coûts. L'un des exemples les plus frappants sont les hydrocarbures, qui sont soumis à une grande pression des lois du marché alors l'incertitude se fait très bien sentir (Dunne et Mu, 2008).

1.3.3 Imprécision

L'imprécision correspond à une difficulté dans l'énoncé de la connaissance, soit parce que des connaissances numériques sont mal connues, soit parce que des termes du langage

naturel sont utilisés pour qualifier une caractéristique du système de façon vague. Le premier cas est la conséquence d'une insuffisance des instruments d'observation (*2000 à 3000 produits finis*), d'erreurs de mesure (*vitesse d'une machine à 1 % près*) ou encore de connaissances flexibles (*le temps d'exécution d'une tâche est environ entre 8 et 10 minutes*). Le second provient de l'utilisation de catégories aux limites mal définies (*petit, moyen, grand*).

1.3.4 Relation entre incertitude, incomplétude, imprécision

Les formes d'imperfection sont dépendantes. Les incomplétudes entraînent des incertitudes. Les imprécisions peuvent être associées à des incomplétudes et elles engendrent des incertitudes au cours de leur manipulation. Donc, d'une manière générique, nous parlerons d'incertitudes.

De nombreux critères ont par ailleurs été proposés pour classifier de manière plus fine les différents types d'incertitudes. Brautigam *et al.* (2003) distinguent des incertitudes endogènes et exogènes, Artigues *et al.* (2002) ont classifié les incertitudes en fonction de leur niveau de connaissance. Ils identifient trois types d'incertitudes dans les environnements manufacturiers : des incertitudes complètement inconnues, qui sont des évènements imprévus pour lesquels aucune information n'est disponible à l'avance (par exemple, une grève ou un accident dans le lieu de travail ou une absence d'un ouvrier, etc.), des suspicions du futur, qui sont données grâce à l'intuition et l'expérience du décideur et des incertitudes partiellement connues (pannes machines,...). Une autre classification a été présentée par Mehta et Uzsoy (1999) qui ont considéré deux classes d'incertitudes : celles liés aux ressources (changement et/ou manque de ressources, etc.), et celles liés aux travaux (ajout ou retrait d'un travail, etc.).

Bonfill (2006) a présenté un état de l'art sur les incertitudes, où elle a distingué entre les incertitudes stratégiques, tactiques et opérationnelles. Les incertitudes stratégiques sont celles qui affectent principalement la prise de décision à long terme. Ces incertitudes sont, donc, des incertitudes externes ou exogènes venant des conditions environnementales, des changements de technologie et des règlements gouvernementaux. Les incertitudes tactiques peuvent changer la prise de décision à moyen terme. Il s'agit par exemple des paramètres

du marché, des perturbations dans l'information et les flux des matières. Les incertitudes opérationnelles affectent principalement la prise de décision à court terme au sujet des détails. Il s'agit par exemple des durées de la transformation, l'absentéisme de l'opérateur, et la disponibilité d'équipements.

1.4 Modélisation de l'incertitude

La présence des incertitudes rend inévitable la question de leur modélisation. En effet, l'effort de modélisation implique la recherche d'un compromis entre une représentation riche et proche de la réalité, et une représentation intelligible (Bouyssou, 1989). Par conséquent, la modélisation constitue un processus où l'homme d'étude est confronté à des choix arbitraires, également valides, de simplification et d'agrégation. French (1995) souligne que les choix théoriques de l'homme d'étude s'effectuent le plus souvent de manière implicite, ils sont largement influencés par sa formation, ses aptitudes et par les outils qu'il a l'habitude de manipuler.

La littérature offre plusieurs modélisations qui varient tant par leur formalisme que par leurs modes de traitement. Nous présentons, dans les sections suivantes, trois approches de modélisations.

1.4.1 Modélisation stochastique et probabiliste

La modélisation stochastique et probabiliste consiste à modéliser les incertitudes du problème par des variables aléatoires. Or, on est souvent dans l'incapacité de déterminer avec précision la distribution de probabilité appropriée. Ceci nous place dans une situation d'ambiguïté. Cette ambiguïté peut être vue comme de l'imprécision sur les probabilités. C'est une situation intermédiaire entre celle du risque au sens strict (une seule distribution dont les paramètres sont connus) et celle de l'incertitude plus fondamentale, où plusieurs distributions sont envisageables, sans que l'on puisse préciser laquelle est la plus appropriée (Einhorn et Hogarth, 1985). Nous citons, comme exemple, la modélisation de l'incertitude de la demande dans l'industrie pétrolière, où plusieurs distributions ont été testées, par exemple la loi normale (Gupta et Zhang, 2006) et exponentielle (Neiro et Pinto, 2006). Un autre exemple concerne les systèmes hospitaliers. La majorité des auteurs, qui traitent

le problème d'ordonnancement des patients, ont employé des distributions normales pour modéliser la distribution chirurgicale (Gerchak *et al.*, 1996), alors que d'autres ont employé la loi log-normale (Strum *et al.*, 2000; Dexter *et al.*, 1999).

1.4.2 Modélisation floue

Les incertitudes ne sont pas toujours de nature aléatoire. La théorie des sous-ensembles flous se présente comme un outil privilégié pour la modélisation des situations présentant des imprécisions (Zadeh, 1965). C'est l'instrument qui nous permet de représenter la notion de classe dont les limites sont mal définies. Ces variations sont présentées par des fonctions d'appartenance. Ce caractère graduel répond au besoin d'exprimer des connaissances imprécises, telles que des informations recueillies en langage naturel, ou des valeurs approximatives dues à des difficultés de mesure.

1.4.3 Modélisation par scénarios

Cette modélisation nécessite la construction d'un ensemble de scénarios (appelés aussi instances ou jeux de données) contenant des valeurs numériques. Le contenu des scénarios est le reflet d'hypothèses concernant l'incertain. Ces incertitudes peuvent être modélisées par des ensembles discrets ou continus (par intervalles où les bornes sont connues et certaines).

Plusieurs méthodes ont été développées pour incorporer l'incertitude des paramètres au modèle d'une manière quelconque. Ces méthodes incluent l'analyse de sensibilité (Tomovic, 1963) et la simulation de Monte Carlo Hammersley et Handscomb (1964).

Dans la section suivante, nous justifions et présentons une nouvelle classification des différentes approches d'ordonnancement sous incertitudes (Chaari *et al.*, 2009). Chaque approche utilise l'une des modélisations présentées ci-dessus.

1.5 Classification des approches d'ordonnancement sous incertitudes

Dans la littérature, plusieurs classifications d'ordonnancement dans un milieu incertain ont été proposées. Par exemple, celles présentées par Suresh et Chaudhuri (1993), Mehta et Uzsoy (1999), Davenport et Beck (2000) et Herroelen et Leus (2005).

La classification proposée dans Suresh et Chaudhuri (1993) est basée sur les méthodes et les techniques de résolution utilisées dans le domaine de l'ordonnancement réactif (*dynamic scheduling*). En effet, cette classification distingue trois types d'approches : les approches conventionnelles, les approches à base de connaissances et les approches distribuées.

Une autre classification est proposée dans Mehta et Uzsoy (1999). Celle-ci comporte quatre catégories : l'approche totalement réactive, l'approche prédictive-réactive, l'approche robuste et l'approche à base de connaissances. En fait, la catégorie des approches à base de connaissance correspond aux approches dont l'objectif est de fournir un mécanisme pour la sélection dynamique d'une politique de ré-ordonnancement appropriée, parmi un ensemble d'alternatives possibles. On peut donc la voir comme une sous-catégorie particulière des approches réactives. De plus, les approches proactives-réactives ne font pas partie de cette classification.

La classification, proposée dans Davenport et Beck (2000) distingue deux types d'approches : les approches proactives et les approches réactives. Les approches proactives tentent de prendre en compte l'incertain lors de la phase d'ordonnancement hors-ligne uniquement. Les approches réactives prennent en compte l'incertain lors de la phase d'ordonnancement dynamique uniquement. Dans cette classification, les auteurs n'ont considéré ni l'approche proactive-réactive, ni l'approche prédictive-réactive, tandis que dans notre classification, ces deux approches existent et sont désignées par le terme "approches hybrides".

Dans Herroelen et Leus (2005), cinq types d'approches d'ordonnancement sous incerti-

tudes sont distingués : les approches réactives, les approches stochastiques, les approches floues, les approches proactives et les approches basées sur l'analyse de sensibilité. Cette classification est orientée par la nature des outils utilisés pour modéliser l'incertitude, et non par une distinction méthodologique de fond. De plus les travaux présentés dans cette classification se basent plus particulièrement sur les problèmes d'ordonnancement de projet sous incertitudes.

La classification, proposée dans LA (2005) distingue trois types d'approches : les approches réactives, les approches prédictives-réactives et les approches proactives-réactives. Dans cette classification, l'auteur a intégré les approches proactives dans les approches proactives-réactives, alors que, dans notre classification, ces deux approches sont distinguées chacune à part.

D'autres classifications existent, mais non spécifiquement dans le domaine de l'ordonnancement sous incertitudes. Par exemple, Sahinidis (2004) a présenté un état de l'art sur l'optimisation sous incertitudes, qui est un domaine plus large que l'ordonnancement. Il s'est concentré sur la programmation stochastique, programmation floue et programmation dynamique stochastique.

Dans ce chapitre, nous proposons une classification globale, qui permet de couvrir toutes les approches d'ordonnancement sous incertitudes dans plusieurs types de problèmes (ordonnancement de projet, ordonnancement de production, etc.). Cette classification comporte trois approches, à savoir les approches proactives, les approches réactives (dynamiques) et les approches hybrides, qui sont elles-mêmes composées de deux approches, les approches prédictives-réactives et les approches proactives-réactives (Chaari *et al.*, 2009). La figure 1.1 présente, graphiquement, notre typologie proposée. Ce système de classification a l'avantage d'être indépendant du modèle. Les sous-sections suivantes décrivent les diverses catégories dans notre classification. Nous classifions alors un grand nombre de travaux de recherche au sujet de l'ordonnancement sous incertitudes, ce qui nous permettra également de vérifier la complétude de notre typologie.

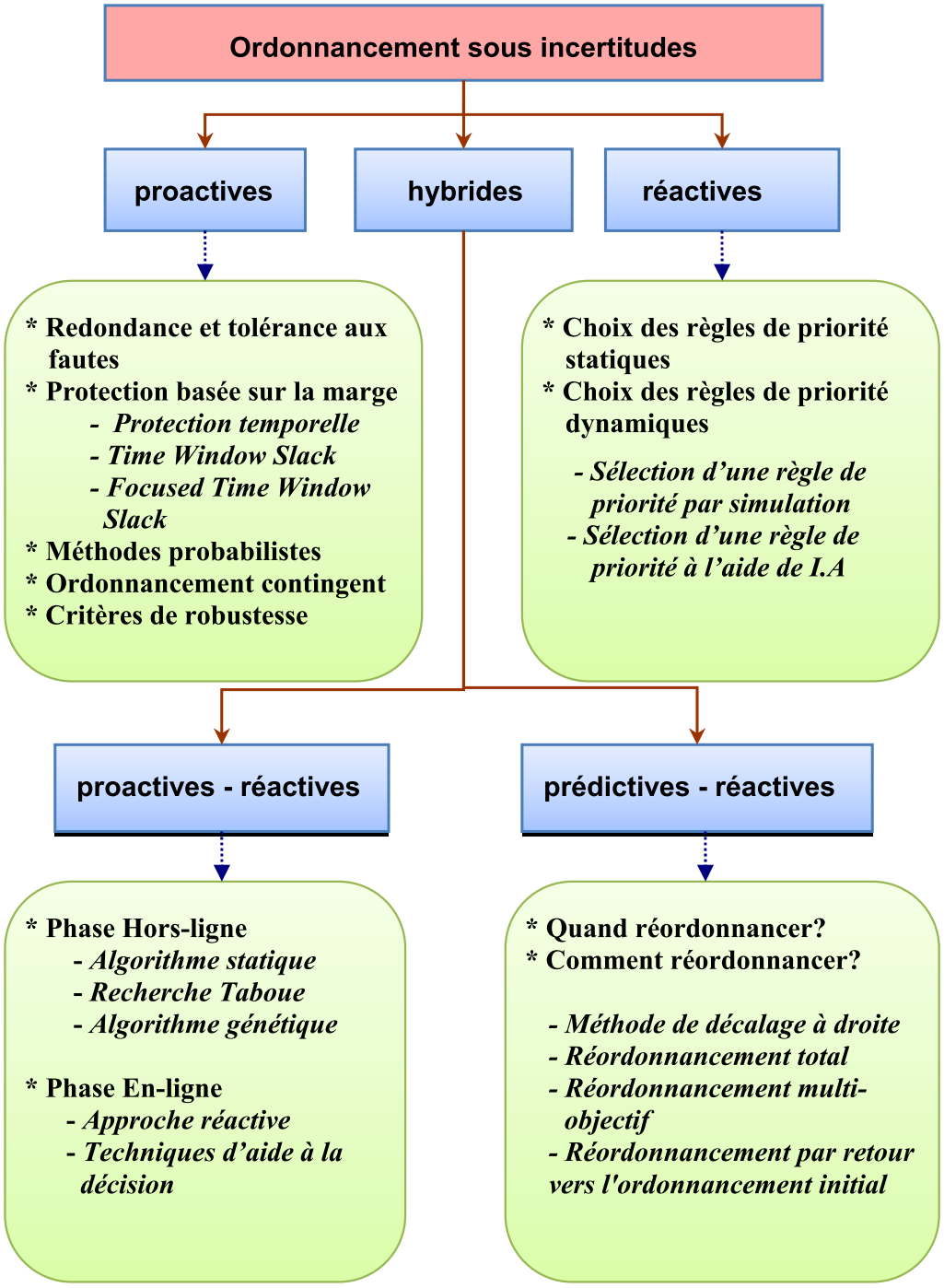

FIG. 1.1 – Représentation graphique de notre typologie sur les approches de l'ordonnancement sous incertitudes

1.5.1 Approches proactives

Les approches proactives (appelées parfois "approches robustes") sont définies dans la littérature dans plusieurs domaines. Dans le domaine des statistiques, Huber (1981) caractérise la robustesse comme une insensibilité à toutes déviations par rapport aux hypothèses. Du point de vue de la conception de système, les auteurs parlent de "stratégie de conception robuste". Ainsi, pour Jichao (1996), il s'agit d'ajuster les paramètres du système de telle sorte que les performances de celui-ci soient relativement insensibles aux sources incontrôlables de variation présentes dans son environnement. En recherche opérationnelle, et plus particulièrement dans le contexte de l'aide à la décision, Roy (1997) propose le concept de conclusion robuste, qui consiste à élaborer des éléments de réponse à un problème auquel est confronté un décideur. Dans la théorie de la décision, l'action la plus robuste est celle qui ne conduit pas à une perte importante quel que soit l'évènement qui va se produire (Pomerol, 2001).

En ordonnancement de production, le terme robuste est très utilisé. Le Pape (1995) définit la robustesse d'un ordonnancement comme sa capacité à satisfaire des exigences de performance d'une façon prévisible. Leon *et al.* (1994) définissent un ordonnancement robuste comme un ordonnancement insensible aux perturbations qui surgissent dans l'atelier, pour une règle de pilotage donnée. Jensen (2000) définit la robustesse comme une qualité de l'ordonnancement supposé rester acceptable si quelques changements imprévus se produisent. Enfin, Davenport et Beck (2000) affirment qu'il n'existe pas de définition d'ordonnancement robuste, mais plutôt une compréhension générale informelle de ce qu'est la robustesse (i.e., la capacité d'un ordonnancement prédictif d'être performant, malgré des évènements inattendus). Il est difficile de donner une définition type de la robustesse en ordonnancement de production, sachant que toutes les définitions citées précédemment sont différentes, mais complémentaires. Nous retenons la définition consensuelle proposée par Billaut *et al.* (2005), *un ordonnancement est robuste si sa performance est peu sensible à l'incertitude des données et aux aléas.*

Les méthodes utilisées pour la résolution de ces approches sont la tolérance aux fautes, la protection basée sur la marge, les techniques probabilistes, l'ordonnancement contingent et les approches qui se basent sur des critères ou des métriques de robustesse.

1.5.1.1 Tolérance aux fautes

La tolérance aux fautes utilise des techniques basées sur la redondance temporelle et/ou de ressources.

⊛ Pour la *redondance temporelle*, des réserves de temps ou insertion des tâches de protection sont maintenues pour masquer les pannes. Cette approche est utilisée par Mehta et Uzsoy (1999) pour un problème à une machine, l'objectif est de minimiser le retard maximum des tâches relativement à un ordonnancement déterministe prédéterminé. La méthode proposée est reprise dans O' Donovan *et al.* (1999) pour le même problème, en prenant comme critère le retard moyen des tâches.

⊛ Pour la *redondance des ressources*, il y a plusieurs moyens possibles pour exécuter une tâche, ce qui permet de mettre des ressources identiques en attente. Ghosh (1996) a fourni une méthode considérant la redondance de ressources en faisant exécuter des copies d'activités sur des ressources supplémentaires.

Pour la majorité des problèmes d'ordonnancement classiques, il est trop coûteux d'utiliser la redondance de ressource. Il est toutefois commun d'utiliser la redondance temporelle, que ce soit en insérant des temps morts entre les tâches ou en augmentant artificiellement la durée des tâches. De la même façon, les temps morts insérés peuvent être spécifiquement dédiés à une activité. Dans certaines industries à risque et à forte exigence, telles que l'industrie pétrolière et surtout les raffineries, les deux types de redondances sont combinées.

1.5.1.2 Protection basée sur la marge

Les méthodes de protection basée sur la marge peuvent être divisées en trois groupes, selon les techniques qu'elles utilisent.

⊛ ***Protection temporelle :***

Chiang et Fox (1990) et Gao *et al.* (1995) ont utilisé les techniques des protections temporelles afin d'augmenter la flexibilité de l'ordonnancement initial. Le problème traité

est de type *job shop* et les incertitudes considérées sont relatives aux pannes de machines, qui peuvent perturber les durées d'exécution des tâches. Ces durées sont augmentées en fonction des durées et des fréquences des pannes. L'ordonnancement est ensuite construit en tenant compte des nouvelles durées. L'inconvénient majeur de cette technique est que la marge ajoutée à chaque tâche ne permet pas de résoudre directement le problème. Cela peut mener à des situations où il est impossible de partager la marge entre plusieurs tâches (Davenport *et al.*, 2001).

⊛ ***Time Window Slack (TWS) :***

La technique *Time Window Slack* (TWS), appelée aussi marge d'intervalle de temps, assurer que chaque tâche aura au moins une quantité spécifique de marge de protection , cette marge étant prise dans une marge générale partagée par toutes les tâches du problème. Cette technique est développée par Davenport *et al.* (2001), qui n'incluent pas la marge de temps dans leurs durées de tâche, mais calculent explicitement la marge de temps disponible par tâche dans la solution de l'ordonnancement.

La quantité de marge de la tâche O_i est :

$$slack_{O_i}(R) \geq \frac{\sum_{O_j \in Acts(R)} dur_{O_j}}{\mu_{tbf}(R)} \mu_{dt}(R) \tag{1.1}$$

$Acts(R)$ est l'ensemble des tâches exécutées sur la ressource R, dur_{O_j} est la durée de la tâche O_j et $\mu_{tbf}(R)$, $\mu_{dt}(R)$ sont respectivement le temps moyen entre deux pannes consécutives sur la ressource R et le temps de panne moyen de la ressource R.

Cette technique traite directement la disponibilité de la tâche et sa marge pendant la résolution du problème, cela est plus intéressant que d'inclure la marge comme une partie de la durée d'exécution d'une tâche. L'avantage de cette méthode est qu'il y a plus d'information concernant la marge pendant la résolution du problème.

⊛ ***Focused Time Window Slack (FTWS) :***

Les techniques de protection temporelle et de TWS, ne tiennent pas compte de l'emplacement de la tâche sur l'axe de l'ordonnancement. La technique FTWS vise à mieux répartir au fil de l'ordonnancement la quantité de marge réservée, en tenant compte des probabilités de pannes. Ainsi, si une tâche est séquencée tard, alors la probabilité qu'une perturbation survienne avant son exécution est plus grande. Donc la tâche a besoin de plus de marge (Davenport *et al.*, 2001).

La marge de temps requise pour une tâche exécutée à l'instant t sur la ressource R est :

$$slack_{O_i}(t,R) \geq \sum_{n=1}^{M} P(N(\mu(n),\sigma(n)) \leq t)\mu_{dt}(R) \tag{1.2}$$

$P(N(\mu(n),\sigma(n)) \leq t)$ désigne la probabilité d'une panne n à l'instant t.
$\mu(n),\sigma(n)$ sont respectivement le temps prévu d'une panne et sa déviation. M est un "très grand" nombre, qui représente le nombre possible de pannes qui peuvent se produire pendant l'exécution.

1.5.1.3 Techniques probabilistes

L'objectif de ces techniques est de construire un ordonnancement tel que la probabilité d'atteindre une certaine performance soit la plus grande possible. Daniels et Carrillo (1997) ont introduit la notion d'ordonnancement $\beta-robuste$, dans le cas d'un problème à une machine avec des durées opératoires incertaines. Le critère considéré est le temps de séjour total des tâches. Ces auteurs proposent une recherche arborescente et une heuristique permettant de construire un ordonnancement maximisant la probabilité d'atteindre un certain niveau de performance.

Wu *et al.* (2009a) ont étudié le même problème où le temps de traitement de chaque activité est caractérisé par une variable aléatoire normalement distribuée, avec le temps de séjour comme objectif principal. L'objectif est de trouver un ordonnancement $\beta-robuste$ qui réduit au minimum le risque du temps de séjour excédant un seuil donné. Ils ont

modélisé le problème comme un problème de satisfaction de contrainte (CSP) et ont effectué sa résolution par une heuristique.

1.5.1.4 Ordonnancement contingent

L'idée principale de l'ordonnancement contingent est de générer des ordonnancements multiples ou un certain nombre d'ordonnancements pour différents résultats possibles. À notre connaissance, le seul exemple trouvé dans la littérature est le *Just-In-Case scheduling* (JIC) développé par Drummond *et al.* (1994). Ces auteurs traitent le domaine de l'ordonnancement d'observation astronomique par télescope où ils utilisent la méthode de JIC pour le résoudre. En effet, la méthode est appliquée à un problème d'ordonnancement à une seule machine, dont la seule ressource est le télescope et les durée des tâches sont incertaines. Chaque observation doit s'exécuter dans un intervalle de temps défini par l'utilisateur.

1.5.1.5 Critères de robustesse

Dans le domaine de l'ordonnancement robuste, les méthodes les plus publiées utilisent des mesures (critères) pour évaluer quantitativement la robustesse des ordonnancements (Sabuncuoglu et Goren, 2009). Les mesures le plus souvent utilisées peuvent être distinguées selon la robustesse caractérisée.

On distingue, néanmoins, deux grandes familles d'approches : celles qui se basent sur l'optimisation d'un critère de robustesse et celles qui imposent des conditions de robustesse que la solution doit satisfaire, pour être considérée comme robuste. Nous présentons une brève description de quelques travaux représentatifs de ces deux familles.

1.5.1.6 Notations

Soient :

Ω un ensemble d'instances du problème ;

I une instance réalisée de Ω, appelée scénario ;

x une solution du problème ;

$z_I(x)$ la fonction d'évaluation de la solution x réalisée sur l'instance I (nous considérons que cette fonction est à minimiser dans la suite de ce chapitre) ;

z_I^* valeur optimale du scénario I ;

m nombre de scénarios ;

λ_j le critère de robustesse.

1.5.1.7 Approches basées sur l'optimisation d'un critère de robustesse

Kouvelis et Yu (1997) introduisent dans leur ouvrage trois critères de robustesse : la robustesse absolue (ou le critère du coût maximal), la déviation robuste (ou le critère du regret maximal) et la robustesse relative (ou le critère du regret relatif) :

- critère du coût maximal (robustesse absolue) : ce critère cherche la fonction objectif maximale pour tous les scénarios ($I \in \Omega$) ;

$$\lambda_1(x, z, \Omega) = \underset{I \in \Omega}{Max}\{z_I(x)\}$$

- critère de regret maximal (déviation robuste) : calcule l'écart entre la fonction objectif du scénario I et la valeur optimale de la solution x sur ce scénario ;

$$\lambda_2(x, z, \Omega) = \underset{I \in \Omega}{Max}\{z_I(x) - z_I^*\}$$

- critère de regret relatif (robustesse relative) : détermine le ratio entre la valeur de la fonction objectif du scénario I et la valeur optimale de la solution x ;

$$\lambda_3(x, z, \Omega) = \underset{I \in \Omega}{Max}\{\frac{z_I(x) - z_I^*}{z_I^*}\}$$

Pour déterminer une solution robuste à partir de l'un des critères définis ci-dessus, il est nécessaire de calculer le minimum. Ces critères se placent dans le pire des cas pour mesurer la robustesse d'une solution. De plus, mesurer la performance pour la pire des instances implique de ne pas la mesurer pour les autres instances.

Sevaux et Sörensen (2004), Sörensen (2001) ont utilisé une fonction d'évaluation robuste, qui est la moyenne de tous les scénarios. Cette fonction est représentée comme suit :

$$\lambda_4(x, z, \Omega) = \frac{1}{m} \sum_{I \in \Omega} z_I(x)$$

L'inconvenient de cette mesure est qu'elle peut conduire à privilégier des ordonnancement parfois très éloignés de la solution optimale pour un scénario donné. De plus, elle ne permet pas de discriminer entre deux solutions qui ont la même valeur d'évaluation.

Jensen (2001) a étudié la qualité des ordonnancements de référence (en anglais *baseline scheduling*) utilisant une mesure de robustesse basée sur le voisinage d'un ordonnancement. La robustesse est évaluée en utilisant d'autres solutions qui peuvent être trouvées dans le voisinage d'une solution initiale. Les voisinages de l'ordonnancement initial sont considérés comme des solutions alternatives, si une perturbation arrive. Un algorithme génétique a été développé pour résoudre ce problème pour un atelier de type *job shop*.

Zuo *et al.* (2009) ont proposé une méthode d'ordonnancement robuste basée sur un voisinage variable multiobjectif, dans lequel l'ordonnancement a été évalué par un critère minimisant simultanément la moyenne et la variance des *makespan* de tous les scénarios. Leur méthode est basée sur un algorithme immunitaire permettant de générer un ensemble de solutions Pareto robustes.

1.5.1.8 Approches basées sur la vérification de la condition de robustesse

Les approches basées sur la vérification de la condition de robustesse permettent au décideur de spécifier ce qu'il attend d'une solution robuste.

⊛ ***p-robustesse :***

Kouvelis *et al.* (1992) ont défini la robustesse comme la capacité d'une solution à garantir une performance qui ne s'éloigne pas de plus de $p\%$ de l'optimal, quelle que soit l'instance de Ω. Les auteurs appellent cette robustesse la p-robustesse :

$$\lambda_3(x, z, \Omega) \leq p\%$$

⊛ ***β-robustesse :***

Klaï et Lamboray (2007) définissent la robustesse comme la capacité d'une solution à garantir une performance qui ne dépasse pas un certain seuil de l'optimum, quelle que soit l'instance de P. Ce seuil est noté β.

$$\lambda_2(x, z, \Omega) \leq \beta$$

Néanmoins, l'approche proactive devient naturellement fortement combinatoire, dès que le nombre de tâches augmente. Si le degré d'incertitude est très élevé, ou si aucun modèle d'incertitude n'est disponible a priori, ou encore si l'environnement de l'ordonnancement est très dynamique, alors une approche réactive peut être plus appropriée. Ce deuxième type d'approche est présenté dans la section suivante (voir figure 1.1).

1.5.2 Approches réactives

Les méthodes d'ordonnancement réactives sont parfois qualifiées de totalement réactives, réactives pures, ou dynamiques. Ces approches réactives sont souvent utilisées dans des environnements fortement perturbés, où les incertitudes sont importantes et ont de fortes amplitudes. Dans un tel environnement, un ordonnancement de référence peut rapidement s'avérer de mauvaise qualité ou même non réalisable. Par conséquent, on suppose qu'il n'existe pas de tel ordonnancement, et que toutes les décisions sont prises en temps réel, en utilisant des stratégies qui privilégient la rapidité des décisions sur leur qualité. Ces approches présentent, en effet, plusieurs avantages, dont le principal réside dans le fait que les prises de décision sont très rapides et intuitivement faciles à comprendre pour les utilisateurs (LA, 2005).

Les méthodes utilisées pour la résolution de ces approches sont généralement de simples règles de priorités, qui consistent, quand une ressource devient disponible, à sélectionner une activité dans la file d'attente selon un certain critère. Ces critères peuvent être optimisés par des interactions avec d'autres centres de prise de décision. Ces interactions peuvent être contrôlées en utilisant une méthode multi-agents, généralement quand la réactivité est exigée.

Les approches multi-agents, ont été largement répondues pour l'ordonnancement réactif sous incertitudes. Querrec *et al.* (1997) ont présenté un modèle générique des systèmes réactifs (voir figure 1.2), dans lequel chaque entité de système est représentée par un agent autonome qui peut prendre des décisions locales. Plusieurs réalisations de cette méthode ont été proposées avec plus ou moins de modifications (Ouelhadj *et al.*, 2005; Albadawi *et al.*, 2006; Reaidy *et al.*, 2006). Pour plus de détails, le lecteur peut se référer à Aissani *et al.* (2008).

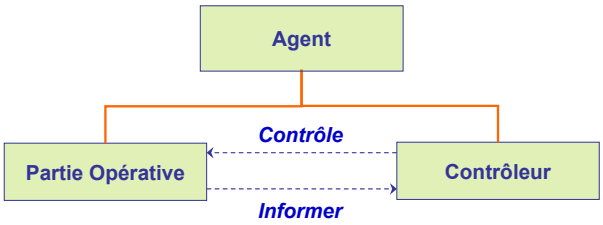

FIG. 1.2 – Modèle conceptuel pour les systèmes réactifs (Querrec *et al.*, 1997)

D'autres études proposent un état de l'art sur le contrôle réactif multi-agents sous incertitudes (par exemple, Baker (1998), ou récemment Shen *et al.* (2006) et Marik et Lazansky (2007)). Pour cette raison, nous avons choisi de ne pas développer cet aspect en détail dans cette thèse.

Dans les méthodes centralisées et distribuées, une des questions les plus importantes est le choix statique ou dynamique des règles de priorité. Les sous-sections suivantes sont consacrées à ce thème.

1.5.2.1 Typologie des règles de priorité

Un état de l'art pour les différentes règles de priorité a été présenté par Rajendran et Holthaus (1999). En général, les règles de priorité peuvent varier selon la fonction objectif considérée (minimisation du *makespan*, du retard,...). Ces règles peuvent être classifiées de plusieurs manières. Nous distinguons tout d'abord les règles statiques (le cas de et les règles dynamiques.

Les règles statiques sont liées aux données associées aux tâches ou aux ressources, elles sont indépendantes du temps (par exemple la règle WSPT - *Weighted Shortest Processing*

Time first). Les règles dynamiques sont dépendantes du temps par exemple la règle MS (*Minimum Slack first*). Une autre distinction concerne le caractère local ou global d'une règle. Une règle locale traite seulement une sous-partie des informations disponibles, par exemple la file d'attente des tâches en attente de libération d'une ressource. Alors qu'une règle globale exploite toutes les informations disponibles.

Indépendamment du type de règles de priorité (statiques ou dynamiques, locales ou globales), l'ensemble des règles de priorité peut toujours être modifié. La section suivante décrit ces modifications.

1.5.2.2 Choix des règles de priorité

Une extension des méthodes d'ordonnancement par règles consiste à permettre au système de choisir ou de modifier dynamiquement, selon le contexte, la règle à utiliser. Selon Priore *et al.* (1998), cette extension permet d'améliorer considérablement la performance du système. La sélection d'une règle peut être réalisée soit à l'aide de simulation, soit à l'aide des modèles de connaissances.

Deux techniques ont été présentées pour la sélection des règles de priorité, la simulation et l'intelligence artificielle (I.A).

⊛ ***Sélection d'une règle de priorité par simulation :***

L'idée de cette méthode consiste, au moment d'une prise de décision, à simuler un ensemble de règles de priorité et sélectionner celle fournissant la meilleure performance, Lopez et Roubellat (2008).

Sabuncuoglu (1998) a présenté une étude complète de simulation sur des règles d'ordonnancement pour les systèmes flexibles de fabrication, en présence de divers niveaux de pannes. Kutanoglu et Sabuncuoglu (1999) ont présenté une étude comparative complète de plus de vingt règles de priorité dans un atelier de type *job shop* avec un critère du retard pondéré, où les *jobs* arrivent dynamiquement.

Kim et Kim (1994) ont proposé un système d'ordonnancement basé sur la simulation avec deux composantes principales : un mécanisme de simulation et un mécanisme réactif de contrôle. Le mécanisme de simulation évalue diverses règles et choisit la meilleure. Le mécanisme réactif de contrôle surveille périodiquement l'exploitation du système et un nouvel ordonnancement est produit au début de chaque période où il y a des perturbations importantes dans le système.

Jeong et Kim (1998) ont utilisé la simulation et des règles de priorité pour l'ordonnancement en temps réel dans un atelier flexible où les travaux arrivent dynamiquement. La simulation a été utilisée pour évaluer les règles de priorité. En se basant sur cette évaluation, l'ordonnanceur choisit la meilleure règle qui satisfait les critères demandés.

Pour éviter d'effectuer beaucoup de simulations pour trouver les meilleures règles de priorité, des méthodes statistiques et d'apprentissages ont été présentées. Parmi ces méthodes, on trouve les méthodes de classifications Chu et Portmann (1993), les méthodes de segmentation Boucon (1991) et l'apprentissage automatique Pierreval et Ralambondrainy (1990).

⊛ ***Sélection d'une règle de priorité à l'aide de l'I.A :***

Dans ces méthodes, des techniques issues de l'Intelligence Artificielle (I.A) sont utilisées. Le modèle de connaissances est construit hors ligne lors d'une phase d'apprentissage. La principale difficulté consiste à calibrer ce modèle, de sorte qu'il soit suffisamment intelligent pour déterminer en ligne quelle règle, ou quelle décision, est la plus appropriée.

Priore *et al.* (1998) présentent une construction hors-ligne d'un modèle à base de connaissance, fondée sur l'exécution itérative de cinq étapes : la définition des paramètres de contrôle du modèle de connaissance, la génération des exemples d'apprentissage, la définition des conditions d'activation d'une règle (une condition d'activation est vraie si la valeur courante des paramètres de contrôle appartient à un domaine prédéfini), la sélection de la meilleure règle parmi celles activées et la vérification de la performance. Tant que la performance n'est pas satisfaisante, les étapes sont réitérées en introduisant des

changements, soit dans les paramètres de contrôle, soit dans les exemples d'apprentissage, soit encore sur les domaines mis en jeu dans les conditions d'activation.

Pierreval (1992) a proposé un système expert pour sélectionner dynamiquement les règles de priorité selon l'état de système et les conditions de fonctionnement. Cette approche a été illustrée par un exemple de type *flow shop* simplifié. Les résultats obtenus montrent qu'un système expert peut obtenir une meilleure performance qu'une combinaison de règles de priorité.

d'autres méthodes à base de connaissance sont présentées dans Chen et Yih (1996), Pierreval (1993) et Mouelhi et Pierreval (2010), où les auteurs focalisent leur étude sur la détermination des paramètres pertinents pour la sélection des règles, en utilisant une approche neuronale.

1.5.3 Approches hybrides

Les approches hybrides sont subdivisées en deux approches : les approches prédictives-réactives et les approches proactives-réactives (Figure 1.1).

1.5.3.1 Approches prédictives-réactives

Les approches prédictives-réactives sont souvent référencées dans la littérature pour faire face aux aléas. Ces approches sont constituées de deux phases. Dans la première phase, un ordonnancement déterministe est construit hors-ligne (c'est-à-dire que l'on considère que les évènements prévisibles. Par exemple, les *jobs* qui doivent être prévus sont tous disponibles dès le départ, les *processing times* sont déterminés et les machines et d'autres ressources sont disponibles durant l'horizon de l'ordonnancement). Durant la deuxième phase, cet ordonnancement est ensuite utilisé, en ligne, pour guider les décisions. Ces décisions sont alors adaptées en temps réel, pour tenir compte des perturbations.

La résolution de ces approches se base sur les réponses aux deux questions : *Quand réordonnancer ?* et *Comment réordonnancer ?*

⊛ ***Quand réordonnancer ?***

Plusieurs cas de réordonnancement ont été proposés dans la littérature, dont les principaux sont :

– **Réordonnancer à chaque fois qu'un évènement inattendu apparaît**. C'est le cas dans Wu *et al.* (1993), où les auteurs proposent une heuristique pour réordonnancer un problème à une machine en cherchant à maximiser la stabilité. La stabilité d'un ordonnancement est référée à la distance entre l'ordonnancement de référence et l'ordonnancement réalisé pour un scénario donné. La distance peut être mesurée par le nombre d'activités perturbées, la différence entre la date de début de l'activité planifiée et la date de début de l'activité réalisée, etc.

– **Réordonnancer à chaque fois qu'un nouveau job arrive**. C'est le cas dans Bierwirth et Mattfeld (1999), où les auteurs présentent un algorithme génétique réutilisant la solution courante pour résoudre un problème de j*ob shop* chaque fois qu'un nouveau *job* arrive. De même, Artigues et Roubellat (2000) ont proposé un algorithme polynomial d'insertion de nouveaux *jobs* (activités) dans un ordonnancement prédictif. Cet algorithme a été appliqué pour résoudre le problème d'ordonnancement de projet multi-modes multi-resources avec des ressources cumulatives.

– **Réordonnancer à des intervalles de temps réguliers**. Cette approche est utilisée dans les travaux de Church et Uzsoy (1992). Les auteurs ont développé une politique de réordonnancement hybride pour des problèmes à une seule machine et machines parallèles avec des arrivées dynamiques de *jobs*. Ils classifient les évènements qui changent l'état du système en deux catégories : les évènements qui exigent la réponse immédiate (exceptions) et les évènements qui peuvent être ignorés jusqu'au réordonnancement suivant. L'ordonnancement est révisé périodiquement et déclenché quand une exception arrive. Dans leur modèle, les exceptions sont les arrivées des *jobs* avec des dates de livraisons serrées. Pour chaque *job* i arrivant entre le temps $(k-1)T$ et T, ils calculent la marge $s_i = d_i - r_i$, où d_i est la date de livraison, r_i est la date de disponibilité, T est la longueur de la période et k un

entier. Si cette valeur est plus petite qu'une constante w (intervalle de temps), alors l'arrivée de *job* est considérée comme une exception et la décision de l'ordonnancement est déclenchée. Les auteurs utilisent la règle *Earliest Due Date* pour générer des ordonnancements à chaque itération. La mesure de performance utilisée est le retard algébrique maximal.

– **Réordonnancer dès que le nombre d'arrivées de jobs atteint un seuil donné**. C'est le cas dans Vieira *et al.* (2000), où les auteurs étudient le problème à une machine dans lequel les activités arrivent dynamiquement. Ils montrent que réordonnancer fréquemment permet d'obtenir de meilleures performances du système, mais entraîne une augmentation du nombre de réglages, tandis que des réordonnancements moins fréquents augmentent les temps de cycles.

⊛ ***Comment réordonnancer ?***

Pour cette question, quatre réponses sont décrites :

– **Méthode de décalage à droite** : consiste à retarder l'exécution des tâches affectées par une panne en maintenant la même séquence de départ. Cette méthode permet de maintenir la stabilité de l'atelier (Smith, 1994; Sadeh *et al.*, 1993). Par exemple, dans le diagramme de Gantt représenté sur la figure 1.3, si la machine M_1 est tombée en panne lorsque le *job* 2 est en exécution et le temps de réparation de cette machine est égal à r unités de temps, alors la période de décalage du *job* 2 (sur la machine M_1) sera retardée de l'instant t à $t + r$. De plus le temps d'achèvement des tâches sur les machines M_1 et M_2 sera décalé aussi de r unités de temps.

– **Réordonnancement total :** consiste à décomposer le problème en sous-problèmes qui sont résolus en séquence. Cette technique a été proposée par Snoek (2001). Cette auteur a traité le problème de type *job shop* dont les incertitudes concernent l'arrivée de nouveaux *jobs*. L'objectif considéré est la minimisation du retard moyen. Le réordonnancement est exécuté à chaque arrivée de nouveaux *jobs*. Afin de réduire la complexité du temps de calcul, les auteurs ont décomposé le problème en n sous-

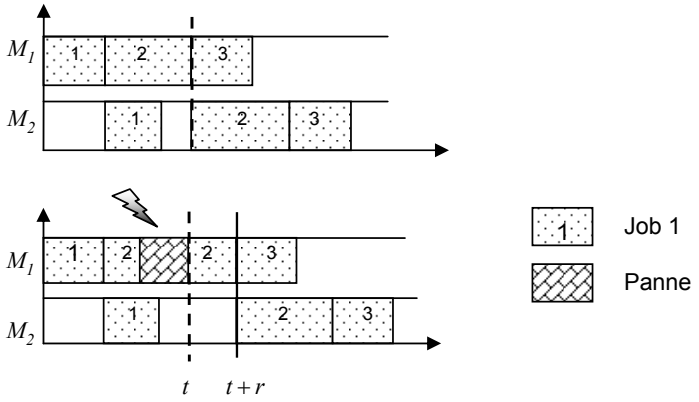

FIG. 1.3 – Exemple de la méthode de décalage à droite

problèmes, qu'ils résolvent en séquence. Un algorithme génétique est développé pour résoudre chaque sous-problème. Une marge disponible pour chaque sous-problème est réalisée, pour faciliter l'intégration des jobs dans la file d'attente.

– **Réordonnancement multiobjectif :** est basé sur une fonction bi-objectif qui prend en considération des critères d'efficacité et de stabilité simultanément. Cette fonction bi-objectif est proposée par Wu *et al.* (1993) pour minimiser le *makespan* et la déviation d'un ordonnancement initial (D). Cette déviation est mesurée par la différence entre la date de début et/ou l'ordre des *jobs* dans l'ordonnancement initial et l'ordonnancement corrigé. Pour calculer le nouvel ordonnancement, ces auteurs appliquent la fonction bi-objectif suivante :

$$Z = r \times M + (1 - r) \times D \tag{1.3}$$

M est la mesure de l'efficacité et $r \in [0, 1]$ est un facteur de pondération.
Dans certains cas, l'efficacité ou la stabilité elle-même est une combinaison des mesures relatives (Rangsaritratsamee *et al.*, 2004).

Masuchun et Ferrell (2004) ont développé une fonction multiobjectif qui s'adresse simultanément à l'efficacité et la stabilité. L'efficacité est définie en terme de *makespan* et de retard, alors que la stabilité est définie comme combinaison linéaire de la déviation entre la date de début initiale du *job* et la fonction de pénalité liée à la dé-

viation totale du temps en cours. Une stratégie de réordonnancement périodique est utilisée pour déterminer la date de début du nouvel ordonnancement. Un algorithme génétique est employé pour obtenir un bon ordonnancement pour le problème de job shop dans un environnement dynamique. Dans un autre travail, Masuchun et Masuchun (2004) ont présenté le concept de l'intervalle bloqué (en anglais *frozen interval*) pour bloquer les *jobs*. L'objectif de cet intervalle est double, soit d'augmenter le nombre de *jobs* pour que l'ordonnancement obtenu reste inchangé, soit de réduire le nombre de *jobs* et dans ce cas l'ordonnancement peut être changé. Les résultats d'une simulation et d'une analyse statistique prouvent que l'utilisation de l'intervalle bloqué dans les problèmes d'ordonnancement permet d'améliorer de manière significative la performance de l'ordonnancement.

– **Réordonnancement par retour vers l'ordonnancement initial :** consiste à modifier les décisions de l'ordonnancement de référence, sur une période de temps, la plus réduite possible, de sorte que cet ordonnancement soit réutilisable à la fin de la période. Cette technique a été utilisée par Bean *et al.* (1991) et Akturk et Gorgulu (1999). Bean *et al.* (1991) ont proposé une procédure permettant de réparer l'ordonnancement prédictif lorsque des perturbations surviennent, de telle sorte que le nouvel ordonnancement recouvre autant que possible l'ordonnancement initial. Akturk et Gorgulu (1999)ont appliqué cette technique au problème de *flow shop* modifié soumis à des pannes de machines. Les résultats ont montré que cette technique est efficace en termes de temps d'exécution, de qualité des ordonnancements et de stabilité des solutions.

1.5.3.2 Approches proactives-réactives

D'après Elkhyari (2003), ces approches sont constituées de deux phases. Dans la première phase, un ordonnancement robuste tenant compte de l'aspect incertain de certaines données est construit par un algorithme statique ou hors-ligne. Dans la deuxième phase, cet ordonnancement est adapté en-ligne en fonction de l'état du système au moment de son exécution, grâce à un algorithme dynamique. La différence entre les approches prédictives-réactives et les approches proactives-réactives est principalement due au fait que, dans les approches proactives-réactives, aucun réordonnancement n'est fait en-ligne ; une solution

d'ordonnancement évaluée est choisie parmi un ensemble de solutions. La plupart des méthodes de résolution des approches proactives-réactives permettent de construire un ensemble d'ordonnancements statiques tels qu'il soit facile de passer de l'un à l'autre, en cas de perturbation.

L'*ORdonnancement d'Atelier Basé sur l'AIde à la Décision* (ORABAID) développé au LAAS-CNRS (Laboratoire d'Analyse et d'Architecture des Systèmes) est l'une des approches proactives-réactives. L'idée générale de cette méthode est de construire, durant la phase hors-ligne, un ordonnancement partiel en déterminant, pour chaque ressource, une séquence de groupes de tâches, les tâches d'un même groupe étant totalement permutables (Roubellat *et al.*, 1998). La méthode utilise une heuristique qui construit une séquence de groupes initiale, puis une méthode de recherche tabou est utilisée pour améliorer les performances. Cet ordonnancement est ensuite exploité en temps réel. Le séquencement des tâches d'un même groupe est défini, au fur et à mesure de l'exécution de l'ordonnancement, par un décideur, qui dispose d'indicateurs, quant à la qualité de chaque choix vis-à-vis de la performance. ORABAID propose des actions correctrices permettant de modifier l'ordre et/ou l'affectation de certaines activités. Enfin, ORABAID permet également l'ajout en temps réel de nouvelles opérations, ce qui fournit un élément de flexibilité supplémentaire.

Wu *et al.* (2009b) ont présenté une nouvelle technique mixte, qui combine l'approche proactive avec l'approche réactive pour traiter le problème d'ordonnancement sous incertitudes. Dans la phase proactive, ils ont construit un ordonnancement de référence robuste, qui minimise la somme des déviations en valeur absolue entre l'ordonnancement de référence et l'ordonnancement attendu. L'ordonnancement de référence robuste contient une flexibilité pour minimiser les procédures de recherche complexes pour l'approche réactive de l'ordonnancement. Les auteurs ont alors développé une procédure réactive d'ordonnancement pour réviser rapidement l'ordonnancement perturbé, quand des évènements inattendu apparaissent. Elle est basée sur des informations industrielles en temps réel.

Aloulou et Portmann (2002) proposent un algorithme génétique construisant plusieurs ordonnancements équivalents et garantissant un bon compromis entre flexibilité et performance. Ensuite, à chaque fois qu'une décision doit être prise suite à un aléa (par exemple :

retards de livraison de matières premières), plusieurs solutions sont proposées au décideur.

1.6 Conclusion

Nous avons présenté dans ce chapitre les différents types d'incertitudes et les techniques de leur modélisation. Une nouvelle classification des différentes approches de l'ordonnancement dans un milieu incertain a été développée. Cette classification comporte trois approches, à savoir les approches proactives, les approches réactives et les approches hybrides. Ces dernières sont composées de deux approches, les approches prédictives-réactives et les approches proactives-réactives. Pour chacune de ces approches, un état de l'art des techniques et des méthodes de résolution, ainsi que des avantages et des inconvénients, a été présenté.

Les problèmes d'ordonnancement sont reconnus, dans la littérature, comme des problèmes NP-difficiles. La présence des incertitudes rend ces problèmes plus compliqués. Afin de déterminer des solutions de bonne qualité en un temps de calcul raisonnable, les approches développées, dans la littérature, pour la résolution de ces types de problèmes, sont des approches basées sur les métaheuristiques, à savoir la recherche tabou, les algorithmes génétiques, le recuit simulé, etc.

Le développement d'un ordonnancement robuste dont la performance est peu sensible aux incertitudes sera l'objectif du chapitre suivant. L'application d'une approche basée sur les métaheuristiques permet de déterminer une solution robuste, de bonne qualité, et en temps de calcul raisonnable. Du fait de la performance prouvée, dans la littérature, de l'algorithme génétique pour la résolution des problèmes d'ordonnancement, nous choisissons cet algorithme pour la résolution d'un problème d'ordonnancement robuste sous incertitudes. Une organisation de type *flow shop* hybride constitue notre cas d'étude. Le choix de ce cas d'étude est basé sur le fait que ce problème peut être assimilé au problème des interventions au sein d'un bloc opératoire à plusieurs étages, où les étages représentent les salles d'opérations, les salles de réveil, etc.

Chapitre 2

Algorithme génétique pour l'ordonnancement robuste : cas du *flow shop* hybride

Sommaire

2.1 Introduction

D'autres travaux sont actuellement menés dans l'équipe sur les ordonnancements sous incertitudes. Ils sont réalisés dans le cadre d'une approche réactive (Aissani *et al.*, 2009). C'est pourquoi nous avons choisi, dans le cadre de cette thèse, de nous focaliser sur une

approche complémentaire, l'approche proactive.

L'approche proactive suppose une connaissance au moins partielle des incertitudes sur les données. Par ailleurs, nous constatons que les industriels sont incapables de fournir des données fiables ou satisfaisantes au problème posé. Nous proposerons une solution robuste de bonne performance et peu sensible aux incertitudes. La recherche de cette solution en temps de calcul raisonnable et de bonne qualité revient à utiliser les métaheuristiques. Ces techniques constituent un ensemble de règles et de mécanismes généraux ayant comme fonction de contrôler et de guider la recherche d'une bonne solution. Elles sont appliquées avec succès pour des problèmes très variés, tels que les problèmes du voyageur de commerce, d'ordonnancement, de localisation, de transport, de partition et de coloration de graphes. Parmi les techniques les plus employées, on trouve les algorithmes génétiques qui présentent des solutions de qualités intéressantes pour la résolution des problèmes combinatoires complexes, leurs applications en ordonnancement sous incertitudes sont peu répondues.

Dans ce chapitre, nous nous intéressons à un problème connu dans l'industrie, le problème du *flow shop* hybride. Les incertitudes portent sur les durées d'exécution des travaux (*jobs*) sur les machines. La modélisation de ces incertitudes est une modélisation par scénarios, au sens du chapitre 1. L'objectif est donc de proposer un algorithme génétique permettant d'obtenir une solution robuste de bonne performance, qui est peu sensible à ces incertitudes. A cet effet, il est nécessaire de développer un algorithme efficace pour avoir un ordonnancement robuste. Avant de présenter notre algorithme génétique pour l'ordonnancement robuste, nous développons un algorithme génétique pour la résolution du problème d'ordonnancement du *flow shop* hybride dans un environnement déterministe. Cet algorithme consiste à minimiser seulement le *makespan*, sans prendre en compte les perturbations sur les temps d'exécution des *jobs* sur les machines. L'objectif de développement de cet algorithme est de vérifier son efficacité dans le cas déterministe. Puis, nous décrivons comment l'algorithme génétique pour l'ordonnancement déterministe peut être modifié pour trouver des solutions qui sont non seulement efficaces (*makespan* minimum) mais aussi robustes faces aux incertitudes. La recherche de cette solution robuste et efficace nous permettra de développer un nouveau mécanisme intégré dans l'algorithme génétique

pour l'ordonnancement robuste et de développer un nouveau critère de robustesse. Le choix de type d'organisation *flow shop* hybride pour illustrer nos travaux a été fait par rapport à une de nos applications réelles privilégiées, qui est le bloc opératoire. En effet, l'organisation d'un bloc opératoire, comme nous allons le montrer plus loin, correspond à une organisation de type *flow shop* hybride.

Après une description générale du principe de l'algorithme génétique dans la section 2.2, la section 2.3 présentera les différentes phases de cet algorithme. La section 2.4 présentera une description du problème du *flow shop* hybride, ainsi que les méthodes de résolution. La section 2.5 présentera une mise en oeuvre d'un algorithme génétique déterministe. La section 2.6 sera consacrée au développement d'un algorithme génétique pour l'ordonnancement robuste.

2.2 Concepts de base d'un algorithme génétique

Les algorithmes génétiques font partie de la famille des algorithmes évolutionnaires. Ils s'inspirent de l'évolution naturelle des espèces. Avec ce type de méthodes, il ne s'agit pas de trouver une solution analytique exacte mais de trouver une bonne solution satisfaisante dans un temps de calcul raisonnable. La première description du processus des algorithmes génétiques a été donnée par Holland en 1975, puis Goldberg (1989) les a utilisés pour résoudre des problèmes concrets d'optimisation.

Le but de ces algorithmes génétiques est d'optimiser une fonction prédéfinie, appelée fonction objectif, ou *fitness* ; ils travaillent sur un ensemble de solutions candidates, appelé "population" d'individus ou chromosomes (on utilisera indifféremment individu ou chromosome). Ces derniers sont constitués d'un ensemble d'éléments, appelés "gènes", qui peuvent prendre plusieurs valeurs, appelées "allèles".

Un chromosome est une représentation ou un codage d'une solution du problème donné. Une première population est choisie soit aléatoirement, soit par des heuristiques ou par des méthodes spécifiques au problème, soit encore par mélange de solutions aléatoires et heuristiques. Cette population doit être suffisamment diversifiée pour que l'algorithme

ne reste pas bloqué dans un optimum local. C'est ce qui se produit lorsque trop d'individus sont semblables. Les algorithmes génétiques génèrent de nouveaux individus, de telle sorte qu'ils soient plus performants que leurs prédécesseurs. Le processus d'amélioration des individus s'effectue par utilisation d'opérateurs génétiques, qui sont la sélection, le croisement et la mutation.

Pour mettre en oeuvre un algorithme génétique, il est nécessaire de disposer :

- une représentation génétique du problème, c'est-à-dire un codage de solutions utilisé sous la forme de chromosomes.

- un mécanisme de génération de la population initiale. Ce mécanisme est indispensable de construire une population d'individus non homogène.

- une fonction qui permet d'évaluer l'adaptation d'un chromosome à son environnement, ce qui offre la possibilité de comparer des individus. Cette fonction est construite à partir du critère que l'on désire optimiser. L'application de cette fonction à un élément de la population donne sa *fitness* ;

- un mode de sélection des chromosomes à reproduire. Cette sélection est basée sur la reproduction et sur le codage génétique, qui stocke les informations décrivant l'individu sous forme de gènes ;

- d'opérateurs de croisement et de mutation permettant de diversifier la population au cours des générations et d'explorer l'espace d'état.

- de paramètres qu'utilise l'algorithme : taille de la population, probabilité de croisement et de mutation, nombre total de générations.

2.3 Les étapes de l'algorithme génétique

L'algorithme génétique commence par une génération d'une population initiale de P_{size} individus, pour lesquels nous calculons leurs *fitness* et nous sélectionnons les individus par une méthode de sélection. Ces individus seront manipulés par un opérateur de croisement qui les choisit selon une probabilité P_{cross}. Leurs résultats peuvent être mutés par un opérateur de mutation avec une probabilité de mutation P_{mut}. Les phases de selection et de recombinaison (croisement et mutation) permettent de générer une nouvelle population d'individus, qui ont de bonne chances d'être plus forts que ceux de la génération précédente. Les individus issus de la phase de recombinaison seront insérés par une méthode d'insertion dans la nouvelle population, dont nous évaluons la valeur de la fonction objectif de chacun de ses individus. De génération en génération, la force des individus de la population augmente et un test d'arrêt sera effectué pout décider quand arrêter l'algorithme. La figure 2.1 présente un schéma de fonctionnement général de l'algorithme génétique. Les différentes étapes de ce dernier sont présentées, en détail, dans les sections suivantes.

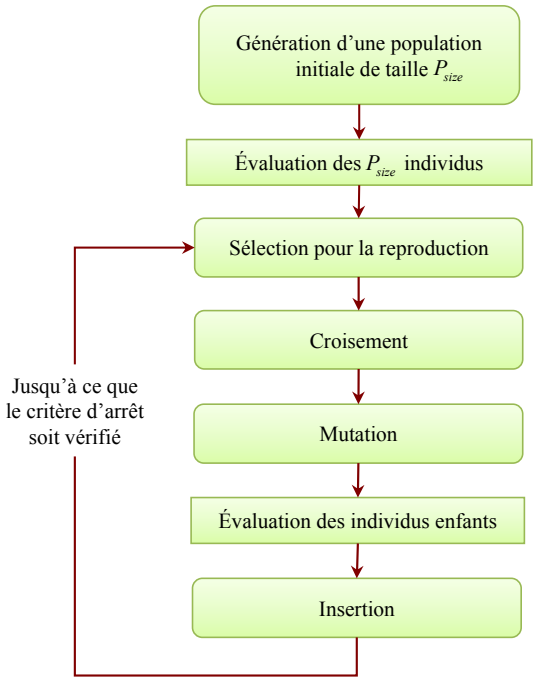

FIG. 2.1 – Fonctionnement général de l'algorithme génétique

2.3.1 Codage

Le premier pas dans l'implantation des algorithmes génétiques est de créer une population d'individus initiaux. Chaque individu de la population est codé par un chromosome. Une population est donc un ensemble de chromosomes. Chaque chromosome code un point de l'espace de recherche. L'efficacité de l'algorithme génétique va donc dépendre du choix du codage d'un chromosome.

Deux types de codages ont été présentés dans la littérature : le codage direct et le codage indirect (Guillaume *et al.*, 2001). Si le chromosome représente implicitement la solution, le codage est direct. Par contre, si le chromosome contient un ensemble d'informations spécifiques à un problème donné et la solution du problème est obtenue après une transformation, le codage est indirect.

Un chromosome est représenté sous forme de chaînes de bits 0-1. Ce codage binaire a permis de résoudre beaucoup de problèmes, mais il s'est avéré que, pour des problèmes d'ordonnancement, il est plus pratique d'utiliser un codage entier des chromosomes. Ce codage permet d'augmenter l'efficacité de l'algorithme génétique.

2.3.2 Génération de la population initiale

Plusieurs mécanismes de génération de la population initiale sont utilisés dans la littérature (Caux *et al.*, 1995). Le problème principal dans cette étape est le choix de la taille de la population. Si la taille de la population est trop grande, le temps de calcul augmente et demande un espace mémoire important. Par contre, une population de taille très petite, la solution obtenue n'est pas satisfaisante. Il faut donc trouver le bon compromis.

2.3.3 Evaluation : *fitness*

Une fonction d'évaluation est utilisée pour mesurer les performances de chaque individu, qui correspond à une solution donnée du problème à résoudre. Cette fonction permet d'évaluer la capacité d'un individu à survivre en lui affectant un poids appelé *fitness*. La force de chaque chromosome de la population est calculée afin que les plus forts soient retenus dans la phase de sélection, puis modifiés dans la phase de croisement et mutation.

La fonction d'évaluation dépend d'un problème à un autre. Par exemple, dans le cas des problèmes d'ordonnancement, cette fonction peut prendre différents critères (*makespan*, retard, etc.)

2.3.4 Sélection

La sélection permet d'identifier les individus susceptibles d'être croisés dans une population. Il existe plusieurs techniques de sélection (Dréo *et al.*, 2003). Nous présentons ici les quatre les plus utilisées parmi elles :

- sélection par rang : consiste à ranger les individus de la population dans un ordre croissant ou décroissant, selon l'objectif (fonction *fitness*) ;

- sélection par roulette : elle consiste à créer une roue de loterie biaisée pour laquelle chaque individu de la population occupe une sélection de la roue proportionnelle à sa valeur d'évaluation ;

- sélection aléatoire : cette sélection se fait aléatoirement, uniformément et sans intervention de la valeur d'adaptation. Chaque individu a donc une probabilité uniforme $\frac{1}{P_{size}}$ d'être sélectionné, où P_{size} est le nombre total d'individus dans la population ;

- sélection par tournoi : le tournoi le plus simple consiste à choisir aléatoirement un nombre k d'individus dans la population et à sélectionner celui qui a la meilleur performance. Les individus qui participent à un tournoi sont remis ou sont retirés de la population, selon le choix de l'utilisateur. Avec le tournoi binaire, sur deux individus en compétition, le meilleur gagne avec une probabilité $p \in [0,5; 1]$;

2.3.5 Croisement

Le croisement permet d'enrichir la population en manipulant les composantes des chromosomes. Un croisement est envisagé avec deux parents et génère un ou deux enfants. Il est appliqué avec une probabilité P_{cross}, appelée probabilité de croisement. Après la sélection de deux individus, nous générons un nombre aléatoire $\alpha \in [0, 1]$. Si $\alpha \leq P_{cross}$, nous

appliquons l'opérateur de croisement sur les deux parents. Selon la littérature, plusieurs opérateurs de croisement sont proposés (Nearchou, 2004), nous citons ici les plus utilisés :

– **Croisement en 1-point :** consiste à diviser chacun des deux parents en deux parties à la même position, choisie au hasard et à recopier la partie inférieure du parent à l'enfant et à compléter les gènes manquants de l'enfant à partir de l'autre parent en maintenant l'ordre des gènes. La figure 2.2 présente un exemple illustratif de ce type de croisement.

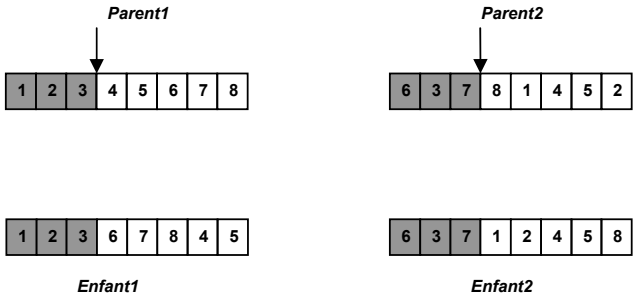

FIG. 2.2 – Croisement en 1-point de deux chromosomes

– **Croisement en 2-points** : Ce type de croisement est utilisé en choisissant aléatoirement 2 points de coupure pour dissocier chaque parent en 3 fragments. Les 2 fragments en extrémités pour le *Parent1* (respectivement *Parent2*) sont copiés à l'*Enfant1* (respectivement *Enfant2*). On complète la partie restante de l'*Enfant1* par les éléments du *Parent2* et la partie restante de l'*Enfant2* par les éléments du *Parent1* en balayant de gauche à droite et en ne reprenant que les éléments non encore transmis. La figure 2.3 présente un exemple illustratif de ce type de croisement.

D'autres opérateurs de croisement sont proposés dans la littérature comme le croisement PMX(*partial-mapped crossover*), OX (*order crossover*), CX (*cycle crossover*), JOX (*job-based order crossover*), ER (*edge recombination crossover*), etc.

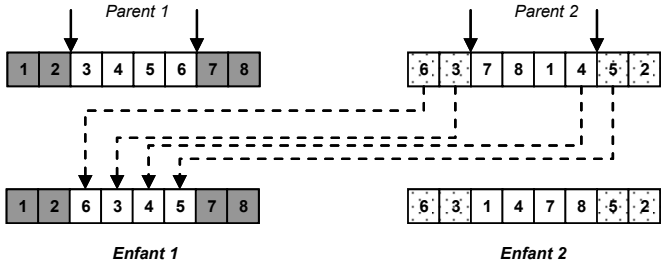

FIG. 2.3 – Croisement en 2-points de deux chromosomes

2.3.6 Mutation

La mutation apporte l'aléa nécessaire à une exploration efficace de l'espace. Elle permet de quitter les extremas locaux. Cet opérateur de mutation est utilisé avec une probabilité P_{mut}. Si β, généré aléatoirement, appartient à $[0, P_{mut}]$, nous appliquons l'opérateur de mutation sur cet individu. Comme pour les croisements, de nombreuses méthodes de mutation ont été présentées dans la littérature (Nearchou, 2004). Nous citons les plus connues :

- **Opérateur d'inversion simple :** cet opérateur consiste à choisir aléatoirement deux points de coupure et inverser les positions des gènes situées au milieu ;

- **Opérateur d'insertion :** cet opérateur consiste à sélectionner au hasard un gène et une position dans le chromosome à muter, puis à insérer le gène sélectionné dans la position choisie ;

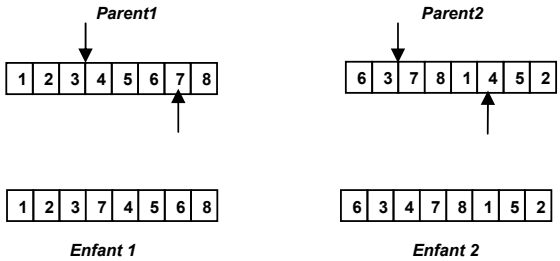

FIG. 2.4 – Mutation : opérateur d'insertion

- **Opérateur d'échange réciproque :** c'est un opérateur qui permet de sélectionner deux gènes et de les changer.

2.3.7 Insertion

Après l'étape de mutation, on utilise une méthode d'insertion pour générer une nouvelle population. Plusieurs stratégies ont été présentées dans la littérature :

- première stratégie, consiste à choisir les P_{size} individus à partir des P_{size} enfants déjà créés par les opérateurs de croisement et mutation. Dans ce cas, les parents sont remplacés par les enfants mutés.
- une autre stratégie consiste à choisir les P_{size} individus à partir des P_{size} parents de la population précédente et de P_{size} nouveaux enfants. A chaque itération, les individus ayant les meilleures *fitness* seront sélectionnés afin de créer des individus fils qui remplaceront les plus mauvais parents. Le reste survit et sera copié dans la nouvelle génération.
- l'élitisme , consiste à copier quelques meilleurs individus dans la nouvelle population. L'objectif est d'éviter que les meilleurs individus soient perdus après les opérateurs de croisement et de mutation. Cette méthode permet de conserver, à une itération donnée, le meilleur individu trouvé dans toutes les populations générées antérieurement.

2.3.8 Critère d'arrêt

Le test d'arrêt joue un rôle très important dans le jugement de la qualité des individus. Les critères d'arrêt sont de deux types :

- arrêt après un nombre fixé a priori de générations ;
- arrêt lorsque la population cesse d'évoluer ou n'évolue plus suffisamment.

Les algorithmes génétiques ont prouvé leurs performances, dans la littérature, en tant qu'outils de résolution des problèmes réels. A cet effet, notre objectif est le développement d'un algorithme génétique pour l'ordonnancement robuste d'un problème réel d'ordonnancement, qui est le problème des blocs opératoires.

Afin de valider la performance de notre algorithme, nous proposons de le tester sur un problème théorique connu dans la littérature et qui se rapproche, dans sa structure, à

notre problème réel. Ce problème est le problème d'ordonnancement du *flow shop* hybride. Ce dernier sera présenté dans la section suivante.

2.4 Le problème d'ordonnancement du *flow shop* hybride

2.4.1 Présentation

Une organisation de type *flow shop* hybride constitue notre cas d'étude. Il s'agit d'ordonnancer un ensemble de n *jobs* types qui sont représentatifs des scénarios de production possibles $J_1, J_2, ..., J_n$ dans un atelier de production composé de plusieurs étages k.

Chaque *job* J_j est constitué de k opérations types O_{jt} : O_{jt} est l'opération du *job* j traitée par l'étage t.

L'étage t contient un ensemble de $M^t \geqslant 1$ machines parallèles identiques, ce qui signifie que les *jobs* peuvent être exécutés indifféremment sur l'une ou l'autre des machines d'un même étage. Chaque *job* doit passer sur une des machines dans chaque étage (voir figure 2.5).

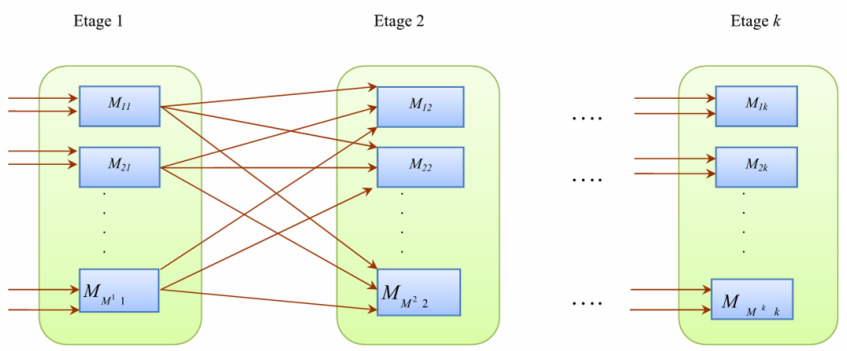

FIG. 2.5 – *flow shop* hybride à k étages

Une fois que l'exécution d'un *job* a débuté sur une machine, celle-ci ne peut pas être interrompue. Ceci revient à dire que l'ordonnancement est non préemptif. Le *job* ne peut pas être exécuté dans des différentes machines. Aucun *job* ne peut commencer sur cette

machine avant la fin du *job* en cours.

Pour chacun des *jobs*, l'ordre de passage des opérations dans les k étages est le même : $O_{j1}, O_{j2}, ..., O_{jt}$. La machine M_{it} avec $i = 1, 2, ..., M^t$ et $t = 1, 2, ..., k$ ne peut exécuter qu'un seul *job* à la fois.

Le temps opératoire du *job* j à l'étage t est noté P_{jt} (les temps de montage, de transport, de décharge... sont contenus dans le temps opératoire du *job*).

Nous présentons dans la section suivante un état de l'art de ce problème.

2.4.2 État de l'art du problème d'ordonnancement du *flow shop* hybride

Le problème d'ordonnancement du *flow shop* hybride est l'un des problèmes les plus difficiles et également les plus courants dans le domaine industriel. Gupta, en 1988, a ouvert le champ d'études des problèmes d'ordonnancement du *flow shop* hybride et prouvé que le cas le plus simple de ce problème est NP-complet (le cas où il y a une seule machine dans le premier étage et deux machines dans le deuxième étage). Depuis, de nombreuses recherches ont été menées et les méthodes et techniques utilisées s'étendent des heuristiques aux métaheuristiques (comme le recuit simulé, la recherche tabou, les algorithmes génétiques, etc.) et aux méthodes hybrides, en passant par des méthodes exactes, avec comme objectif la minimisation du *makespan* ou le temps de séjour total (ou moyen) des travaux ou du nombre des travaux en retard, etc.

Par exemple, Allaoui et Artiba (2006) ont étudié le problème d'ordonnancement du *flow shop* hybride à 2 étages, une machine au premier étage et plusieurs machines au deuxième étage, avec des contraintes de disponibilité des machines dont l'objectif est la minimisation du *makespan*. Ils l'ont résolu par une procédure de séparation et évaluation. Brah et Hunsucker (1991) ont utilisé la même technique de résolution, mais dans le cas de plusieurs étages. Rajendran et Holthaus (1999) s'adressent aussi au même problème mais en considérant le temps de séjour total comme critère à minimiser.

Gupta (1988) a développé des heuristiques pour résoudre un problème d'ordonnancement particulier du *flow shop* hybride à 2 étages dont une seule machine dans le deuxième étage. L'objectif est la minimisation du *makespan*. Par la suite, Gupta *et al.* (2002) ont proposé des heuristiques pour résoudre un problème d'ordonnancement du flow shop hybride à k étages, dont l'objectif est la minimisation du retard. Sriskandarajah (1993) a proposé aussi une heuristique qui partitionne tout d'abord le *flow shop* hybride en plusieurs *flow shop* simples, ensuite répartit les travaux à ordonnancer entre les différents *flow shop* et finalement ordonnance chaque *flow shop* à l'aide de l'algorithme de Gilmore et Gomory (1964) .

Différentes métaheuristiques ont été proposées pour obtenir des solutions approchées. Gourgand *et al.* (1999) ; Jin *et al.* (2006) ont utilisé le recuit simulé pour la résolution du problème d'ordonnancement du *flow shop* hybride. Nowicki et Smutnicki (1998) ont appliqué la recherche tabou pour la résolution de ce même problème. Ruiz *et al.* (2005b) ont utilisé l'algorithme génétique pour la résolution du problème d'ordonnancement du *flow shop* hybride avec des machines différentes.

Pour plus de détails, le lecteur peut se référer aux travaux de Linn et Zhang (1999), Vignier *et al.* (1999) et Ruiz et Vázquez-Rodríguez (2010) qui ont réalisé un état de l'art des problèmes d'ordonnancement de *flow shop* hybride.

Dans la pratique et dans la plupart des problèmes réels d'ordonnancement, plusieurs facteurs incontrôlables peuvent perturber le temps d'achèvement et par la suite l'ordonnancement ne peut pas être réalisé comme prévu. Pour cet effet, il est nécessaire de développer un algorithme efficace pour avoir un ordonnancement robuste. En effet, avant de présenter notre algorithme génétique pour l'ordonnancement robuste, nous développons un algorithme génétique pour la résolution du problème d'ordonnancement du flow shop hybride dans un environnement déterministe. Cet algorithme consiste à minimiser seulement le *makespan*, sans prendre en compte les perturbations sur les temps d'exécution des *jobs* sur les machines. L'objectif de développement de cet algorithme est de vérifier son efficacité dans le cas déterministe. Puis, nous décrivons comment l'algorithme génétique

pour l'ordonnancement déterministe peut être étendu pour trouver des solutions qui sont non seulement efficaces (*makespan* minimum), mais aussi robustes face aux incertitudes.

2.5 Algorithme génétique pour l'ordonnancement déterministe : cas du problème *flow shop* hybride

2.5.1 Codage utilisé pour notre algorithme

Puisque nous sommes dans le cas de la résolution d'un problème d'ordonnancement de permutation. Le codage que nous avons retenu est de longueur égale au nombre total de *jobs* à réaliser (Yamada et Nakano, 1992; Nearchou, 2004). L'individu est représenté par une séquence de *jobs*. Cette séquence est appliquée seulement au premier étage. Pour les autres étages, c'est-à-dire de l'étage 2 à l'étage k, les *jobs* sont ordonnés selon leurs dates d'achèvement minimales de l'étage précédent. Cela signifie que la règle FIFO (*First In First Out*) est utilisée pour trouver la séquence des *jobs* pour les étages suivants. Il est à noter que la séquence des *jobs* diffère d'un étage à un autre. L'affectation des *jobs* sur les machines est faite en appliquant la règle FAM (*First available Machine*) : nous affectons le *job* à la machine la plus disponible. La population initiale est aléatoirement produite. La population est formée par un nombre donné de chromosomes notés par P_{size}. L'objectif principal est de minimiser le *makespan* (temps d'achèvement des *jobs*).

La figure 2.6 présente le codage d'une solution quelconque. Le *job* numéro 4 sera lancé en production le premier, suivi du *job* numéro 7...et enfin le *job* numéro 8, placé dans le dernier chromosome du code, sera lancé le dernier. Nous dirons alors que le code génétique d'une solution envisagée est constitué de N *jobs*.

4	7	3	1	6	5	2	8

FIG. 2.6 – Codage d'un chromosome

2.5.2 Génération d'une population initiale

Dans cette étape, une population initiale doit être générée, où chaque chromosome représente une solution réalisable du problème. La procédure que nous avons utilisée pour générer la population initiale des individus, adaptée à notre problème est une génération aléatoire de P_{size} individus. Ce type de génération est le meilleur choix, car il permet l'hétérogénéité de la population.

2.5.3 Opérateur de sélection

La méthode utilisée est la sélection aléatoire. Cette sélection se fait aléatoirement, uniformément et sans intervention de la valeur de la fonction objectif. Chaque individu a donc une probabilité uniforme $(1/P_{size})$ d'être sélectionné.

2.5.4 Opérateur de croisement

Pour obtenir de nouveaux individus (enfants) à partir d'une population initiale d'une itération, nous utiliserons l'opérateur de croisement en 2-points décrit dans la section 2.3.5. Ce croisement s'effectuera comme suit :

Étape 1 : Choisir deux individus de la population actuelle comme parents P_1 et P_2 pour générer deux enfants E_1 et E_2 ;

Étape 2 : Générer aléatoirement deux positions $k_1 \in [1, ..., N]$ et $k_2 \in [1, ..., N]$ avec $k_1 < k_2$ et générer une probabilité de croisement α ;

Étape 3 : Si $\alpha \leq P_{cross}$;

Étape 3-1 : Copier tous les éléments (les éléments qui sont inférieurs à k_1 et supérieurs à k_2) des parents choisis sur les deux enfants respectivement, ici, nous copions ceux du parent P_1 à l'enfant O_1, ceux du parent P_2 à l'enfant O_2 ;

Étape 3-2 : Compléter la partie restante de l'enfant $E1$ par les éléments du parent P_2 et la partie restante de l'enfant E_2 par les éléments du parent P_1 en balayant de gauche à droite et en ne reprenant que les éléments non encore transmis ;

Étape 4 : Si $\alpha > P_{cross}$, copier le parent P_1 à l'enfant E_1 et le parent P_2 à l'enfant E_2.

2.5.5 Opérateur de mutation

Les individus de la population issue du croisement vont ensuite subir un processus de mutation. L'opérateur de mutation utilisé est l'opérateur d'insertion décrit dans la section 2.3.6.

Étape 1 : Générer aléatoirement deux gènes $s_1 \in [1, ..., N]$ et $s_2 \in [1, ..., N]$ avec $s_1 \neq s_2$ et générer une probabilité de mutation β ;

Étape 2 :

- Si $s_1 < s_2$ insérer le gène s_1 après le gène s_2 pour avoir un enfant muté noté E_{mut_1} (respectivement E_{mut_2}) ;
- Si $s_1 > s_2$ insérer le gène s_1 avant le gène s_2 pour avoir un enfant muté noté E_{mut_1} (respectivement E_{mut_2}) ;

Étape 3 :

- Si $\beta \leq P_{mut}$, muter seulement l'enfant E_1 pour avoir un enfant muté E_{mut_1} ;
- Si $\beta > P_{mut}$, muter seulement l'enfant E_2 pour avoir un enfant muté E_{mut_2} ;

2.5.6 Opérateur d'insertion

L'insertion dans le contexte de l'ordonnancement est employée pour réduire au minimum la fonction *fitness*. Cette insertion permet d'éliminer de la population les chromosomes les plus faibles. En effet, les individus qui sont générés aléatoirement et les enfants mutés (égale à $2 * P_{size}$)sont triés selon leur fonction de *fitness* dans un ordre croissant. Seule la moitié supérieure de la population, correspondant aux meilleurs individus, est sélectionnée. Il est à noter que la taille de la population reste fixe (égale à P_{size}) de génération en génération.

2.5.7 Fonction d'évaluation : *fitness*

La fonction d'évaluation consiste à minimiser le *makespan* (noté $Cmax$).

$$Cmax = \underset{\substack{j=1,...,n \\ t=1,...,k}}{Max} \{C_{jt}\} \tag{2.1}$$

C_{jt} désigne le temps d'achèvement du *job* j dans l'étage t.

2.5.8 Critère d'arrêt

Le critère d'arrêt utilisé est le nombre de générations égal à $Iter_{Max}$. Après $Iter_{Max}$ générations, l'algorithme génétique s'arrête et donne le meilleur chromosome qui possède un $Cmax$ minimum.

Notre algorithme génétique pour l'ordonnancement déterministe est initialisé par une taille de population (P_{size}), une probabilité de croisement (P_{cross}), une probabilité de mutation (P_{mut}) et par un critère d'arrêt ($Iter_{Max}$). Les étapes de cet algorithme sont présentées dans l'algorithme 1.

Algorithme 1 Algorithme génétique pour l'ordonnancement déterministe

1: Initialiser : P_{size}, P_{cross}, P_{mut}, $Iter_{Max}$, $\mu = 0$
2: Générer aléatoirement P_{size} séquences de *jobs*.
3: **répéter**
4: $i = 0$
5: **Tant que** $\mu < \frac{P_{size}}{2}$ **faire**
6: Sélectionner aléatoirement deux parents de la population
7: Croisement des deux parents pour obtenir deux enfants par une probabilité P_{cross}
8: Muter les deux enfants par une probabilité P_{mut}
9: $\mu = \mu + 1$
10: **Fin tant que**
11: Évaluer tous les chromosomes ($2P_{size}$, parents et enfants) par la fonction d'évaluation *fitness*
12: Ranger les parents et les enfants dans l'ordre croissant selon leur fonction d'évaluation
13: Supprimer les P_{size} chromosomes faibles et enregistrer les meilleurs chromosomes ($Best_{Pop}$) selon leur fonction d'évaluation *fitness*
14: Remplacer ($P_{size}, Best_{Pop}$)
15: $i = i + 1$
16: **Jusqu'à** $i = Iter_{Max}$

2.6 Algorithme génétique pour l'ordonnancement robuste : cas du problème *flow shop* hybride

Dans cette section, nous présenterons une application de l'algorithme génétique pour la résolution du problème d'ordonnancement du *flow shop* hybride. Cet algorithme fournit une solution non seulement de bonne qualité mais également robuste. La recherche de

cette solution consiste à étendre l'algorithme génétique pour l'ordonnancement déterministe, où la fonction d'évaluation *fitness* de l'algorithme génétique pour l'ordonnancement déterministe doit être remplacée par une fonction d'évaluation robuste. La détermination de cette dernière est basée sur un critère de robustesse agrégeant deux objectifs.

Deux nouveaux mécanismes possibles permettent d'obtenir une solution efficace et robuste. Le premier mécanisme consiste à évaluer, à chaque génération, dans l'algorithme génétique, tous les chromosomes sur un ensemble de scénarios perturbés (c'est-à-dire que la population est générée aléatoirement et les nouveaux enfants mutés sont évalués sur les mêmes scénarios perturbés). Cet ensemble de scénarios change d'une génération à une autre. En effet ce premier mécanisme permet d'augmenter la probabilité d'obtenir une solution robuste sur un nombre important de scénarios perturbés (c'est-à-dire, si $Iter_{Max} = 1.000$ et le nombre de scénarios perturbés est $N = 10$, alors la solution robuste est évaluée sur $1.000 * 10 = 10.000$ scénarios perturbés). Le deuxième mécanisme permet d'évaluer chaque chromosome sur un ensemble de scénarios perturbés (c'est-à-dire que chaque chromosome est évalué sur son propre ensemble de scénarios perturbés). Cet ensemble de scénarios change à chaque évaluation d'un chromosome.

Le premier mécanisme est le plus cohérent, puisque l'évaluation des chromosomes s'effectue sur le même ensemble de scénarios. Dans cette thèse, nous choisissons le premier mécanisme et l'implémentation du deuxième est envisageable dans des travaux futurs.

2.6.1 Fonction d'évaluation robuste

Soit x une solution du problème (une séquence de *jobs*). La qualité de la solution x est calculée par une fonction d'évaluation $f(x)$. I est un scénario initial qui représente les caractéristiques (*inputs*) du problème (dans notre cas, le temps d'exécution des *jobs* dans chaque étage). Alors, la fonction d'évaluation de la solution x du scénario I s'écrira $f_I(x)$. En effet, pour calculer une solution robuste, la fonction d'évaluation $f(x)$ est remplacée par une fonction d'évaluation robuste $f^r(x)$ (ou critère de robustesse).

Pour représenter les incertitudes sur les caractéristiques du problème (temps d'exécution incertain), nous choisissons la modélisation par scénarios (décrite dans la section

1.4.3). Le contenu des scénarios est le reflet d'hypothèses concernant l'incertain. Cette modélisation nécessite la construction d'un ensemble de scénarios (appelé aussi instances ou jeux de données) contenant les temps d'exécution des *jobs* dans chaque étage. L'ensemble des scénarios, noté par une fonction ξ, est représenté par une fonction d'échantillonnage à partir de l'instance initiale I. On notera aussi $\xi_i(I)$ le $i^{ième}$ échantillon des paramètres de l'instance I (avec $i = 1, ..., N$, où N représente l'ensemble des instances perturbées). $\xi_i(I)$ est une variable aléatoire uniforme sur $[P_I - \alpha P_I, P_I + \alpha P_I]$, avec P_I le temps d'exécution du scénario initial et $\alpha \in [0, 1]$ exprime le degré d'incertitude des temps d'exécution.

La fonction d'évaluation robuste est présentée comme suit :

$$f^r(x) = \lambda * f_I(x) + (1 - \lambda) * \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(f_{\xi_i(I)}(x) - f_I(x)\right)^2} \qquad (2.2)$$

$f_{\xi_i(I)}(x)$ (avec $i = 1, ..., N$) est la fonction d'évaluation du scénario $\xi_i(I)$. Cette fonction est obtenue en intégrant la solution x obtenue par le scénario initial I dans l'instance $\xi_i(I)$.

λ est un poids fixé par le décideur ($\lambda \in [0, 1]$) et qui reflète l'importance qu'il accorde à chacun des objectifs.

Ce critère de robustesse a pour objectif de minimiser simultanément la fonction d'évaluation du scénario initial et la déviation entre la fonction d'évaluation de tous les scénarios et le scénario initial. Cette déviation permet de mesurer la dispersion des valeurs de la fonction d'évaluation des scénarios perturbés autour de la valeur de la fonction d'évaluation du scénario initial.

Ce critère de robustesse est générique et peut être utilisé dans plusieurs environnements de production (*flow shop*, *job shop*, *job shop* flexible, etc.). Il peut être appliqué sur différents critères de performance (le *makespan*, le retard maximal, le temps total de présence (*total flow time*), le temps moyen de présence (*mean flow time*) le retard moyen

(*mean tardiness*), le retard algébrique (*maximum lateness*),etc). En effet, il suffit de remplacer la fonction d'évaluation f par le critère approprié.

Puisque le critère de performance utilisé dans cette thèse est de minimiser le *makespan* (noté $Cmax$), il s'agit de remplacer $f_I(x)$ par $Cmax_I(x)$ et $f_{\xi_i(I)}(x)$ par $Cmax_{\xi_i(I)}(x)$. Alors la fonction d'évaluation robuste $f^r(x)$ consiste à minimiser simultanément le *makespan* du scénario initial et la déviation entre le *makespan* des scénarios perturbés et le *makespan* du scénario initial. Cette déviation permet de mesurer la dispersion des *makespan* des scénarios perturbés autour du *makespan* du scénario initial. Cette fonction d'evaluation robuste s'écrira comme suit :

$$f^r(x) = \lambda * Cmax_I(x) + (1-\lambda) * \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(Cmax_{\xi_i(I)}(x) - Cmax_I(x)\right)^2} \tag{2.3}$$

$Cmax_I(x)$ est le *makespan* du scénario initial.

$Cmax_{\xi_i(I)}(x)$ est le *makespan* du scénario perturbé.

La fonction d'évaluation robuste est une agrégation de deux objectifs. Nous remarquons que ces deux objectifs n'ont pas les mêmes échelles de mesures. Afin d'avoir des échelles communes aux deux objectifs, nous avons normalisé les valeurs correspondant à ces unités. La fonction d'évaluation robuste $f^r(x)$ devient comme suit :

$$f^r(x) = \lambda * \frac{Cmax_I(x) - LB}{LB} + (1-\lambda) * \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(Cmax_{\xi_i(I)}(x) - Cmax_I(x)\right)^2}}{DEV\_MAX(x^*)} \tag{2.4}$$

LB est la borne inférieure du scénario initial.

$$DEV\_MAX(x^*) = Max\left\{[Cmax_I(x^*) - Cmax_{I_{Min}}(x^*)], [Cmax_{I_{Max}}(x^*) - Cmax_I(x^*)]\right\}. \tag{2.5}$$

(x^*) est la meilleure solution (séquence) calculée par l'algorithme génétique pour l'ordonnancement déterministe (l'algorithme 1).

$Cmax_{I_{Min}}(x^*)$ est le *makespan* du scénario minimal. Ce scénario noté I_{Min}, contient les temps d'exécution des *jobs* minimaux calculé par le scénario initial, où $I_{Min} = I - \alpha I$.

$Cmax_{I_{Max}}(x^*)$ est le *makespan* du scénario maximal. Ce scénario noté I_{Max} contient les temps d'exécution des *jobs* maximaux calculé par le scénario initial, où $I_{Max} = I + \alpha I$.

Ces deux dernières valeurs ($Cmax_{I_{Min}}(x^*)$ et $Cmax_{I_{Max}}(x^*)$) sont calculées par l'intégration de la solution (x^*) obtenue par le scénario initial I dans le scénario minimal I_{Min} et maximal I_{Max}.

Pour trouver un ensemble de solutions pour ce problème, nous faisons varier la valeur λ entre 0 et 1 et nous appliquons l'algorithme génétique pour l'ordonnancement robuste pour chaque valeur de λ. Plus λ est important, plus la priorité est donnée à la minimisation de l'écart associé à $Cmax_I$ par rapport à celui associé à la déviation entre les scénarios perturbés et le scénario initial. Par exemple, pour trouver une solution robuste, ce qui est le cas de $\lambda = 0$, nous favorisons la minimisation de la déviation entre les scénarios perturbés et le scénario initial. Pour trouver une solution efficace, ce qui est le cas de $\lambda = 1$, nous favorisons la minimisation du *makespan* du scénario initial ($Cmax_I$). Dans ce dernier cas, la solution nommée comme efficace peut être obtenue non seulement par l'algorithme génétique pour l'ordonnancement robuste, mais aussi par l'algorithme génétique pour l'ordonnancement déterministe (décrit par l'algorithme 1) puisque les perturbations n'interviennent pas dans les deux algorithmes.

Dans le cadre de cette thèse, la fonction d'évaluation robuste proposée est une agrégation monocritère de deux objectifs. Bien évidemment, d'autres méthodes, telles que les méthodes multicritères basées sur l'analyse de Pareto, sont également envisageables

afin d'optimiser simultanément ces deux objectifs. Ceci constitue une des perspectives de notre thèse. Nous avons opté pour une démarche progressive (agrégation monocritère, puis à terme multicritères) pour avancer étape par étape sur la maîtrise de la notion de robustesse et de sa caractérisation.

2.6.2 Test de convergence

Ce test de convergence permet de déterminer le nombre de scénarios perturbés N. plusieurs techniques de convergence consistent à déterminer le nombre de scénarios perturbés. Parmi ces techniques, la moyenne, l'écart-type, etc. Dans cette thèse, nous choisissons comme test de convergence, la moyenne de tous les scénarios perturbés. Parce que dans nos résultats expérimentaux, présenté dans le chapitre 3, nous comparons la moyenne des makespan de tous les scénarios perturbés par rapport au makespan du scénario initial.

En effet, le nombre de scénarios perturbés est égal à N, si la moyenne des $Cmax$ des N scénarios perturbés, moins la moyenne des $Cmax$ de $N + 10$ des scénarios perturbés est inférieure à ε, alors le test de convergence est satisfait.

$$\frac{1}{N}\sum_{i=1}^{N}\left(Cmax_{\xi_i(I)}(x)\right) - \frac{1}{N+10}\sum_{i=1}^{N+10}\left(Cmax_{\xi_i(I)}(x)\right) \leq \varepsilon \tag{2.6}$$

Dans notre algorithme génétique pour l'ordonnancement robuste, le nombre des scénarios perturbés $N \in [10, 20, ..., 100]$ avec pas de 10, et $\varepsilon = 0,5$. Si le test de convergence n'est pas satisfait, alors ce test est arrêté et le nombre de scénarios perturbés est fixé à $N = 100$. Suite à quelques expérimentations effectuées, nous avons constaté que, pour une valeur de $\varepsilon = 0,5$ nous obtenons de bons résultats en un temps de calcul raisonnable. A cet effet, nous avons fixé une valeur de $\varepsilon = 0,5$ dans notre algorithme génétique pour l'ordonnancement robuste.

2.6.3 Description de l'algorithme génétique pour l'ordonnancement robuste

Dans cette section nous décrivons comment l'algorithme génétique pour l'ordonnancement déterministe peut être modifié pour donner des solutions robustes et efficaces.

Les opérateurs utilisés dans l'algorithme génétique pour l'ordonnancement robuste sont les mêmes opérateurs utilisés dans l'algorithme génétique pour l'ordonnancement déterministe. De plus, le fonctionnement de l'algorithme génétique pour l'ordonnancement robuste est semblable au fonctionnement de l'algorithme génétique pour l'ordonnancement déterministe, mais ici nous intégrons un nouveau mécanisme. Ce mécanisme consiste à évaluer, à chaque génération de l'algorithme génétique, tous les chromosomes, sur un ensemble de scénarios perturbés. Cet ensemble de scénarios change d'une génération à une autre. De plus, la fonction d'évaluation *fitness* (décrite dans la section 2.5.7) est remplacée par la fonction d'évaluation robuste (décrite dans la section 2.6.1).

Cet algorithme permet de générer une séquence robuste qui minimise le *makespan* du scénario initial et la déviation entre ce *makespan* et le *makespan* des scénarios perturbés. Le processus de l'algorithme génétique pour l'ordonnancement robuste est illustré dans l'algorithme 2. Cet algorithme est initialisé par une taille de la population P_{size}), une probabilité de croisement (P_{cross}), une probabilité de mutation (P_{mut}), une petite valeur pour tester la convergence (ε) et par un critère d'arrêt ($Iter_{Max}$).

Algorithme 2 Algorithme génétique pour l'ordonnancement robuste

1: Initialiser : P_{size}, P_{cross}, P_{mut}, $Iter_{Max}$, $\mu = 0$, N, $\varepsilon = 0.5$.
2: Calculer DEV_MAX
3: Générer aléatoirement P_{size} séquences de *jobs*.
4: **répéter**
5: $i = 0$
6: Génération de N scénarios perturbés à partir du scénario initial.
7: Test de convergence des scénarios perturbés.
8: **Tant que** $\mu < \frac{P_{size}}{2}$ **faire**
9: Sélectionner aléatoirement deux parents de la population
10: Croisement des deux parents pour obtenir deux enfants par une probabilité P_{cross}
11: Muter les deux enfants selon une probabilité P_{mut}
12: $\mu = \mu + 1$
13: **Fin tant que**
14: Évaluer tous les chromosomes ($2P_{size}$, parents et enfants) par la fonction d'évaluation *fitness*
15: Ranger les parents et les enfants dans l'ordre croissant, selon leur fonction d'évaluation
16: Supprimer les P_{size} chromosomes faibles et enregistrer les meilleurs chromosomes ($Best_{Pop}$), selon leur fonction d'évaluation *fitness*
17: Remplacer ($P_{size}, Best_{Pop}$)
18: $i = i + 1$
19: **Jusqu'à** $i = Iter_{Max}$

2.7 Conclusion

Dans ce chapitre, nous avons développé un algorithme génétique pour l'ordonnancement robuste pour le problème du *flow shop* hybride. Un critère de robustesse a été développé. Ce critère consiste à trouver une solution de bonne performance et qui est peu sensible aux incertitudes. L'objectif est de minimiser simultanément la date d'achèvement des *jobs* (*makespan*) et la déviation entre les scénarios perturbés et le scénario initial. Nous avons développé aussi un nouveau mécanisme intégré dans l'algorithme génétique. Ce mécanisme consiste à évaluer, à chaque génération de l'algorithme génétique, tous les chromosomes, sur un ensemble de scénarios perturbés. Une illustration du rôle de la fonction d'évaluation robuste à l'aide d'un exemple académique est présentée dans l'annexe A. Dans le chapitre, suivant nous proposons de vérifier et de comparer notre algorithme génétique pour l'ordonnancement déterministe à d'autres méthodes développées dans la littérature et de valider notre algorithme génétique pour l'ordonnancement robuste par la simulation. Ce modèle de simulation nous permettra de tester et valider la robustesse.

Chapitre 3

Vérification et validation des algorithmes génétiques

Sommaire

3.1 Introduction

Dans le chapitre précédent, nous avons présenté un algorithme génétique pour la résolution du problème d'ordonnancement d'un flow shop hybride. Deux types d'algorithmes ont été ainsi définis. Le premier est un algorithme génétique pour l'ordonnancement déterministe minimisant seulement le *makespan*, sans tenir compte des perturbations sur les temps d'exécution des *jobs* dans les étages et le second est un algorithme génétique pour l'ordonnancement robuste minimisant simultanément le *makespan* du scénario initial et la déviation entre les scénarios perturbés et le scénario initial. L'objectif est de trouver une solution efficace (un *makespan* minimum) et robuste qui résiste face aux perturbations.

Dans ce chapitre, nous testons en premier lieu l'efficacité de l'algorithme génétique pour l'ordonnancement déterministe minimisant le $Cmax$. Dans le but de vérifier les résultats obtenus, cet algorithme a été comparé à deux méthodes de résolution (une méthode de type *Branch and Bound* (B&B) et la méthode de système immunitaire artificiel (AIS)) présentées dans la littérature. En deuxième lieu, nous validons notre algorithme génétique pour l'ordonnancement robuste développé dans le chapitre 2. Cette validation est réalisée par la simulation.

3.2 Vérification de l'algorithme génétique pour l'ordonnancement déterministe

Dans cette section, nous présentons l'efficacité de notre algorithme génétique pour l'ordonnancement déterministe dont l'objectif est de minimiser seulement le *makespan* ($Cmax$). Cette série d'expérimentations est effectuée de manière statique, sans intégrer des perturbations. Les données utilisées sont déterministes et ne prennent pas en compte des incertitudes sur les temps d'exécution.

3.2.1 Description des *Benchmarks*

L'algorithme génétique pour l'ordonnancement déterministe a été testé sur des *Benchmarks* (jeux-tests, instances) présentés par (Carlier et Néron, 2000). Le nombre total de ces *benchmarks* est 77, dont la taille est de 10 ou 15 *jobs*, et 5 ou 10 étages, suivant les cas. Les temps d'exécution sont générés aléatoirement dans l'intervalle $[3, 20]$, quel que soit le nombre de machines dans les étages.

Pour un nombre de *jobs* et un nombre d'étages fixés, il existe plusieurs configurations machines possibles (a, b, c et d). Nous appelons une configuration de machine une liste de nombres représentant le nombre de machines dans chacun des étages (voir tableau 3.1). La configuration de type a dispose de 2 machines dans chaque étage, sauf le premier, ou le dernier, ou l'étage qui se situe au milieu (selon le nombre de *jobs* et d'étages) qui dispose seulement d'une seule machine. Par exemple l'instance $j10c5a2$ correspond à une

configuration de 22122[1]. La configuration de type b dispose d'une seule machine dans le premier étage et de 2 machines dans les autres étages (c'est à dire 12222 dans le cas de 5 étages), sauf le cas de 15 *jobs* et 10 étages, où une seule machine est disponible dans le dernier étage. La configuration de type c dispose de 3 machines dans chaque étage, sauf l'étage qui se situe au milieu, qui dispose de 2 machines seulement(par exemple l'instance $j10c5c1$ est une instance de 10 *jobs* et 5 étages, dans lequelle les deux premiers étages disposent de 3 machines, seulement 2 machines dans le troisième étage et les deux derniers étages disposent de 3 machines). La configuration de type d dispose de 3 machines dans chaque étage.

Types	$j10c5$ et $j15c5$	$j10c10$	$j15c10$
a	22122	2222122222	1222222222
b	12222	1222222222	2222222221
c	33233	3333233333	-
d	33333	-	-

TAB. 3.1 – Différents types de configurations

En se basant sur Carlier et Néron (2000), certains des *benchmarks* sont groupés en tant qu'instances difficiles. Ces instances comprennent les configurations de types c et d pour les instances de 10 *jobs* et 5 étages et les instances de 15 *jobs* et 5 étages ($j10c5\times$ et $j15c5\times$). Le reste des instances (de types a, b et $j10c10c$) sont identifiées en tant qu'instances faciles.

3.2.2 Expérimentations et résultats

Notre algorithme a été implémenté avec Microsoft Visual C++ 6.0 et testé sous Windows XP professionnel. Les expérimentations sont effectuées sur un ordinateur personnel Pentium IV avec 1,70 GHz et 512 Mo de RAM.

Après plusieurs expérimentations, nous avons retenu les paramètres suivants :

- ⋆ une taille de population égale à 200 ;
- ⋆ une probabilité de croisement égale à 90% ;
- ⋆ une probabilité de mutation égale à 20% ;
- ⋆ un critère d'arrêt qui est le nombre de générations, égal à 1000 ($Iter_{Max} = 1.000$).

[1]22122 : de gauche à droite : nous avons 2 machines à l'étage 1, 2 à l'étage 2, 1 à l'étage 3, ...

Les résultats de l'algorithme génétique ont été comparés à deux méthodes de résolution qui ont été développées dans la littérature et ce sont les seules méthodes qui ont utilisé des *Benchmarks* qui sont accessible. La première méthode, notée B&B, est une procédure par séparation et évaluation (*Branch & Bound*) développée par Carlier et Néron (2000). La deuxième méthode est un algorithme basé sur le système immunitaire artificiel, noté AIS, développé par Engin et Doyen (2004). Cette méthode est une catégorie d'algorithme inspirée par les principes et le fonctionnement du système immunitaire naturel des vertébrés. Ce type d'algorithme exploite typiquement les caractéristiques du système immunitaire, pour ce qui est de l'apprentissage et de la mémorisation comme moyens de résolution de problèmes. Dans ces deux méthodes (B&B et AIS), les auteurs ont limité leurs algorithmes par un temps de calcul égal à 1600 secondes. Si, au bout de 1600 secondes, la solution optimale n'est pas trouvée, alors l'algorithme s'arrête et la meilleure solution rencontrée est acceptée comme ordonnancement final.

Les résultats obtenus ont été comparés à une borne inférieure globale (LB) développée par Carlier et Néron (2000).Le pourcentage de déviation est calculé par la formule suivante :

$$\%Déviation = \frac{Cmax - LB}{LB} * 100 \tag{3.1}$$

Les tableaux 3.2 - 3.14 présentent une comparaison des pourcentages de déviation par rapport à la borne inférieure LB de notre algorithme génétique pour l'ordonnancement déterministe avec l'algorithme de B&B et l'algorithme AIS développés dans la littérature. Cette comparaison montre que le pourcentage de déviation de notre algorithme génétique pour l'ordonnancement déterministe est, dans la plupart des instances, égal et parfois meilleur que l'algorithme du B&B et l'algorithme AIS. Ces derniers sont parfois meilleurs que notre algorithme génétique pour l'ordonnancement déterministe dans quelques instances.

Instances	*Cmax*			LB	*%Déviation*		
	AG	AIS	B&B		**AG**	AIS	B&B
J10c5a2	**88**	88	88	88	**0**	0	0
J10c5a3	**117**	117	117	117	**0**	0	0
J10c5a4	**121**	121	121	121	**0**	0	0
J10c5a5	**122**	122	122	122	**0**	0	0
J10c5a6	**110**	110	110	110	**0**	0	0
Moyenne	**111,60**	111,60	111,60	111,60	**0**	0	0

TAB. 3.2 – Comparaison entre AG, AIS et B&B pour l'instance de type J10c5a (instance facile)

Instances	*Cmax*			LB	*%Déviation*		
	AG	AIS	B&B		**AG**	AIS	B&B
J10c5b1	**130**	130	130	130	**0**	0	0
J10c5b2	**107**	107	107	107	**0**	0	0
J10c5b3	**110**	109	109	109	**0,92**	0	0
J10c5b4	**122**	122	122	122	**0**	0	0
J10c5b5	**153**	153	153	153	**0**	0	0
J10c5b6	**115**	115	115	115	**0**	0	0
Moyenne	**122,83**	122,67	122,67	122,67	**0,15**	0	0

TAB. 3.3 – Comparaison entre AG, AIS et B&B pour l'instance de type J10c5b (instance facile)

Instances	*Cmax*			LB	*%Déviation*		
	AG	AIS	B&B		**AG**	AIS	B&B
J10c5c1	**68**	68	68	68	**0**	0	0
J10c5c2	**75**	74	74	74	**1,35**	0	0
J10c5c3	**71**	72	71	71	**0**	1,41	0
J10c5c4	**66**	66	66	66	**0**	0	0
J10c5c5	**78**	78	78	78	**0**	0	0
J10c5c6	**69**	69	69	69	**0**	0	0
Moyenne	**71,17**	71,17	71	71	**0,23**	0,23	0

TAB. 3.4 – Comparaison entre AG, AIS et B&B pour l'instance de type J10c5c (instance difficile)

Instances	*Cmax*			LB	*%Déviation*		
	AG	AIS	B&B		**AG**	AIS	B&B
J10c5d1	**66**	66	66	66	**0**	0	0
J10c5d2	**73**	73	73	73	**0**	0	0
J10c5d3	**64**	64	64	64	**0**	0	0
J10c5d4	**70**	70	70	70	**0**	0	0
J10c5d5	**66**	66	66	66	**0**	0	0
J10c5d6	**62**	62	62	62	**0**	0	0
Moyenne	**66,83**	66,83	66,83	66,83	**0**	0	0

TAB. 3.5 – Comparaison entre AG. AIS et B&B pour l'instance de type J10c5d (instance difficile)

Instances	*Cmax*			LB	*%Déviation*		
	AG	AIS	B&B		**AG**	AIS	B&B
J10c10a1	**139**	139	139	139	**0**	0	0
J10c10a2	**159**	158	158	158	**0,63**	0	0
J10c10a3	**149**	148	148	148	**0,68**	0	0
J10c10a4	**149**	149	149	149	**0**	0	0
J10c10a5	**148**	148	148	148	**0**	0	0
J10c10a6	**146**	146	146	146	**0**	0	0
Moyenne	**148,33**	148	148	148	**0,22**	0	0

TAB. 3.6 – Comparaison entre AG, AIS et B&B pour l'instance de type J10c10a (instance facile)

Instances	*Cmax*			LB	*%Déviation*		
	AG	AIS	B&B		**AG**	AIS	B&B
J10c10b1	**163**	163	163	163	**0**	0	0
J10c10b2	**158**	157	157	157	**0,64**	0	0
J10c10b3	**169**	169	169	169	**0**	0	0
J10c10b4	**159**	159	159	159	**0**	0	0
J10c10b5	**165**	165	165	165	**0**	0	0
J10c10b6	**165**	165	165	165	**0**	0	0
Moyenne	**163,17**	163	163	163	**0,11**	0	0

TAB. 3.7 – Comparaison entre AG, AIS et B&B pour l'instance de type J10c10b (instance facile)

Instances	*Cmax*			LB	*%Déviation*		
	AG	AIS	B&B		**AG**	AIS	B&B
J10c10c1	**114**	115	127	113	**0,88**	1,77	12,40
J10c10c2	**119**	119	116	116	**2,59**	2,59	0
J10c10c3	**116**	116	133	98	**18,37**	18,37	35,70
J10c10c4	**120**	120	135	103	**16,50**	16,50	31,10
J10c10c5	**125**	126	145	121	**3,31**	4,13	19,80
J10c10c6	**105**	106	112	97	**8,25**	9,28	15,50
Moyenne	**116,50**	117	128	108	**8,32**	8,77	19,08

TAB. 3.8 – Comparaison entre AG, AIS et B&B pour l'instance de type J10c10c (instance facile)

Instances	*Cmax*			LB	*%Déviation*		
	AG	AIS	B&B		**AG**	AIS	B&B
J15c5a1	**178**	178	178	178	**0**	0	0
J15c5a2	**165**	165	165	165	**0**	0	0
J15c5a3	**130**	130	130	130	**0**	0	0
J15c5a4	**156**	156	156	156	**0**	0	0
J15c5a5	**164**	164	164	164	**0**	0	0
J15c5a6	**178**	178	178	178	**0**	0	0
Moyenne	**161,83**	161,83	161,83	161,83	**0**	0	0

TAB. 3.9 – Comparaison entre AG. AIS et B&B pour l'instance de type J15c5a (instance facile)

Instances	*Cmax*			LB	*%Déviation*		
	AG	AIS	B&B		**AG**	AIS	B&B
J15c5b1	**170**	170	170	170	**0**	0	0
J15c5b2	**152**	152	152	152	**0**	0	0
J15c5b3	**157**	157	157	157	**0**	0	0
J15c5b4	**147**	147	147	147	**0**	0	0
J15c5b5	**166**	166	166	166	**0**	0	0
J15c5b6	**175**	175	175	175	**0**	0	0
Moyenne	**161,17**	161,17	161,17	161,17	**0**	0	0

TAB. 3.10 – Comparaison entre AG. AIS et B&B pour l'instance de type J15c5b (instance facile)

Instances	*Cmax*			LB	*%Déviation*		
	AG	AIS	B&B		**AG**	AIS	B&B
J15c5c1	**85**	85	85	85	**0**	0	0
J15c5c2	**91**	91	90	90	**1,11**	1,11	0
J15c5c3	**87**	87	87	87	**0**	0	0
J15c5c4	**89**	89	90	89	**0**	0	1,10
J15c5c5	**75**	74	84	73	**2,74**	1,37	15,10
J15c5c6	**91**	91	91	91	**0**	0	0
Moyenne	**86,33**	86,17	87,83	85,83	**0,64**	0,41	2,70

TAB. 3.11 – Comparaison entre AG, AIS et B&B pour l'instance de type J15c5c (instance difficile)

Instances	*Cmax*			LB	*%Déviation*		
	AG	AIS	B&B		**AG**	AIS	B&B
J15c5d1	**167**	167	167	167	**0**	0	0
J15c5d2	**84**	84	85	82	**2,44**	2,44	3,70
J15c5d3	**83**	83	96	77	**7,79**	7,79	24,70
J15c5d4	**84**	84	101	61	**37,70**	37,70	65,60
J15c5d5	**79**	80	97	67	**17,91**	19,40	44,80
J15c5d6	**81**	82	87	79	**2,53**	3,80	10,10
Moyenne	**96,33**	96,67	105,50	88,83	**11,40**	11,86	24,82

TAB. 3.12 – Comparaison entre AG, AIS et B&B pour l'instance de type J15c5d (instance difficile)

Instances	*Cmax*			LB	*%Déviation*		
	AG	AIS	B&B		**AG**	AIS	B&B
J15c10a1	**236**	236	236	236	**0**	0	0
J15c10a2	**204**	200	200	200	**2**	0	0
J15c10a3	**198**	198	198	198	**0**	0	0
J15c10a4	**225**	225	225	225	**0**	0	0
J15c10a5	**182**	182	183	182	**0**	0	0,50
J15c10a6	**200**	200	200	200	**0**	0	0
Moyenne	**207,50**	206,83	207	206,83	**0,33**	0	0,08

TAB. 3.13 – Comparaison entre AG, AIS et B&B pour l'instance de type J15c10a (instance facile)

Instances	*Cmax*			LB	*%Déviation*		
	AG	AIS	B&B		**AG**	AIS	B&B
J15c10b1	**223**	222	222	222	**0,45**	0	0
J15c10b2	**187**	187	187	187	**0**	0	0
J15c10b3	**222**	222	222	222	**0**	0	0
J15c10b4	**221**	221	221	221	**0**	0	0
J15c10b5	**202**	200	200	200	**1**	0	0
J15c10b6	**219**	219	219	219	**0**	0	0
Moyenne	**212,33**	211,83	211,83	211,83	**0,24**	0	0

TAB. 3.14 – Comparaison entre AG, AIS et B&B pour l'instance de type J15c10b (instance facile)

Dans le tableau 3.15, nous comparons l'efficacité de notre algorithme génétique pour l'ordonnancement déterministe par rapport à deux méthodes mentionnées ci-dessus pour les instances faciles et difficiles. Nous remarquons que notre algorithme génétique pour l'ordonnancement déterministe fournit de meilleurs résultats, en termes de pourcentage de déviation moyen, que les deux autres méthodes, pour les problèmes difficiles (le pourcentage de déviation moyen de notre algorithme génétique pour l'ordonnancement déterministe est 3.06 % pour les problèmes difficiles). D'autre part, l'algorithme AIS obtient de meilleures solutions comparées à l'algorithme B&B, en termes de pourcentage de déviation moyen par rapport à LB. Pour des problèmes faciles, notre algorithme génétique pour l'ordonnancement déterministe fournit de meilleurs résultats que B&B, mais l'algorithme AIS est meilleur que notre algorithme génétique pour l'ordonnancement déterministe.

Méthodes	*%Déviation*	
	Instances faciles	Instances difficiles
AG	**1,09**	**3,06**
AIS	0,99	3,13
B&B	2,16	6,88

TAB. 3.15 – Performance des trois méthodes

Par ailleurs, nous remarquons que le pourcentage de déviation des instances faciles est inférieur à celui des instances difficiles, ce qui implique que les instances faciles convergent très rapidement vers la borne inférieure. En effet, les configurations des machines dans les différents étages ont un impact important sur la complexité des instances et par conséquent un effet sur la qualité de la solution (Engin et Doyen, 2004).

La comparaison des temps de calcul des trois méthodes est impossible, vu que les trois méthodes ont été implémentées et testées sur des machines de configurations différentes. Néanmoins, nous pouvons mentionner les temps de calculs suivants à titre indicatif. Nous remarquons, d'après le tableau 3.16, que les instances faciles convergent plus rapidement que les instances difficiles. En effet, pour les instances faciles, le temps moyen de calcul de l'algorithme de B&B et de l'algorithme AIS est égal à 240 secondes. Pour les instances difficiles, l'algorithme de B&B prend 1500 secondes et l'algorithme AIS prend 600 secondes en moyenne pour converger vers une meilleure solution. Notre algorithme génétique pour l'ordonnancement déterministe converge très rapidement quelle que soit le type d'instance (0,1 seconde pour les instances faciles et 0,2 seconde pour les instances difficiles).

Méthodes	CPU(s)	
	Instances faciles	Instances difficiles
AG	**0,1**	**0,2**
AIS	240	600
B&B	240	1500

TAB. 3.16 – Temps d'exécution moyens des trois méthodes

L'objectif de ces expérimentations n'est pas de donner de meilleurs résultats en termes de pourcentage de déviation par rapport à LB ou de montrer que notre algorithme génétique pour l'ordonnancement déterministe est meilleur que d'autres méthodes. Notre objectif est de montrer que notre algorithme génétique pour l'ordonnancement déterministe et les opérateurs que nous avons utilisés (la sélection, le croisement, la mutation, etc.) permettent de donner des solutions fiables ou même acceptables. Étant donné cette vérification, nous pouvons maintenant valider le comportement de notre algorithme génétique pour l'ordonnancement robuste par l'utilisation de la simulation.

3.3 Validation de l'algorithme génétique pour l'ordonnancement robuste par la simulation

Dans cette section, nous présenterons les résultats des expérimentations de l'algorithme génétique robuste. Notre but est de valider l'algorithme génétique pour l'ordonnancement

robuste par la simulation en utilisant quelques instances présentées dans la section précédente et d'autres instances générées aléatoirement. Au travers de ce travail, nous pourrons également caractériser de manière concrète le comportement de notre algorithme vis-à-vis du concept de "robustesse".

3.3.1 Modélisation des scénarios

Nous utilisons dans cette partie quelques instances utilisées pour l'évaluation de l'algorithme génétique pour l'ordonnancement déterministe. Le nombre de ces instances est égal à 8. On a utilisé seulement la première instance pour chaque type de configuration de taille 10 et 15 *jobs*, dans les 5 étages. Nous générons aussi aléatoirement 3 instances de tailles variant entre 20 et 100 *jobs* dans deux étages. Le tableau 3.17 présente les différentes configurations de chaque instance ainsi que la variation des temps d'exécution. Toutes ces instances représentent les scénarios initiaux (comme il a été noté dans le chapitre 2). Le scénario initial (I) est caractérisé par les temps d'exécution des *jobs* dans chaque étage. A partir de ce scénario initial, nous construisons, dans chaque génération, N autres scénarios perturbés. Cette perturbation concernera les temps d'exécution des *jobs* qui seront générés selon une loi uniforme $\mathcal{U}(P_{jt_I} - \alpha P_{jt_I}; P_{jt_I} + \alpha P_{jt_I})$, avec P_{jt_I} le temps d'exécution du *job* j dans l'étage t du scénario initial I et α le degré d'incertitude.

Instances	Nombre des étages	Nombre des *jobs*	Configuration des machines dans chaque étage	Temps d'exécution
J10c5a2	5	10	2 2 1 2 2	[3, 20]
J10c5b1	5	10	1 2 2 2 2	[3, 20]
J10c5c1	5	10	3 3 2 3 3	[3, 20]
J10c5d1	5	10	3 3 3 3 3	[3, 20]
J15c5a1	5	15	2 2 1 2 2	[3, 20]
J15c5b1	5	15	1 2 2 2 2	[3, 20]
J15c5c1	5	15	3 3 2 3 3	[3, 20]
J15c5d1	5	15	3 3 2 3 1	[3, 20]
2center20Job	2	20	2 2	[5, 25]
2center50Job	2	50	3 3	[5, 25]
2center100Job	2	100	4 4	[5, 25]

TAB. 3.17 – Les instances (scénarios initiaux) utilisées dans les expérimentations

3.3.2 Description de la démarche expérimentale

Dans cette section, nous validons notre algorithme génétique pour l'ordonnancement robuste, présenté dans la section 2.6, par la simulation. Un modèle de simulation a été implémenté sous un logiciel ARENA 12.0. Ce modèle permet de tester et valider la robustesse obtenue par algorithme génétique pour l'ordonnancement robuste. La figure 3.1 présente notre démarche expérimentale de la validation :

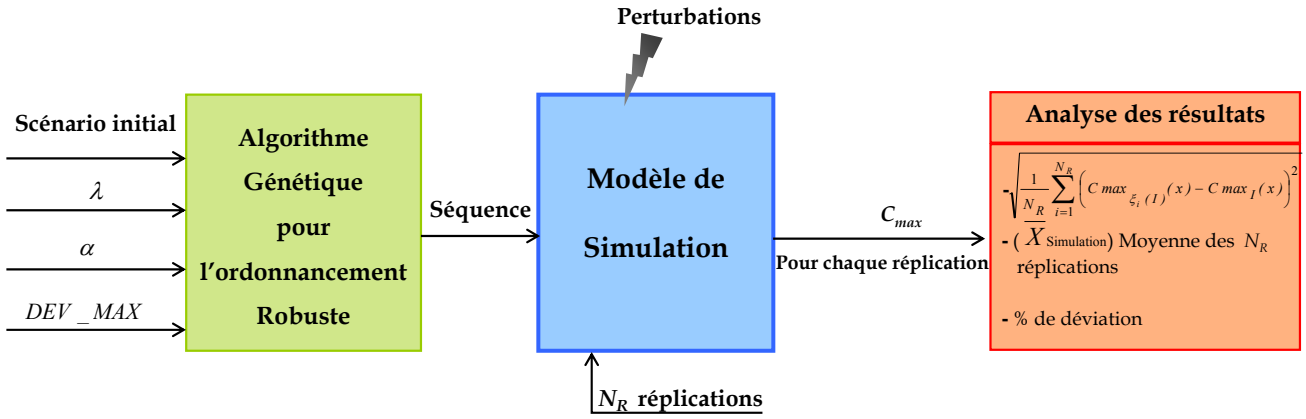

FIG. 3.1 – Démarche expérimentale

Le modèle de simulation correspond à une organisation de type *flow shop* hybride à 5 étages. L'algorithme génétique pour l'ordonnancement robuste prend en considération l'instance initiale et les instances perturbées (obtenues à partir de l'instance initiale en perturbant les temps d'exécution des *jobs* selon la loi uniforme $\mathcal{U}(P_{jt_I} - \alpha P_{jt_I}; P_{jt_I} + \alpha P_{jt_I})$ avec un degré d'incertitude α). Cet algorithme donne, pour chaque valeur de λ, comme donnée de sortie (*output*), une séquence minimisant la fonction d'évaluation robuste $f^r(x)$ décrite dans la section 2.6.1. Le simulateur reçoit en entrée cette séquence des *jobs* obtenue par l'algorithme génétique pour l'ordonnancement robuste.

Dans ce modèle de simulation nous introduisons des perturbations dans chaque instance simulée, traduites par la génération des temps d'exécution des *jobs* à chaque étage aléatoirement selon la même loi uniforme $\mathcal{U}(P_{jt_I} - \alpha P_{jt_I}; P_{jt_I} + \alpha P_{jt_I})$ utilisée dans l'al-

gorithme génétique pour l'ordonnancement robuste. Chaque itération du simulateur est appelée une *réplication*. Le nombre de réplications N_R est égal à 100 ($N_R = 100$). La sortie du modèle de simulation est le *makespan* de chaque réplication (voir figure 3.1). La règle *First In First Out* (FIFO) est utilisée dans le modèle de simulation. Cette règle de séquencement des *jobs* dans les étages k (*avec* $k \geq 2$) est la même règle utilisée dans l'algorithme génétique pour l'ordonnancement robuste. De plus, la règle d'affectation des *jobs* aux différentes machines est la même que la règle d'affectation utilisée dans l'algorithme génétique pour l'ordonnancement robuste. Cette règle est *First Available Machine* (FAM). Le choix de cette règle consiste à affecter les *jobs* sur les machines au plus tôt et permet d'obtenir plus de marge entre la date de fin d'un *job* et la date de début du *job* à affecter sur la même machine. Cette dernière a été modélisée aussi sous ARENA 12.0 en amont de chaque étage (voir le sous-modèle *Waiting for availability S1* dans la figure 3.2). Dès que la simulation des N_R réplications est réalisée, les *makespan* des N_R réplications sont identifiés et analysés pour déterminer la déviation entre le *makespan* du scénario initial et les *makespan* des scénarios perturbés $\left(\sqrt{\frac{1}{N_R}\sum_{i=1}^{N_R}\left(Cmax_{\xi_i(I)}(x) - Cmax_I(x)\right)^2}\right)$, la moyenne des *makespan* des N_R réplications ($\bar{X}_{simulation}$) et le pourcentage de déviation (*% de déviation*) entre la moyenne des *makespan* des N_R réplications obtenu par le simulateur et le *makespan* du scénario initial obtenu par l'algorithme génétique pour l'ordonnancement robuste.

3.3.3 Résultats expérimentaux et discussions

Notre algorithme génétique pour l'ordonnancement robuste est implémenté en Microsoft Visual C++ 6.0, sur un ordinateur Pentium IV avec 1,70 GHz et 512 Mo de RAM. Nous choisissons les mêmes valeurs des paramètres qui sont utilisées dans l'algorithme génétique pour l'ordonnancement déterministe. Ces paramètres sont initialisés par :

- ⋆ une taille de population égale à 200 ;
- ⋆ une probabilité de croisement égale à 90% ;
- ⋆ une probabilité de mutation égale à 20% ;
- ⋆ un critère d'arrêt qui est le nombre de générations, égal à 1000 ($Iter_{Max} = 1.000$) ;

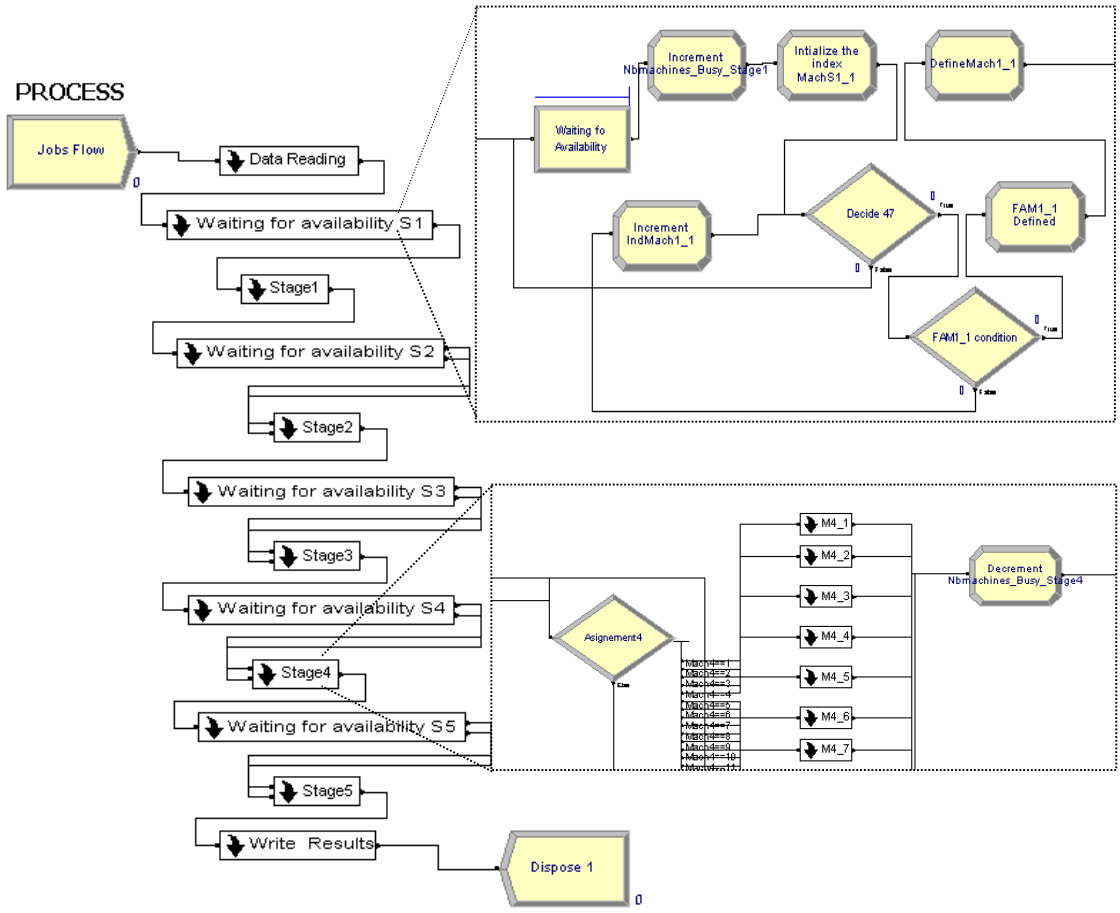

FIG. 3.2 – Modèle de simulation sous ARENA 12.0

Plusieurs degrés d'incertitude, sont fixés pour chaque instance. Ces degrés varient entre 10% et 50%. Pour chaque degré d'incertitude α et pour chaque valeur de λ, l'algorithme génétique pour l'ordonnancement robuste a été exécuté pour obtenir une séquence. Une fois que cette séquence est obtenue, le simulateur reçoit comme *input* cette séquence des *jobs*, 100 réplications ($N_R = 100$) pour chaque type d'instance sont évaluées par cette séquence avec des données modifiées (perturbées).

Les tableaux 3.18 - 3.20 présentent les résultats détaillés de l'algorithme génétique pour l'ordonnancement robuste sur un ensemble d'instances (ou scénarios initiaux) avec des degrés d'incertitude $\alpha = 10\%$, $\alpha = 25\%$ et $\alpha = 50\%$. Dans ces tableaux, la colonne "LB" représente la borne inférieure du scénario initial I. La colonne "λ" contient la valeur du paramètre qui fixe l'importance relative des deux objectifs. La colonne suivante "$Cmax_I$"

représente la valeur du *makespan* du scénario initial obtenu par l'algorithme génétique pour l'ordonnancement robuste. Le nombre de scénarios perturbés utilisé dans chaque itération de l'algorithme génétique pour l'ordonnancement robuste est mentionné dans la colonne "N". La colonne "CPU" indique le temps de calcul de l'algorithme génétique pour l'ordonnancement robuste en secondes. Les deux colonnes suivantes montrent les résultats de la simulation avec les 100 réplications des scénarios perturbés. La séquence obtenue par l'algorithme génétique pour l'ordonnancement robuste est utilisée pour évaluer ces 100 réplications et la valeur de déviation entre le *makespan* du scénario initial et le *makespan* de tous les scénarios perturbés est affichée dans la colonne "$\sqrt{\frac{1}{N_R}\sum_{i=1}^{N_R}\left(Cmax_{\xi_i(I)}(x) - Cmax_I(x)\right)^2}$". La colonne "$\bar{X}_{simulation}$" représente la moyenne des *makespan* pour les 100 réplications. La dernière colonne représente le pourcentage de déviation entre la moyenne des *makespan* des 100 réplications et le *makespan* du scénario initial obtenu par l'algorithme génétique pour l'ordonnancement robuste "$\%\ de\ déviation = \frac{\bar{X}_{simulation} - Cmax_I}{Cmax_I}$". Cette valeur indique la déviation ou l'augmentation de $Cmax$ (en %) des 100 scénarios perturbés, comparée au *makespan* du scénario initial. Par exemple, pour le scénario j10c5a2, si nous minimisons seulement le *makespan* du scénario initial, une modification dans les temps d'exécution (avec un degré d'incertitude α égal 10 %) pourrait mener à une augmentation de 9.13 % du *makespan*, en moyenne. Mais si nous minimisons seulement la déviation entre les scénarios perturbés et le scénario initial, une modification pourrait mener à une augmentation beaucoup plus faible de 0.66% du *makespan*, en moyenne.

Pour $\lambda = 1$, nous favorisons la minimisation du *makespan* initial ($Cmax_I$). Cette valeur peut être obtenue non seulement par l'algorithme génétique pour l'ordonnancement robuste, mais aussi par l'algorithme génétique pour l'ordonnancement déterministe. En effet, dans les deux algorithmes, la fonction d'évaluation est la même (la minimisation du *makespan* du scénario initial ($Cmax_I$)) et les perturbations n'interviennent pas.

Pour $\lambda = 0$, nous favorisons la minimisation de la déviation entre les *makespan* des scénarios perturbés et le scénario initial $\left(\sqrt{\frac{1}{N_R}\sum_{i=1}^{N_R}\left(Cmax_{\xi_i(I)}(x) - Cmax_I(x)\right)^2}\right)$.

Nous remarquons, selon les tableaux 3.18 - 3.20, que plus la valeur de λ diminue, plus le *makespan* du scénario initial augmente et plus la déviation entre les *makespan* des scé-

narios perturbés et le *makespan* du scénario initial diminue.

Instances	LB	λ	$Cmax_I$	N	CPU en(s)	$\sqrt{\frac{1}{N_R}\sum_{i=1}^{N_R}\left(Cmax_{\xi_i(I)}(x) - Cmax_I(x)\right)^2}$	$(\bar{X}_{simulation})$ moyenne des 100 réplications	*% de déviation*
J10c5a2	88	$\lambda = 1$	88	-	0,02	8,83	96,04	**9,13**
		$\lambda = 0,5$	93	10	295,10	4,24	96,29	**3,53**
		$\lambda = 0$	124	20	22,81	3,73	124,83	**0,66**
J10c5b1	130	$\lambda = 1$	130	-	0,01	7,06	136,66	**5,12**
		$\lambda = 0,5$	134	10	374,45	4,34	137,42	**2,55**
		$\lambda = 0$	146	20	189,71	4,01	148,1	**1,43**
J10c5c1	68	$\lambda = 1$	68	-	0,05	7,80	75,37	**10,83**
		$\lambda = 0,5$	76	10	210,04	3,60	78,64	**3,47**
		$\lambda = 0$	89	10	52,78	2,80	89,39	**0,43**
J10c5d1	66	$\lambda = 1$	66	-	0,03	7,92	73,53	**11,40**
		$\lambda = 0,5$	72	10	15,67	2,70	73,01	**1,40**
		$\lambda = 0$	79	20	148,36	1,91	79,74	**0,93**
J15c5a1	178	$\lambda = 1$	178	-	0,02	9,30	186,48	**4,76**
		$\lambda = 0,5$	185	10	26,93	5,13	188,6	**1,94**
		$\lambda = 0$	205	20	319,04	5,07	205,2	**0,09**
J15c5b1	170	$\lambda = 1$	170	-	0,02	9,86	179,31	**5,47**
		$\lambda = 0,5$	175	10	580,31	6,82	181,05	**3,45**
		$\lambda = 0$	188	30	466,17	4,79	190,47	**1,31**
J15c5c1	85	$\lambda = 1$	85	-	0,08	10,82	95,47	**12,31**
		$\lambda = 0,5$	94	10	417	3,85	95,92	**2,04**
		$\lambda = 0$	114	10	94,15	2,83	113,61	**-0,34**
J15c5d1	167	$\lambda = 1$	167	-	0,03	8,63	175,08	**4,83**
		$\lambda = 0,5$	174	10	156,18	6,04	179,1	**2,93**
		$\lambda = 0$	183	20	493,82	4,71	184,22	**0,66**
2centre20job	161	$\lambda = 1$	161	-	0,02	8,97	169,23	**5,11**
		$\lambda = 0,5$	165	10	313,18	4,04	168,3	**2,00**
		$\lambda = 0$	177	30	158,54	3,93	175,23	**-1,00**
2centre50job	256	$\lambda = 1$	256	-	0,2	10,98	266,75	**4,19**
		$\lambda = 0,5$	269	10	211,53	4,66	272,97	**1,47**
		$\lambda = 0$	298	10	156,32	4,27	298,29	**0,09**
2centre100job	402	$\lambda = 1$	402	-	2,2	15,15	416,92	**3,71**
		$\lambda = 0,5$	417	10	970,07	5,86	421,82	**1,15**
		$\lambda = 0$	433	10	1107,31	4,96	435,74	**0,63**

TAB. 3.18 – Résultats expérimentaux avec un degré d'incertitude $\alpha = 10\%$

Instances	LB	λ	$Cmax_I$	N	CPU en(s)	$\sqrt{\frac{1}{N_R}\sum_{i=1}^{N_R}\left(Cmax_{\xi_i(I)}(x) - Cmax_I(x)\right)^2}$	$(\bar{X}_{simulation})$ moyenne des 100 réplications	*% de déviation*
J10c5a2	88	$\lambda = 1$	88	-	0,02	9,63	96,19	**9,30**
		$\lambda = 0,5$	93	30	61,81	5,36	96,07	**3,30**
		$\lambda = 0$	109	20	10,04	4,18	107,85	**-1,05**
J10c5b1	130	$\lambda = 1$	130	-	0,01	7,10	135,4	**4,15**
		$\lambda = 0,5$	134	10	211,12	5,51	136,21	**1,64**
		$\lambda = 0$	139	40	323,18	5,43	140,89	**1,35**
J10c5c1	68	$\lambda = 1$	68	-	0,05	9,88	77,14	**13,44**
		$\lambda = 0,5$	75	60	274,15	4,25	77,56	**3,41**
		$\lambda = 0$	84	30	229,82	3,87	83,6	**-0,47**
J10c5d1	66	$\lambda = 1$	66	-	0,03	8,07	73,39	**11,19**
		$\lambda = 0,5$	72	10	80,57	3,42	72,21	**0,29**
		$\lambda = 0$	78	30	125,26	2,83	76,81	**-1,52**
J15c5a1	178	$\lambda = 1$	178	-	0,02	12,78	188,32	**5,79**
		$\lambda = 0,5$	183	20	241,79	9,28	188,88	**3,21**
		$\lambda = 0$	208	40	143,71	8,42	209,74	**0,80**
J15c5b1	170	$\lambda = 1$	170	-	0,02	9,36	177,41	**4,35**
		$\lambda = 0,5$	171	50	247,01	7,68	176,05	**2,95**
		$\lambda = 0$	182	40	7,54	6,96	185,06	**1,68**
J15c5c1	85	$\lambda = 1$	85	-	0,08	11,95	96,33	**13,32**
		$\lambda = 0,5$	93	10	24,10	3,68	95,62	**2,81**
		$\lambda = 0$	102	40	80,84	3,48	102,26	**0,25**
J15c5d1	167	$\lambda = 1$	167	-	0,03	9,31	174,25	**4,34**
		$\lambda = 0,5$	172	20	681,28	6,57	175,07	**1,78**
		$\lambda = 0$	180	30	8,79	5,98	180,58	**0,32**
2centre20job	161	$\lambda = 1$	161	-	0,02	11,00	170,02	**5,60**
		$\lambda = 0,5$	164	30	299,96	6,27	167,72	**2,26**
		$\lambda = 0$	178	40	268,06	6,18	178,76	**0,42**
2centre50job	256	$\lambda = 1$	256	-	0,18	13,50	268.6	**4,92**
		$\lambda = 0,5$	265	40	252,46	6,89	269,14	**1,56**
		$\lambda = 0$	284	30	964,51	5,90	286,04	**0,71**
2centre100job	402	$\lambda = 1$	402	-	2,2	16,50	417,64	**3,89**
		$\lambda = 0,5$	415	30	887,29	8,64	421,4	**1,54**
		$\lambda = 0$	441	20	57,06	7,68	442,02	**0,23**

TAB. 3.19 – Résultats expérimentaux avec un degré d'incertitude $\alpha = 25\%$

Instances	LB	λ	$Cmax_I$	N	CPU en(s)	$\sqrt{\frac{1}{N_R}\sum_{i=1}^{N_R}\left(Cmax_{\xi_i(I)}(x) - Cmax_I(x)\right)^2}$	$(\bar{X}_{simulation})$ moyenne des 100 réplications	*% de déviation*
J10c5a2	88	$\lambda = 1$	88	-	0,02	13,12	98,9	**12,38**
		$\lambda = 0,5$	93	60	95,95	8,34	97,52	**4,86**
		$\lambda = 0$	111	40	26	8,15	109,95	**-0,94**
J10c5b1	130	$\lambda = 1$	130	-	0,01	11,50	136,12	**4,70**
		$\lambda = 0,5$	130	30	38,64	10,03	135,17	**3,97**
		$\lambda = 0$	137	40	134,82	9,85	138,81	**1,32**
J10c5c1	68	$\lambda = 1$	68	-	0,05	13,12	79,76	**17,29**
		$\lambda = 0,5$	73	40	405,14	7,28	77,61	**6,31**
		$\lambda = 0$	79	10	264,32	6,14	80,66	**2,10**
J10c5d1	66	$\lambda = 1$	66	-	0,03	10,06	74,37	**12,68**
		$\lambda = 0,5$	72	30	330,18	5,28	73,84	**2,55**
		$\lambda = 0$	79	20	137,21	5,02	78,31	**-0,87**
J15c5a1	178	$\lambda = 1$	178	-	0,02	16,52	186,86	**4,97**
		$\lambda = 0,5$	178	30	558,06	14,76	184,8	**3,82**
		$\lambda = 0$	186	20	467,11	13,46	187,97	**1,05**
J15c5b1	170	$\lambda = 1$	170	-	0,02	13,18	177,51	**4,41**
		$\lambda = 0,5$	170	40	195,85	12,24	175,9	**3,47**
		$\lambda = 0$	182	30	512,17	11,71	183,71	**0,93**
J15c5c1	85	$\lambda = 1$	85	-	0,08	13,78	97,68	**14,91**
		$\lambda = 0,5$	91	40	64,37	7,29	96,35	**5,87**
		$\lambda = 0$	100	40	52,59	6,76	99.93	**-0,07**
J15c5d1	167	$\lambda = 1$	167	-	0.03	13,36	172,15	**3,08**
		$\lambda = 0,5$	167	10	507,43	11,81	169,83	**1,69**
		$\lambda = 0$	173	20	130,62	11,43	174,12	**0,64**
2centre20job	161	$\lambda = 1$	161	-	0,02	12,59	168,98	**4,95**
		$\lambda = 0,5$	162	20	289,15	10,51	167,61	**3,46**
		$\lambda = 0$	169	20	322,03	9,37	169,52	**0,30**
2centre50job	256	$\lambda = 1$	256	-	0,2	17,44	270,99	**5,85**
		$\lambda = 0,5$	256	20	300,03	10.47	270,21	**2,35**
		$\lambda = 0$	273	60	362,54	10,07	277,49	**1,64**
2centre100job	402	$\lambda = 1$	402	-	2,14	19,92	418,71	**4,15**
		$\lambda = 0,5$	405	50	867,64	15,98	414,83	**2,42**
		$\lambda = 0$	432	30	946,86	11,98	434,99	**0,69**

TAB. 3.20 – Résultats expérimentaux avec un degré d'incertitude $\alpha = 50\%$

Pour une meilleure compréhension, nous expliquons en détail le cas de l'instance j10c5a2. Pour $\lambda = 1$, l'algorithme génétique pour l'ordonnancement robuste donne une séquence avec une valeur du *makespan* du scénario initial égale à 88, qui est proche (ou même égale) à la borne inférieure LB. Si nous intégrons cette séquence dans le simulateur, sachant que les temps d'exécution peuvent être perturbés avec un degré d'incertitude α égal à 10 %, nous pourrions nous attendre après la simulation à un *makespan* de 96,04 (en moyenne), ce qui correspond à une augmentation du coût de 9,13 %. Cette solution est donc plus efficace.

Si $\lambda = 0$, l'algorithme génétique pour l'ordonnancement robuste donne une séquence avec une valeur de *makespan* du scénario initial égale à 124, ce qui est très loin de la borne inférieure LB. Si maintenant nous utilisons cette séquence pour les scénarios perturbés, la valeur de *makespan* du scénario initial attendue sera égale à 124,83 (en moyenne) qui conduit à une augmentation de 0,66 % seulement. La solution donnée par l'algorithme génétique pour l'ordonnancement robuste est alors plus robuste, mais très peu efficace.

Si maintenant nous cherchons un compromis entre l'efficacité et la robustesse d'un ordonnancement (le cas de $\lambda = 0,5$), l'algorithme génétique pour l'ordonnancement robuste donne une séquence avec un *makespan* du scénario initial égal à 93. Après la simulation, nous nous attendons à un *makespan* de 96,29 (en moyenne) ce qui correspond à une augmentation du coût de 3,53 %. Cette solution est plus robuste que la solution trouvée pour $\lambda = 1$, et plus efficace que $\lambda = 0$.

Nous remarquons que, pour différents degrés d'incertitude, le *% de déviation* diminue quand λ diminue. Cette diminution signifie que nous accorderons plus de confiance à la solution robuste, même si le *makespan* du scénario initial est plus élevé. Quand l'incertitude augmente, les ordonnancements robustes ne se dégraderont pas aussi rapidement que les ordonnancements non robustes. Autrement dit, notre intuition initiale est confirmée : un ordonnancement non robuste deviendra plus rapidement inefficace avec les incertitudes croissantes qu'un ordonnancement robuste, mais un ordonnancement robuste sera moins efficace qu'un ordonnancement non robuste, en l'absence d'incertitudes (les séquences robustes résistent au mieux aux incertitudes que les séquences non robustes). Cette propriété

nous permet de présenter notre proposition comme un outil de gestion des risques : plus il y a d'incertitudes, plus il sera risqué d'accorder du crédit aux séquences non robustes, qui sont malgré tout très séduisantes (car très efficaces) et à l'inverse, moins il sera risqué d'accorder du crédit aux séquences robustes, certes moins efficaces, mais dont la dégradation est mieux maîtrisée.

Si nous comparons les résultats entre $\lambda = 1$ et $\lambda = 0,5$, nous remarquons que leurs moyennes des 100 réplications par simulation ($\bar{X}_{simulation}$) sont approximativement égales. Même pour des degrés d'incertitude plus importants, nous constatons que la moyenne des 100 réplications avec $\lambda = 0.5$ est toujours inférieure à la moyenne des 100 réplications pour $\lambda = 1$. En effet, nous réussissons à trouver des solutions, pour $\lambda = 0,5$, sur lesquelles sont très proches ou même égales à la borne inférieure LB avec une déviation plus faible entre le *makespan* des scénarios perturbés et le *makespan* du scénario initial $\left(\sqrt{\frac{1}{N_R}\sum_{i=1}^{N_R}\left(Cmax_{\xi_i(I)}(x) - Cmax_I(x)\right)^2}\right)$. Cela **permet d'affirmer que notre approche est plus intéressante avec des degrés d'incertitude plus importants**. Cette découverte est observée dans les figures 3.3, 3.4 et 3.5, qui présentent les résultats des 100 réplications par simulation pour l'instance j10c5a2 avec des degrés d'incertitudes $\alpha = 10\%$, $\alpha = 25\%$ et $\alpha = 50\%$.

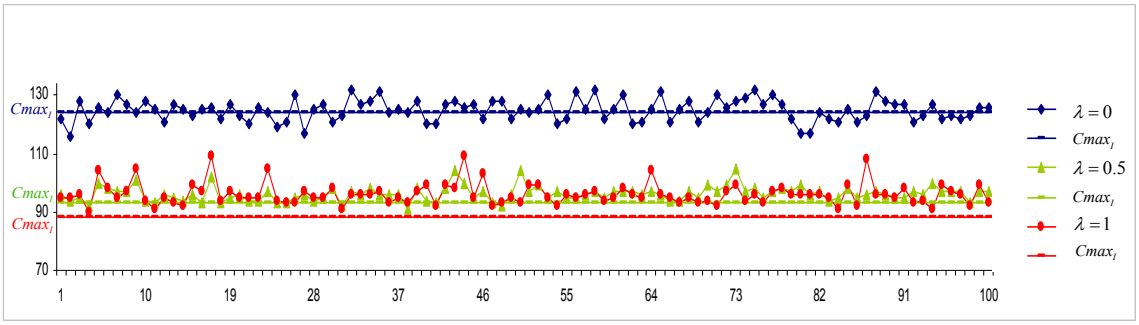

FIG. 3.3 – Exemple des 100 réplications pour le scénario J10c5a2 avec un degré d'incertitude $\alpha = 10\%$

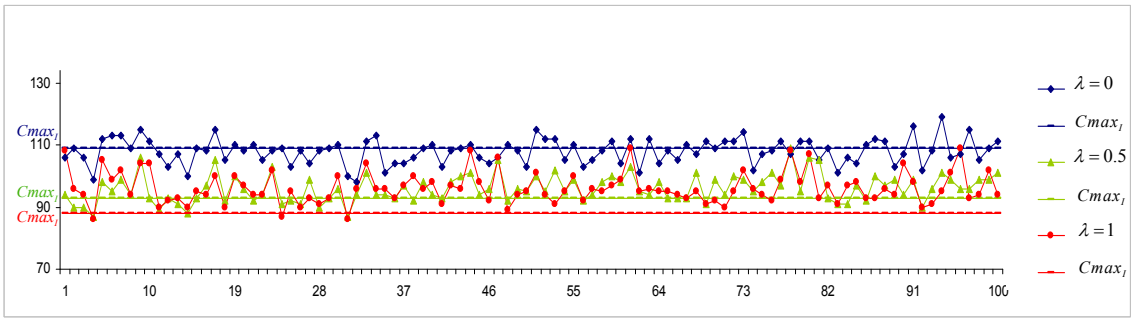

FIG. 3.4 – Exemple des 100 réplications pour le scénario J10c5a2 avec un degré d'incertitude $\alpha = 25\%$

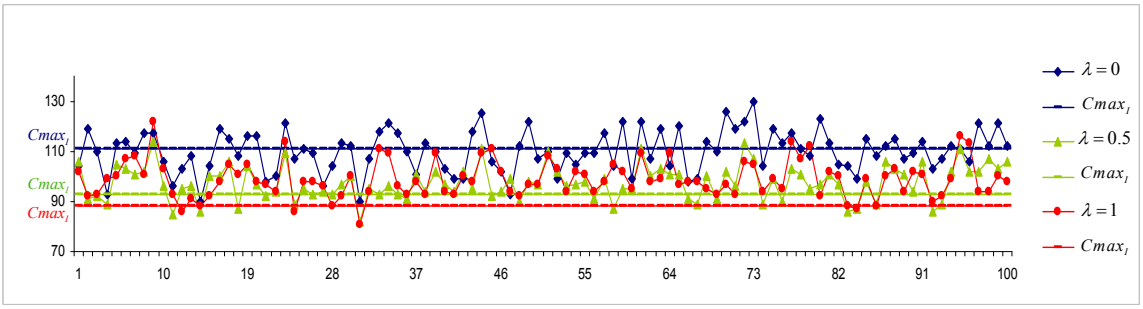

FIG. 3.5 – Exemple des 100 réplications pour le scénario J10c5a2 avec un degré d'incertitude $\alpha = 50\%$

Le tableau 3.21 présente, pour chaque valeur de λ, les pourcentages de déviation moyenne (*% de déviation*) de toutes les instances avec les différents degrés d'incertitude. Nous remarquons que, en *% de déviation*, l'écart entre $\lambda = 1$ et $\lambda = 0.5$ est plus important que l'écart entre $\lambda = 0.5$ et $\lambda = 0$. Les résultats confirment qu'il est toujours meilleur de prendre $\lambda \in [0.5; 1[$ qui correspond au meilleur compromis entre efficacité et robustesse.

	$\alpha = 10\%$	$\alpha = 25\%$	$\alpha = 50\%$
$\lambda = 1$	7.31	7.64	8.52
$\lambda = 0.5$	2.47	2.32	3.78
$\lambda = 0$	0.42	0.24	0.61

TAB. 3.21 – *% de déviation* moyenne de toutes les instances

Néanmoins, les temps de calcul (CPU) de l'algorithme génétique pour l'ordonnancement robuste sont beaucoup plus élevés que les temps de calcul de l'algorithme génétique pour l'ordonnancement déterministe, parce que le calcul de la fonction d'évaluation robuste exige un nombre important de scénarios dans chaque itération. Le temps de calcul moyen, sur toutes les instances, est égal 291,2 secondes (266,18 secondes dans le cas où $\lambda = 0$ et 316,2 secondes lorsque nous cherchons une solution de compromis entre les deux objectifs) pour l'algorithme génétique pour l'ordonnancement robuste, alors que le temps de calcul moyen de l'algorithme génétique pour l'ordonnancement déterministe est égale à 0,2 seconde.

3.4 Conclusion

Dans ce chapitre, nous avons testé l'efficacité de notre algorithme génétique pour l'ordonnancement robuste, dont l'objectif est de minimiser seulement le *makespan* ($Cmax$). Les données utilisées sont déterministes et ne prennent pas en compte les incertitudes sur les temps d'exécution. Cet algorithme a été comparé à deux méthodes de résolution présentées dans la littérature. Les résultats prouvent que cet algorithme permet de donner des solutions acceptables. De plus, notre algorithme converge très rapidement quel que soit le type d'instance (facile ou difficile). Dans un deuxième temps, nous avons validé notre algorithme génétique pour l'ordonnancement robuste par la simulation. Les résultats de simulation montrent que la méthode proposée peut produire de meilleures solutions, dites de compromis entre efficacité et robustesse, pour un degré d'incertitude très élevé. Ceci nous garantira une meilleure maîtrise des risques, pour faire face aux perturbations auxquelles le système de production sera confronté en phase d'exploitation.

Dans le chapitre suivant, nous appliquons cette approche de robustesse dans le domaine de conception et en particulier pour l'aide au dimensionnement du bloc opératoire. Plusieurs autres cadres applicatifs sont envisageables pour notre proposition, qui nous semble ainsi relativement féconde. En particulier, un axe de recherche de l'équipe traite du pilotage temps réel de la production par produits intelligents. L'idée serait d'embarquer notre algorithme génétique robuste dans les produits intelligents, leur permettant ainsi de calculer un ordonnancement robuste en temps réel et de le confronter à ceux des

autres produits, afin d'éviter un problème connu des systèmes répartis ou distribués, celui de la "myopie" à long terme. Une autre application concernerait la proposition d'ordonnancements robustes dans des milieux à risques et en environnement perturbés.

Chapitre 4

Démarche méthodologique d'aide au dimensionnement basé sur des ordonnancements robustes : application aux blocs opératoires

Sommaire

4.1 Introduction

Ces dernières années, les entreprises manufacturières ont été confrontées constamment à la croissance accrue de la concurrence, augmentée par les innovations technologiques, les changements de l'environnement du marché et les évolutions des demandes des clients.

Cette situation critique a nécessité une révision des visions stratégiques et de la pratique des modèles classiques (Sharifi et Zhang, 1999). Il était donc nécessaire de développer et d'améliorer la flexibilité organisationnelle. Dans le passé, la majorité des entreprises ont adopté la restructuration et la ré-ingénierie pour répondre aux exigences des clients et de la demande. Malheureusement ces mesures se sont avérées insuffisantes (Sutclife, 1999). Une nouvelle caractéristique des entreprises a émergé suite à ces évolutions, c'est la notion d'*agilité*. Une entreprise doit être agile, capable de proposer de nouvelles approches, méthodes et outils d'exploitation des systèmes de production lui permettant de réagir rapidement aux changements de marché et des demandes des clients. Cette notion d'agilité trouve son succès dans l'intégration des systèmes d'information et des nouvelles technologies, la gestion et l'exploitation des ressources humaines et matérielles, la maîtrise des processus de production et la mise en place d'une organisation flexible (Ching-Torng *et al.*, 2006).

Pour intégrer ces nouvelles caractéristiques d'agilité et de flexibilité, la conception est une étape essentielle du cycle de vie des systèmes de production, puisqu'elle conditionne son exploitation (Korbaa, 2003). En effet, elle détermine le nombre et la disposition des ressources dans un système de production. Cette étape peut revenir constamment, même en phase d'exploitation, dans le but de réorganiser ou restructurer le système, afin de répondre aux changements du marché et de la demande des clients.

Ajoutée à son importance dans le cycle de vie des systèmes de production, la phase de conception est fortement liée à la phase d'exploitation. Ainsi le système conçu conditionnera les différents ordonnancements.

Dans ce cas, il serait intéressant de baser les choix de dimensionnement sur les résultats des ordonnancements robustes par rapport aux évènements qui pourraient survenir dans le futur. Ainsi, pour être utilisés afin d'aider au dimensionnement, les ordonnancements proposés doivent être robustes, afin d'être sûr du maintien de leur performances face à un ensemble de perturbations. Dans cet objectif, nous proposons une démarche méthodologique basée sur l'ordonnancement robuste, obtenu par l'application de l'algorithme génétique présenté dans le chapitre précédent, couplée avec des techniques de simulation

tenant compte des incertitudes.

L'objectif de ce chapitre est alors de travailler conjointement le dimensionnement et l'ordonnancement dès la phase de conception. A cet égard, après une présentation de la problématique de dimensionnement du système de production dans la section 2 et un état de l'art sur les problèmes de dimensionnement dans la section 3, la section 4 présente l'approche d'optimisation du dimensionnement basée sur les ordonnancements proposés dans un cadre statique (déterministe). Une approche méthodologique générique est proposée dans la section 5 et utilisée dans un des cas applicatifs réels possibles.

4.2 La problématique de dimensionnement de système de production

La réduction de plus en plus rapide des cycles de développement/déploiement des systèmes de production (voir par exemple le taux de rotation des produits en téléphonie mobile, en micro-processeurs, en assemblage informatique, automobile, etc.) rend la phase de conception de ceux-ci de plus en plus cruciale dans le cycle de vie des systèmes de production.

Le but de la conception consiste à déterminer le nombre, les dimensions et la disposition des équipements nécessaires pour chacune des opérations unitaires de production intervenant dans les procédés d'élaboration des différents produits. L'objectif est d'assurer un niveau de production donné, tout en optimisant un ou plusieurs critères technico-économiques.

Cependant, à cause des coûts élevés des équipements composant de tels systèmes, il est fondamental d'analyser, de manière soignée, la conception du système de production, c'est à ce moment là que sont déterminés, par exemple, les types de machines, de convoyeurs, de stockage et les quantités ou tailles de chaque équipement. Dans ce chapitre, on s'intéresse seulement à la détermination du nombre des ressources composant un système de production, ce problème est connu sous le nom du problème de dimensionnement de système de production. Le dimensionnement consiste ici à fixer le nombre de chaque type

de ressources ou d'opérateurs nécessaires pour réaliser une bonne performance en exploitation (Feyzioglu *et al.*, 2005).

La principale problématique du dimensionnement est la suivante : si nous définissons un nombre trop élevé de ressources, des investissements inutiles risquent d'être réalisés. Si, au contraire, nous en définissons un nombre trop faible, le système ne pourra pas produire les quantités désirées dans les délais souhaités, ce qui constitue également un risque de perte financière (pénalité de retard, délais de production trop long comparés à la concurrence, etc.). A ceci, nous pouvons ajouter des contraintes de plus en plus fortes en termes de respect de l'environnement et de développement durable (approche *lean* de la production). Par conséquent, le dimensionnement doit être réalisé au plus juste. Dans ce cadre, notre approche consiste à prendre en compte des contraintes d'exploitation future et potentielle, en phase de dimensionnement.

En plus de l'aspect économique, la phase de conception est fortement liée à la phase d'exploitation. En effet, le nombre des différentes ressources définies, le nombre des ressources choisies, la disposition des ressources, etc. seront des contraintes pour les ordonnancements (respect du nombre maximal de machines, respect de la disponibilité des ressources, respect des contraintes de précédence des *jobs* obtenus par la disposition des ressources et la circulation des flux, etc.).

La question est alors de savoir pourquoi nous ne tenons pas compte de la performance des ordonnancements dès la phase de conception. Il s'agit donc de proposer, dès la phase de conception, un dimensionnement de l'atelier (nombre de chaque type de machines, opérateurs et stocks, etc.) qui est le moins coûteux, mais qui assure l'obtention de bons ordonnancements, respectant les critères de performance de l'activité de production. Ceci revient donc à prendre la décision de dimensionnement conjointement avec la définition d'un ou des ordonnancements potentiels soumis à des objectifs de performances (niveau de stockage, respect des délais, utilisation rationnelle des ressources, etc.). Ce type d'approche est répandu dans d'autres domaines, en particulier, celui de la conception produit, sous différents vocables, notamment le *design for X* : *design for manufacturing*, etc.

4.3 État de l'art sur les approches de résolution des problèmes de dimensionnement des systèmes de production

Plusieurs travaux ont traité les problèmes de conception (plus précisément de dimensionnement) des ateliers de production. A notre connaissance, peu de travaux se sont intéressés au dimensionnement, tout en anticipant les problèmes d'ordonnancement dès la phase de conception.

Nous présentons dans cette partie l'état de l'art des approches de résolution des problèmes de dimensionnement. Ces dernières peuvent être classées en quatre groupes : les approches analytiques, les approches basées sur l'optimisation, les approches basées sur la simulation et les approches basées sur l'optimisation-simulation.

4.3.1 Approches analytiques

L'utilisation des approches analytiques pour les problèmes de dimensionnement a fait l'objet d'un certain nombre d'articles.

Lin et Yang (1996) ont proposé une approche multicritère utilisant la méthode AHP (Analyse Hiérarchique de Procédés) pour le dimensionnement des machines les plus appropriées permettant de fabriquer un type particulier de pièces.

Dans la même problématique, d'autres travaux portent sur des techniques de modélisation ; telles que les réseaux de files d'attente (Govil et Fu, 1999), les réseaux de Petri (Proth et Xie, 1994) et les chaînes de Markov (Askin et Standridge, 1993).

Les approches analytiques souffrent principalement de leur niveau d'abstraction et de leur incapacité à représenter certaines caractéristiques réelles, telles que l'indisponibilité d'une palette de transport, les collisions des véhicules, les machines à vitesses multiples, les conséquences des pannes, etc.

4.3.2 Approches basées sur l'optimisation

De Matta *et al.* (1999) ont développé un algorithme de *Branch and Bound* pour le dimensionnement des ressources dans un atelier flexible dont l'objectif est de minimiser les coûts d'investissement avec des pénalités de retard.

Aubry *et al.* (2005) ont développé une technique de configuration robuste d'un atelier de photolithographie constitué de machines parallèles partiellement multi-fonctions (c'est-à-dire chaque machine est capable de traiter un sous-ensemble de produits). L'objectif considéré est la minimisation des coûts liés à la configuration de l'atelier (machines). Cette configuration permettra de garantir les performances d'équilibre de la charge, tout en tenant compte des incertitudes sur la demande des produits. L'évaluation de ces incertitudes se traduit par la définition et le calcul du rayon de stabilité qui mesure l'amplitude maximale d'une perturbation sur la demande. Cette mesure a amené à considérer le problème de compromis entre la robustesse et le coût de l'atelier. Le critère de robustesse considéré est la maximisation du taux d'occupation des machines (c'est-à-dire que les machines ne doivent jamais être oisives et doivent finir leur production en même temps, malgré les incertitudes sur la demande). Pour résoudre ce type de problème, les auteurs ont utilisé la Procédure par Séparation et Évaluation (PSE).

Dans un autre travail, Aubry *et al.* (2006), ont essayé de généraliser leur idée, tout en traitant le problème du parc de machines parallèles multi-fonctions. L'objectif est de trouver un meilleur compromis entre le coût et la robustesse. De ce fait, le problème traité est un problème d'optimisation bi-critère. Les critères considérés sont le coût de configuration, qui est un critère à minimiser (la minimisation du coût de qualification) et le critère de la maximisation de la robustesse contre les incertitudes sur la demande. Pour résoudre ce problème, une procédure par Séparation et Évaluation (PSE) a été développée. L'objectif de cette méthode est de maximiser la robustesse d'une configuration dans laquelle le coût est fixe.

Plusieurs travaux ont été présentés dans la littérature pour minimiser le nombre de postes de travail. L'idée de base est d'affecter les opérations (d'usinage ou d'assemblage) aux postes de travail, de sorte que le nombre de postes soit minimal, tout en respectant

des contraintes de précédence et d'autres contraintes liées aux processus de fabrication et aux caractéristiques des équipements. Cette affectation détermine la configuration de chaque poste de travail et son coût. Le premier modèle mathématique du problème d'affectation des opérations aux postes de travail sous contraintes de précédence et de temps de cycle a été publié par Salveson en 1955. Plus tard, en 1986, Baybars (1986) a donné, à ce modèle simplifié, le nom *Simple Assembly Line Balancing Problem* (SALBP). Des méthodes exactes et des méthodes approchées ont été développées pour résoudre ce type de problème (pour plus de détails, le lecteur peut se référer à Rekiek *et al.* (2002) et Scholl et Becker (2006).

D'autres extensions pour ce type de problème ont été développées dans la littérature. Par exemple Dolgui *et al.* (1999) ont traité le problème d'optimisation de la configuration des lignes d'usinage à boîtiers multibroches. Par analogie avec SALBP, ce problème a été appelé TLBP (*Transfer Line Balancing Problem*). Une ligne d'usinage comporte plusieurs postes de travail, chacun étant équipé avec des boîtiers multibroches. Un boîtier multibroche exécute plusieurs opérations en parallèle. Lors de la conception en avant-projet, toutes les opérations sont affectées à des boîtiers et des postes de travail, dont l'objectif est de minimiser le nombre de postes et de boîtiers utilisés. Dans un premier temps, deux méthodes exactes ont été développées pour sa résolution. La première consiste à transformer le problème initial en un problème de recherche du plus court chemin sous contraintes dans un graphe spécialement construit (Dolgui *et al.*, 1999). La deuxième approche utilise la modélisation du problème sous forme d'un programme linéaire en variables mixtes (*Mixed Integer Programming*), la résolution de ce programme est faite à l'aide d'un logiciel d'optimisation Cplex (Dolgui *et al.*, 2006). La dernière méthode est une approche mixte développée par Guchinskaya et Dolgui (2009). Cette approche comporte les trois étapes suivantes : la résolution heuristique pour la génération des solutions initiales, la décomposition en sous-problèmes et, enfin, la résolution exacte de ces derniers.

4.3.3 Approches basées sur la simulation

La simulation permet la considération de différents évènements aléatoires dans les systèmes de production, tels que l'arrivée des ordres de fabrication, l'arrivée de pièces ou de matières premières, le temps d'inspection, le temps de chargement/déchargement,

le temps d'installation, etc. On peut aussi observer le comportement dynamique d'un système en fonctionnement (par exemple les pannes). Ces qualités et ces possibilités de modélisation d'un éventail de systèmes complexes ont rendu les techniques de simulation parmi les techniques les plus utilisées et appliquées pour résoudre les problèmes de dimensionnement.

McHaney et Douglas (1997) ont utilisé la simulation et l'analyse de régression (avec la conception des expériences) pour déterminer le nombre de véhicules autoguidés (Automated Guided Vehicule), et d'opérateurs. Czarneck *et al.* (1997) ont simulé l'impact de l'ajout d'un nouvel équipement au système de production dans le but de satisfaire la demande. Williams et Gevaert (1997) ont étudié par la simulation l'effet du nombre de palettes par rapport aux taux de production, dans le cas d'une industrie automobile.

Dumbrava (1997) a présenté l'avantage de la simulation dans la conception des ateliers flexibles (*Flexible Management System*). Le nombre de machines dans chaque groupe est déterminé pour réduire la capacité des stocks et la durée des processus et pour obtenir un bon compromis entre le nombre de machines et la productivité du système.

Law et Kelton (2000) ont modélisé et identifié le nombre correct de conteneurs et d'élévateurs dans un atelier d'assemblage, pour réduire les investissements et les coûts d'inventaires.

Patel *et al.* (2002) ont simulé une industrie de fabrication de moteurs de véhicules et ont changé, un par un, le nombre d'opérateurs, les stations mécaniques et les stations de peinture pour déterminer leur taille optimale et augmenter le taux de production.

4.3.4 Approches basées sur l'optimisation-simulation

La simulation n'est qu'un outil d'évaluation de la performance d'un système en fonction de ses paramètres. Il faut donc utiliser la simulation conjointement à d'autres méthodes, afin de rechercher les solutions optimales (ou acceptables). Ce couplage optimisation-simulation est connu sous le terme d'optimisation via simulation, où le module d'optimisation propose successivement des solutions, que l'on espère de plus en plus performantes,

au module de simulation. Des méthodes intégrant des techniques d'optimisation et de simulation ont été appliquées à divers problèmes de dimensionnement.

Choon (1991) a déterminé le nombre de chaque type de machine, le nombre de véhicules autoguidés et la capacité des stocks en amont et en aval de chaque machine. La méthode de résolution utilisée est la procédure de recherche aléatoire couplée avec la simulation.

Spieckermann *et al.* (2000) ont essayé de trouver une disposition efficace satisfaisant le taux de production désiré pour une compagnie de fabrication de voitures, en réduisant le nombre des stocks et la durée de cycle. Ils ont employé les algorithmes génétiques et le recuit simulé dans leur modèle de simulation.

Feyzioglu *et al.* (2005) ont minimisé le nombre de ressources (machines) sous contraintes des critères de performance. Le problème a été défini comme un problème d'optimisation stochastique multicritère basé sur la simulation. L'approche proposée utilise un modèle de simulation, afin d'éviter des hypothèses restrictives et de prendre en compte la dimension stochastique du problème. La démarche retenue consiste à déterminer le nombre minimal de ressources de chaque type (le nombre de machines identiques dans un îlot) permettant d'obtenir des performances ayant une probabilité suffisante d'atteindre les performances qui sont stipulées dans le cahier des charges. Pour formaliser analytiquement les contraintes de ce problème multiobjectif, les auteurs ont utilisé une méthode d'échantillonnage par "*bootstrap*", permettant d'estimer le quantile d'une variable de performance du système en utilisant la simulation, afin de prendre en compte la nature stochastique du système étudié. Une méthode de métamodélisation par régression, destinée à exprimer, sous forme de polynôme, le quantile de la variable de performance, à partir des quantités de ressource proposées, permet ainsi de déduire les contraintes du problème d'optimisation. En conséquence, au lieu d'optimiser les mesures de performance (satisfaire la demande), les auteurs l'ont abordé comme des contraintes à satisfaire. Les contraintes du problème doivent être vérifiées par simulation. Il est donc nécessaire de disposer d'un modèle de simulation approprié, dont les entrées sont les variables à dimensionner et les sorties sont les mesures de performance utilisées dans le cahier de charges. Comme l'expérimentation

nécessite d'examiner diverses solutions dans un espace à plusieurs dimensions, les auteurs ont utilisé les plans d'expériences. La formulation des contraintes du modèle impose d'exprimer les relations qui relient les variables de décision aux quantiles de façon analytique. Pour cela, une approche de métamodélisation est utilisée. En général, les modèles de simulation cherchent à approcher les performances d'un système, tandis que les métamodèles de simulation tentent d'approcher les résultats du modèle de simulation. La résolution du problème d'optimisation multiobjectif est faite par deux méthodes : la méthode lexicographique et la programmation mathématique avec buts (*goal programming*).

Les algorithmes évolutionnaires ont prouvé leur efficacité pour résoudre des problèmes d'optimisation via la simulation. Paris *et al.* (1996) présentent une application pour la conception d'un système multiproduit juste à temps. L'algorithme évolutionnaire permet de chercher le nombre de *kanban* dans chaque étape pour chaque produit et les règles de priorité à utiliser dans chaque station. La fonction d'évaluation *fitness* est une fonction pondérée incorporant simultanément le retard de production et le nombre moyen des travaux en cours.

Pierreval et Tautou (1997) ont déterminé le nombre de ressources d'un atelier de production de pots de yaourt en plastique en utilisant la simulation de flux couplée à un algorithme génétique.

Pierreval et Paris (2000) ont proposé un algorithme évolutionnaire distribué pour l'optimisation-simulation d'un système manufacturier. Chaque processeur utilise son propre algorithme évolutionnaire, appelé "Algorithme Evolutionnaire Local" (AEL) qui dirige l'évolution de sa propre population et exécute ses propres runs de simulation. Un opérateur de migration est utilisé pour échanger des individus entre populations selon un taux de migration donné. Cet opérateur va permettre de propager les bonnes solutions afin d'aider les AEL piégés dans des optima locaux. Ce système est utilisé pour choisir la taille de lot et déterminer une stratégie d'allocation de transporteur dans un problème de type *flow shop*.

Pierreval et Paris (2003) ont utilisé un algorithme évolutionnaire pour dimensionner

le nombre de machines dans chaque cellule minimisant le temps de fabrication d'un plan de production donné sous une contrainte budgétaire.

Cheikhrouhou *et al.* (2002) présentent une méthode de dimensionnement des lignes de transfert dans un environnement stochastique basée sur une approche d'analyse de perturbation finie pour le calcul de sensibilité des mesures de performance, la technique adoptée intègre ces informations dans une procédure d'optimisation multidimensionnelle pour déterminer la meilleure configuration des paramètres du système. D'un côté, la simulation permet de donner une estimation de la mesure de performance moyenne. De l'autre côté, l'apport des techniques de recherche basées sur l'analyse de perturbation consiste à déterminer une direction de l'exploration en calculant les gradients de la mesure de performance par rapport aux paramètres du système. Le noyau de la technique est un algorithme stochastique auto-régressif basé sur un calcul de gradient du taux de production par analyse de perturbation fini. La problématique du dimensionnement traitée est la recherche de l'allocation optimale des capacités de stockage sur l'ensemble des emplacements physiques disponibles et des temps de service sur les machines de transfert, de telle sorte que la configuration globale maximise le taux de production de la ligne.

Nous avons remarqué, dans tous ces travaux, l'absence de considération des problèmes d'ordonnancement. Les techniques et les approches proposées ont été élaborées pour définir le nombre de machines, dimensionner les stocks, définir les temps de cycles, etc. L'ordonnancement sera considéré seulement dans la phase d'exploitation ayant comme données et contraintes le dimensionnement et la configuration déterminés en phase de conception.

4.4 Approche statique

Nous présentons une formalisation mathématique du problème de dimensionnement basé sur les ordonnancements, en vue de son optimisation (Chaari *et al.*, 2008a). Nous nous intéressons à une organisation de type *flow shop* hybride à plusieurs étages. Un programme linéaire à variables mixtes a été élaboré, permettant de minimiser conjointement la date d'achèvement des travaux ($Cmax$) et le nombre de machines dans chaque étage pour un

ensemble de tâches types de l'activité prévisionnelle. Différents jeux de poids sont attribués à chacun de ces sous-objectifs. L'objectif de ce programme est de déterminer un compromis entre le nombre de machines à affecter dans chaque étage et la date d'achèvement des travaux ($Cmax$).

4.4.1 Paramètres et variables

– ***Paramètres et données :***

t	Indice de l'étage $t = 1, ..., k$;
M^t_{Max}	Nombre maximal de machines dans chaque étage t ;
i	Indice de machine, $i = 1, ..., M^t_{Max}$;
j, l	Indice de *job* $j, l = 1, ..., n$;
$Cout_{it}$	Coût de la machine i à l'étage t ;
P_{jt}	Temps d'exécution du *job* j dans l'étage t ;
λ	Un poids relatif aux deux objectifs $\in [0, 1]$;
B	Grande valeur positive ;

– ***Variables :***

C_{jt}	Temps d'achèvement du *job* j dans l'étage t ;
$Cmax$	*makespan*.

$$X_{ijlt} = \begin{cases} 1 & \text{si le } job\ j \text{ est exécuté immédiatement avant le } job\ l \\ & \text{sur la machine } i \text{ à l'étage } t\,; \\ 0 & \text{si non}\,; \end{cases}$$

$$X_{i0lt} = \begin{cases} 1 & \text{si le } job\ j \text{ est exécuté le premier sur la machine } i \text{ à l'étage } t\,; \\ 0 & \text{si non}\,; \end{cases}$$

$$X_{ij(n+1)t} = \begin{cases} 1 & \text{si le } job\ j \text{ est exécuté le dernier sur la machine } i \text{ à l'étage } t\,; \\ 0 & \text{si non}\,; \end{cases}$$

$$Y_{it} = \begin{cases} 1 & \text{si la machine } i \text{ est utilisée dans l'étage } t\,; \\ 0 & \text{si non}\,; \end{cases}$$

4.4.2 Modèle mathématique à variables mixtes

Ce problème est formulé comme suit :

$$Minimiser[\lambda \times Ecart(Cmax) + (1-\lambda) \times Ecart^{'}(\sum_{i=1}^{M_{max}^{t}} \sum_{t=1}^{k} Cout_{it} * Y_{it})] \tag{4.1}$$

$$\sum_{i=1}^{M_{max}^{t}} \sum_{j=0}^{n} X_{ijlt} = 1 \qquad \forall t = 1,2,...,k \qquad \forall l = 1,2,...,n \tag{4.2}$$

$$\sum_{i=1}^{M_{max}^{t}} \sum_{l=1}^{n+1} X_{ijlt} = 1 \qquad \forall t = 1,2,...,k \qquad \forall j = 1,2,...,n \tag{4.3}$$

$$\sum_{l=1}^{n} X_{i0lt} = Y_{it} \qquad \forall t = 1,2,...,k \qquad \forall i = 1,2,...,M_{max}^{t} \tag{4.4}$$

$$\sum_{j=1}^{n} X_{ij(n+1)t} = Y_{it} \qquad \forall t = 1,2,...,k \qquad \forall i = 1,2,...,M_{max}^{t} \tag{4.5}$$

$$\sum_{j=0}^{n} X_{ijlt} = \sum_{j=1}^{(n+1)} X_{iljt} \qquad \forall t = 1,2,...,k \quad \forall i = 1,2,...,M_{max}^{t} \tag{4.6}$$

$$\forall l = 1,2,...,n$$

$$X_{ijlt} \leq Y_{it} \qquad \forall t = 1,2,...,k \qquad \forall i = 1,2,...,M_{max}^{t} \tag{4.7}$$

$$\forall l = 1,2,...,n \qquad \forall j = 1,2,...,n$$

$$X_{ijjt} = 0 \qquad \forall t = 1,2,...,k \quad \forall i = 1,2,...,M_{max}^{t} \tag{4.8}$$

$$\forall j = 1,2,...,n$$

$$C_{lt} \geq C_{jt} + \sum_{i=1}^{M_{max}^{t}} X_{ijlt} P_{lt} + B[\sum_{i=1}^{M_{max}^{t}} X_{ijlt} - 1] \quad \forall t,l,j \quad j \neq l \tag{4.9}$$

$$C_{lt} \geq C_{l(t-1)} + P_{lt} \qquad \forall t = 2,...,k \quad \forall l = 1,2,...,n \tag{4.10}$$

$$C_j^k \leq Cmax \qquad \forall j = 1, ..., n \tag{4.11}$$

$$X_{ijlt} \in \{0, 1\} \quad \forall i = 1, 2, ..., M_{max}^t \qquad \forall j = 1, 2, ..., n \tag{4.12}$$

$$\forall l = 1, 2, ..., n \qquad \forall t = 1, 2, ..., k$$

$$Y_{it} \in \{0, 1\} \quad \forall i = 1, 2, ..., M_{max}^t \qquad \forall t = 1, 2, ..., k \tag{4.13}$$

$$C_{jt} \geq 0 \qquad \forall j = 1, 2, ..., n \qquad \forall t = 1, 2, ..., k \tag{4.14}$$

4.4.2.1 Commentaires

La fonction objectif consiste à minimiser une somme pondérée de l'écart entre le *makespan* et la meilleure date d'achèvement des *jobs* et de l'écart entre le coût du dimensionnement obtenu défini par la somme des coûts des machines parallèles dans chaque étage et le dimensionnement le moins coûteux (4.1). Afin d'avoir des échelles connues aux deux critères, nous avons normalisé les valeurs correspondantes à ces unités en utilisant la fonction d'utilité (voir section 4.4.2.2). Deux approches peuvent être envisageables. Une première, qui consiste à permettre de combiner linéairement les deux critères et une deuxième, qui permet à l'utilisateur de borner un des deux critères et de minimiser l'autre. Dans cette thèse, nous avons choisi la première approche et l'implémentation de la deuxième est envisageable dans des travaux futurs.

L'ensemble des contraintes 4.2 et 4.3 assurent que chaque *job* est affecté à une seule machine dans chaque étage. Les contraintes 4.4 assurent qu'un seul *job* est exécuté en première position sur chaque machine à chaque étage (on ne peut pas exécuter deux *jobs* en même temps sur une seule machine). Les contraintes 4.5 indiquent qu'un seul *job* est exécuté en dernière position sur chaque machine à chaque étage. Les contraintes 4.6 imposent une séquence de *jobs* dans chaque machine pour chaque étage. Les contraintes 4.7 permettent de maintenir que le *job* j affecté à la machine i à un étage donné est exécuté sur cette même machine à cet étage. Les contraintes 4.8 indiquent que le *job* qui a fini son exécution sur une machine dans un même étage ne peut pas être ré-exécuté dans une autre machine à ce même étage.

L'ensemble des contraintes 4.9-4.11 déterminent le temps d'achèvement de chaque *job*. Les contraintes 4.9 indiquent que si le *job* j est exécuté avant le *job* l par une même machine dans un étage particulier, le *job* j complète son exécution avant que le *job* l ne commence. Les contraintes 4.10 permettent d'assurer que chaque *job* ne peut commencer son exécution dans l'étage $1+t$ que lorsqu'il s'achève dans l'étage t. Les contraintes 4.11 assurent que le $Cmax$ doit être supérieur au temps d'achèvement de chaque *job* j dans le dernier étage k. Les contraintes 4.12 et 4.13 sont des variables binaires de décision qui peuvent prendre la valeur 0 ou 1. Les contraintes 4.14 imposent que la date d'achèvement des *jobs* soit positive.

4.4.2.2 Définition de la fonction *Ecart*

Nous définissons tout d'abord deux bornes pour les valeurs de $Cmax$: la borne inférieure (BI) et la borne supérieure (BS).

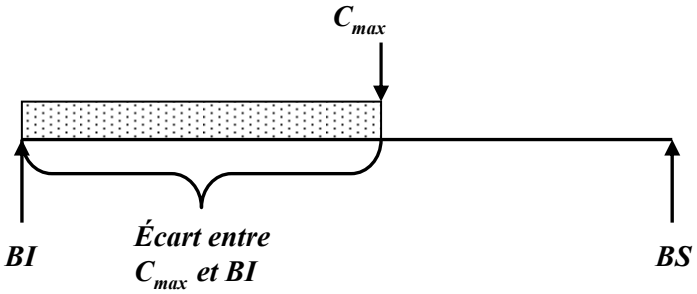

FIG. 4.1 – Intervalle des valeurs de $Cmax$

Plus $Cmax$ se rapproche de BI, plus on a un ordonnancement efficace. Et par conséquent un $Cmax$ minimal, quand l'écart entre BI par rapport à l'intervalle des valeurs de $Cmax$. Donc l'écart de $Cmax$ est défini par :

$$Ecart(Cmax) = \frac{Cmax - BI}{BS - BI}$$

La borne inférieure BI est la valeur de $Cmax$ minimale que nous pouvons obtenir dans le meilleur des cas. Dans notre cas, c'est le $Cmax$ obtenu quand le nombre de machines

correspond au nombre de *jobs* à ordonnancer.

$$BI = \underset{j=1,\ldots,n}{Max} \sum_{t=1}^{k} P_{jt}$$

La borne supérieure BS est la valeur de $Cmax$ maximale que nous pouvons obtenir dans le pire des cas. Dans notre cas, c'est le $Cmax$ obtenu quand nous n'avons qu'une seule machine par étage.

$$BS = \sum_{j=1}^{n} \sum_{t=1}^{k} P_{jt}$$

Propriété : $Ecart(Cmax) \in [0, 1]$

L'utilité de $Cmax$ est donc définie par :

$$U : \mathbb{N} \longrightarrow [0, 1]$$
$$Cmax \longmapsto 1 - Ecart(Cmax)$$

Donc :

$$U(BI) = 1 \quad \text{et} \quad Ecart(BI) = 0$$

Et

$$U(BS) = 0 \quad \text{et} \quad Ecart(BS) = 1$$

4.4.2.3 Définition de la fonction $Ecart'$

Cette fonction concerne le dimensionnement de l'atelier, c'est-à-dire le nombre de machines dans chaque étage. Nous définissons un meilleur dimensionnement comme celui qui comporte le minimum de machines ; dans notre cas, il est égal à k (une machine par étage). Le plus coûteux dimensionnement est celui qui comporte autant de machines que de *jobs* par étage, donc n machines par étage. Plus la somme des machines dans les k étages est petite, moins le dimensionnement obtenu est coûteux. La fonction d'écart d'un dimensionnement par rapport au dimensionnement optimal est définie par :

$$Ecart'(\sum_{i=1}^{M_{max}^t}\sum_{t=1}^{k} Cout_{it}Y_{it}) = \frac{\sum_{i=1}^{M_{max}^t}\sum_{t=1}^{k} Cout_{it}Y_{it} - \sum_{t=1}^{k} \min_{i=1..M_{max}^t}(Cout_{it})}{\sum_{i=1}^{M_{max}^t}\sum_{t=1}^{k}(Cout_{it}) - \sum_{t=1}^{k} \min_{i=1..M_{max}^t}(Cout_{it})}$$

Propriété : $Ecart'(\sum_{i=1}^{M_{max}^t}\sum_{t=1}^{k} Cout_{it}Y_{it})) \in [0,1]$

L'utilité de dimensionnement est donc définie par :

$$U' : \mathbb{N} \longrightarrow [0,1]$$
$$x \longmapsto 1 - Ecart'(x)$$

Donc :

$$U'(\sum_{t=1}^{k} \min_{i=1..M_{max}^t}(Cout_{it})) = 1 \quad \text{et} \quad Ecart'(\sum_{t=1}^{k} \min_{i=1..M_{max}^t}(Cout_{it})) = 0$$

Et

$$U'(\sum_{i=1}^{M_{max}^t}\sum_{t=1}^{k}(Cout_{it})) = 0 \quad \text{et} \quad Ecart'(\sum_{i=1}^{M_{max}^t}\sum_{t=1}^{k}(Cout_{it})) = 1$$

4.4.3 Expérimentations et résultats

Le programme linéaire présenté ci-dessus a été implémenté en C++ et résolu par le solveur CPLEX 9.1. Les instances ont été générées aléatoirement. Nous avons testé des instances avec un nombre d'étages égal à 2, nombre de *jobs* types représentatifs des futures demandes, pour le *flow shop* hybride étudié, variant de [5 ; 7 ; 8 ; 10]. Les temps opératoires prévisionnels ont été générés aléatoirement dans l'intervalle [3, 14] unités de temps. Par ailleurs, nous avons considéré que toutes les machines dans tous les étages ont le même coût.

Nous présentons dans le tableau 4.1 le nombre de contraintes et de variables pour chaque instance, λ, la fonction objectif obtenue et le temps de calcul CPU (en secondes).

Pour $\lambda = 0$ nous favorisons la minimisation du nombre de machines disponibles pour chaque étage et pour $\lambda = 1$ nous favorisons la minimisation de $Cmax$.

Nous remarquons, d'après le tableau 4.1, que plus on s'approche d'un compromis entre la minimisation de $Cmax$ et la minimisation du coût du dimensionnement, plus la valeur

Nombre d'étages	Nombre de Jobs	Nombre de variables	Nombre de contraintes	λ	**Objectif**	CPU(s)
2	5	373	550	0	**0**	0.305
				0,1	**0,0372**	69,06
				0,5	**0,1638**	1042,47
				0,9	**0,05**	416,875
				1	**0**	0,461
2	7	913	1246	0	**0**	0,4424
				0,1	**0,042**	41921,5
				0,5	**0,1435**	42071,3
				0,9	**0,04167**	9519,6
				1	**0**	10,66
2	8	1315	1744	0	**0**	15,32
				0,1	**0,1487**	43200
				0,5	**0,2505**	43200
				0,9	**0,0785**	43200
				1	**0**	107,531
2	10	2443	3100	0	**0**	1,829
				0,1	**0,049**	43200
				0,5	**0,1707**	43200
				0,9	**0,0333**	43200
				1	**0**	8400,03

TAB. 4.1 – Résultats des expérimentations

de l'objectif augmente pour atteindre un maximum qui correspond à $\lambda = 0,5$. Ceci montre qu'un compromis entre les deux objectifs détériore la fonction objectif globale par rapport aux valeurs extrêmes de λ ($\lambda = 0$ et $\lambda = 1$).

Le temps de calcul devient plus élevé également, quand on se rapproche de la valeur de $\lambda = 0,5$ (compromis des deux objectifs). Ce temps peut atteindre les 43 000 secondes. Pour certaines expérimentations, nous étions obligés de limiter le temps de calcul, afin de remédier aux problèmes de mémoire et d'obtenir des solutions.

Dans la figure 4.2 nous présentons un exemple d'évolution de la fonction objectif en fonction de différentes valeurs de λ (allant de 0 à 1, avec un pas égal à 0,1). Les résultats présentés dans cette figure confirment le constat réalisé suite au tableau global des résultats sur l'impact de la valeur de λ sur la valeur de l'objectif obtenu.

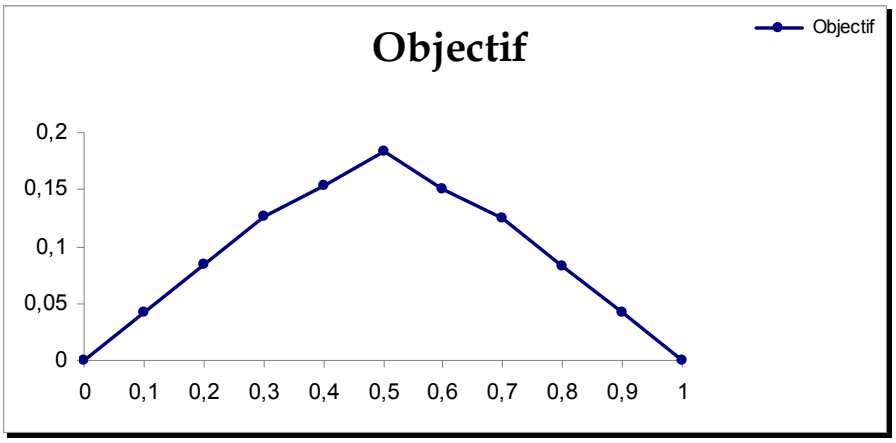

FIG. 4.2 – Évaluation de l'objectif en fonction de λ

La figure 4.3 montre que la fonction d'utilité de $Cmax$ croît en fonction des valeurs de λ. Ceci résulte de fait que, plus λ augmente, plus la priorité est donnée à la minimisation de $Cmax$ et donc il s'approche de BI. Inversement pour le nombre total de machines (dimensionnement), plus λ augmente, plus le coût du dimensionnement de machines augmente, et donc le coût du dimensionnement s'éloigne de sa valeur minimale.

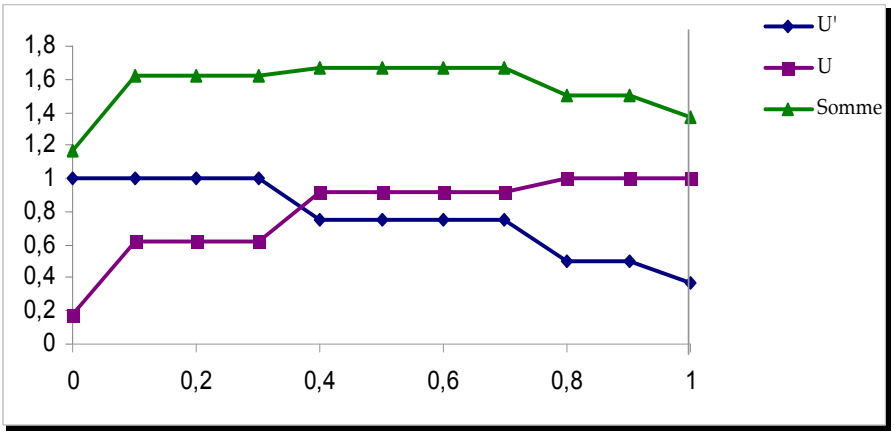

FIG. 4.3 – Évolution des fonctions d'utilité en fonction de λ

Nous avons défini des fonctions d'écart correspondant aux écarts entre la valeur de $Cmax$ trouvée et la borne inférieure et entre le coût total de machines dans les étages et leur somme de coûts minimale. Nous pouvons alors associer un coût à ces écarts. En effet, plus l'écart est élevé, plus c'est coûteux. C'est-à-dire, dans le cas du dimensionnement

plus on s'éloigne du dimensionnement le moins coûteux (la machine la moins coûteuse par étage), plus on doit payer de l'argent et dans le cas de $Cmax$ plus on s'éloigne de la meilleure date d'achèvement (BI), plus ça coûte de l'argent. Donc le coût total d'une solution, est la somme des coûts engendrés par les écarts de $Cmax$ et les coûts engendrés par les écarts du dimensionnement. En d'autres termes, les écarts représentent l'évolution des coûts totaux d'une solution. Donc l'interprétation des courbes des écarts est associée à l'interprétation du coût total d'une solution.

Nous remarquons, d'après la figure 4.4, que plus λ augmente plus l'écart de $Cmax$ par rapport à sa valeur minimale diminue et inversement pour le dimensionnement. L'écart global atteint sa valeur minimale aux valeurs de λ proches du compromis. Ceci nous montre que le compromis des deux objectifs minimise le coût global engendré par ce choix, par rapport aux cas où l'on favorise l'une ou l'autre des minimisations.

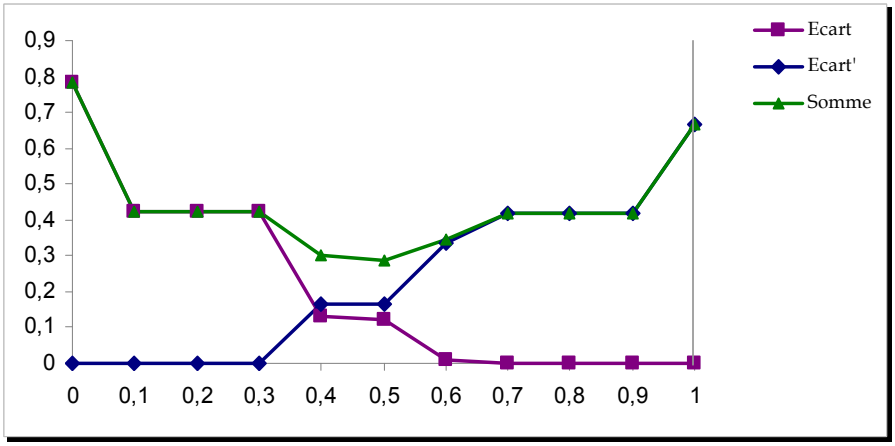

FIG. 4.4 – Évolution des écarts en fonction de λ

Au vu des résultats obtenus, nous pouvons conclure que cette approche peut servir à obtenir une solution exacte à un problème d'ordonnancement et de dimensionnement d'un atelier organisé en flow shop hybride. Malheureusement, cette approche trouve ses limites, pour deux principales raisons. La première, c'est le temps de calcul, important, dès que nous essayons de chercher une solution de compromis. La deuxième, la plus importante : cette approche n'est pas générique, elle s'intéresse à un type d'organisation particulier, celui du problème d'ordonnancement de type *flow shop* hybride. L'implémentation et le

test d'une deuxième approche, qui est une variante de celle-ci, peut être envisageable, dans laquelle l'utilisateur est capable de borner un des deux critères et de minimiser l'autre.

Pour remédier aux inconvénients de l'approche précédente, nous développons dans la section suivante une approche intégrant des techniques d'optimisation et de simulation dans le cadre d'une démarche méthodologique. Elle aura pour objectif de dimensionner un atelier de production, en tenant compte de la performance des ordonnancements, dès la phase de conception. Elle doit aboutir à un dimensionnement basé sur des ordonnancements robustes.

4.5 Approche méthodologique

Nous présentons, dans cette section, l'architecture globale de notre démarche méthodologique, dont le but est de travailler conjointement le dimensionnement et l'ordonnancement dès la phase de conception, en tenant compte des incertitudes qui peuvent perturber le plan de production.

4.5.1 Démarche méthodologique et outil d'aide au dimensionnement

Nous présentons dans la figure 4.5 l'architecture globale de notre démarche méthodologique. L'objectif de cette architecture est d'aider le décideur à prendre des décisions sur la conception de son système de production. L'idée sera d'intégrer et utiliser plusieurs approches, dans le but d'avoir une démarche hybride. Les plans d'expériences sont utilisés pour définir des scénarios de production intégrant efficacement l'ensemble des paramètres et des données d'entrée. Ces scénarios définissent alors les expériences à réaliser. Les paramètres sont appelés les "facteurs contrôlés", qui illustrent les décisions prises par le décideur et qui vont être testées et simulées. Ces scénarios de production vont être utilisés en entrée du module d'ordonnancement robuste, qui calculera la séquence de réalisation des *jobs* la plus robuste. La simulation permettra par la suite de simuler le fonctionnement de ce système ainsi conçu, en se basant sur la séquence de réalisation robuste. Les résultats de simulation (indicateurs de performances) sont alors analysés et comparés selon les expériences. La fonction reliant les indicateurs de performances et les

facteurs contrôlés (paramètres) est ainsi calculée. Chaque indicateur de performance est par la suite comparé aux critères de performance ou aux objectifs fixés en entrée par le décideur. Le décideur va choisir la meilleure conception, suite à cette comparaison, selon ses objectifs. Cette architecture a été conçue pour être générique et appliquée à n'importe quel type d'organisation (*flow shop* hybride, Machines parallèles, lignes de transfert, etc.).

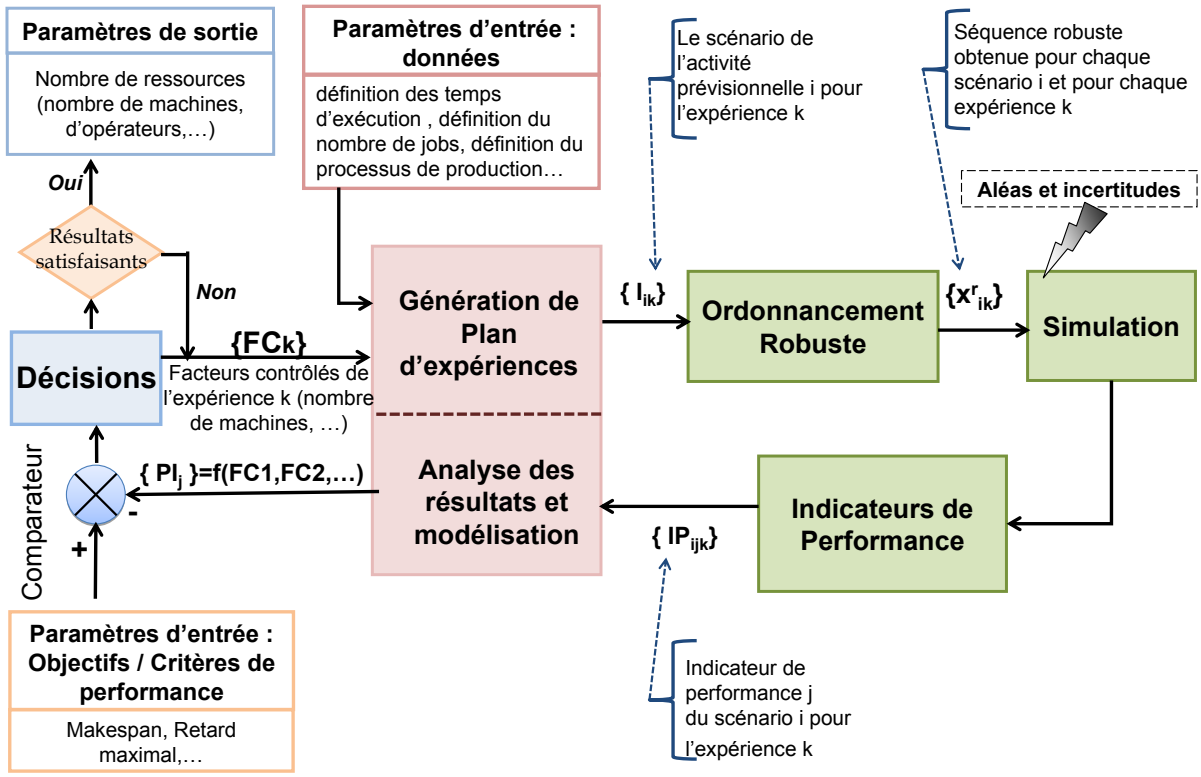

FIG. 4.5 – Architecture de la démarche méthodologique d'aide au dimensionnement des systèmes de production

D'autres auteurs ont travaillé sur d'autres types de démarche méthodologique intégrant des techniques d'optimisation couplées avec la simulation (Pierreval et Paris, 2003), (Paris *et al.*, 1996), (Pierreval et Tautou, 1997),(Pierreval et Paris, 2000), (Cheikhrouhou *et al.*, 2002) mais ils ne considèrent pas le problème d'ordonnancement.

Nous présentons par la suite la description des différents modules constituant l'architecture de la démarche méthodologique d'aide à la conception des systèmes de production :

1. **Paramètres d'entrée : Données**

   Dans ce module, l'utilisateur (décideur) doit définir les différents éléments du système. Il doit fournir les entrées et les paramètres du système (par exemple : le temps d'exécution, le nombre de *jobs*, le nombre d'étages,...).

2. **Décisions** :

   Ce module sera l'interface qui va permettre au décideur de choisir la meilleure conception qui répond à ses objectifs et critères de performance. Initialement, le décideur va fixer les différents facteurs (paramètres) qu'il va contrôler, notés FC_k (Facteurs contrôlés de l'expérience k) et qui constitueront les décisions à prendre (par exemple le nombre de machines, le nombre d'opérateurs, la taille des lots, les affectations des opérations, disposition des machines, etc.). Ces facteurs seront utilisés dans la construction de plans d'expériences. Une fois les expériences réalisées (ordonnancement, simulation) et les indicateurs de performance, ainsi que les fonctions les reliant avec les facteurs contrôlés calculés, le décideur comparera ces résultats avec les objectifs et les critères de performance qu'il s'est fixés. Si les résultats sont satisfaisants le décideur définit alors les paramètres de sortie (meilleure conception obtenue), sinon il doit réajuster les facteurs et reboucler avec les modules de la démarche. Dans ce module, nous pouvons mettre en place un système d'aide multicritère à la décision, pour assister le décideur dans le choix de la meilleure conception. Le développement de ce système est envisageable dans des travaux futurs.

3. **Génération de plans d'expériences** :

   L'objet du plan d'expériences est de définir les différents scénarios possibles de conception du système, sous forme d'essais ou expériences (dans chaque expérience, on fait varier les facteurs contrôlés (paramètres)). Ces scénarios, notés I_{ik} (I_{ik} est le scénario initial i de l'expérience k), sont définis par l'attribution d'une seule valeur à chaque facteur contrôlé.

4. **Ordonnancement robuste** :

   Ce module permet de choisir une méthode d'ordonnancement qui répond aux ob-

jectifs choisis par le décideur. Les méthodes de résolution pour un tel problème d'ordonnancement peuvent être des méthodes exactes, des heuristiques, des méta-heuristiques ou même des règles de priorité. Ces méthodes doivent fournir des ordonnancements robustes par rapport aux évènements qui pourraient survenir dans le futur. La sortie de ce module est une séquence robuste de *jobs* de chaque scénario pour chaque expérience. Cette séquence est notée x^r_{ik}.

5. **Simulation** :

   Un modèle de simulation de système de production sera construit pour simuler chaque expérience définie dans le module "Génération du plan d'expérience". Ce modèle de simulation intègre des données et des évènements incertains (temps d'exécution perturbé, demande incertaine, probabilité des pannes sur les machines, etc.). Après la simulation, nous obtiendrons un ensemble d'indicateurs de performances qui doivent être caractérisés, pour évaluer la meilleure conception parmi les différentes expériences.

6. **Indicateurs de performance** :

   Pour chaque expérience simulée, nous calculons un ensemble d'indicateurs de performance (IP_{ijk} : Indicateur de performance j du scénario i pour l'expérience k) comme par exemple la cadence de production, la durée de séjour des produits, le taux d'occupation des machines, etc. Ces indicateurs dépendent des caractéristiques du système de production et des produits fabriqués. Il n'existe pas de normes standards pour le choix d'un indicateur, tout dirigeant est amené à définir des indicateurs de performance propres à son système. Ce choix devrait être fait de manière à assurer un système cohérent de gestion de la performance.

7. **Analyse des résultats et modélisation** :

   Ce module nous permettra d'interpréter et analyser les résultats. Deux types d'activités peuvent être envisagés. Le premier est un modèle définissant l'impact de tous les facteurs contrôlés (FC_k) sur la valeur de chaque indicateur de performance IP_j. Le second concerne une analyse multicritère des résultats, intégrant des analyses de Pareto optimalité permettant au décideur de trouver la solution Pareto optimale.

8. **Paramètres d'entrée : objectifs et critères de performance** :
   Les objectifs et les indicateurs de performance cibles sont définis dans ce module (par exemple, cadence de production désirée, le taux d'occupation ciblé, etc.). Ces objectifs sont exprimés sous différentes formes : valeurs cibles des indicateurs de performances IP_j^*, contraintes propres au domaine, relation entre les indicateurs de performance, etc.

9. **Paramètres de sortie : objectifs et critères de performance** :
   Basé sur les résultats obtenus dans le module "Analyse des résultats et modélisation" comparés aux objectifs et aux critères de performances fixés, le décideur définit la "meilleure" conception de son système et constituera le résultat de ce module.

Cette démarche méthodologique est caractérisée par sa généricité et son applicabilité. Cette généricité est due au fait que les modules ne sont pas cloisonnés dans des outils ou concepts prédéfinis. C'est une méthodologie aidant à la conception des systèmes de production, en anticipant des questions et des problèmes qui seront posés lors de la phase d'exploitation. Elle peut être adaptée à différents systèmes de production de biens ou de services.

Nous nous intéressons par la suite à une des applications possibles de cette démarche dans un cas applicatif réel, permettant ainsi de valider les différents modules et de montrer l'apport de cette approche et son intérêt. Notre application s'intéresse à un problème particulier de la conception, qui est le dimensionnement et plus précisément le dimensionnement des blocs opératoires (Chaari *et al.*, 2008b).

4.5.2 Application : démarche méthodologique et outil d'aide au dimensionnement des blocs opératoires

Ces dernières années, les hôpitaux ont connu de fortes mutations dues aux réductions des enveloppes budgétaires, au vieillissement de la population, etc. Au vu de cet environnement, des projets de regroupement d'hôpitaux ou de sites, de restructuration et de réorganisation ont vu le jour. Ces projets nécessitent des prises de décisions sur le plan

stratégique, qui vont conditionner les décisions aux niveaux tactique et opérationnel.

Le bloc opératoire est une des structures qui a vu des changements dans son organisation et son activité. De blocs opératoires éclatés et dédiés à des services, nous sommes passés à des blocs opératoires regroupés et mutualisés pour plusieurs services différents. Dans d'autres projets nous avons assisté à des regroupements des établissements appartenant à des structures juridiques différentes, comme ce qui se passe actuellement à Valenciennes, avec le nouvel hôpital regroupant le Centre Hospitalier de Valenciennes (établissement public) et la clinique Teissier (établissement privé à but lucratif). Une des problématiques à soulever est de concevoir ce nouveau bloc opératoire, ou, de façon plus large, le plateau médico-technique.

Dans cette section, nous nous intéressons donc au problème de conception des blocs opératoires et plus particulièrement au problème de dimensionnement des blocs opératoires. Dans un premier temps, nous présentons la similitude entre le problème d'ordonnancement du bloc opératoire et le problème d'ordonnancement d'un *flow shop* hybride à deux étages. Dans un deuxième temps, nous exposons l'application de la démarche méthodologique et la réalisation de l'outil d'aide au dimensionnement des blocs opératoires.

4.5.2.1 Problématiques d'ordonnancement et de dimensionnement des blocs opératoires

Le problème d'ordonnancement des interventions au sein d'un bloc opératoire (BO) peut être assimilé à un problème de type ordonnancement d'un *flow shop* hybride à deux étages, avec des durées opératoires différentes entre les étages. Le premier étage est composé des salles d'opération (SO) identiques et le deuxième étage est constitué de lits identiques, dans la Salle de Surveillance Post-Interventionnelle (SSPI).

Des travaux de recherche se sont intéressés aux flux de patients dans les blocs opératoires. Les principaux flux de patients qui ont subi des interventions chirurgicales sont présentés dans la figure 4.6.

Le patient peut avoir différents cheminements possibles avant et après son intervention chirurgicale. Dans cette thèse, nous nous limitons au flux à l'intérieur du bloc opératoire.

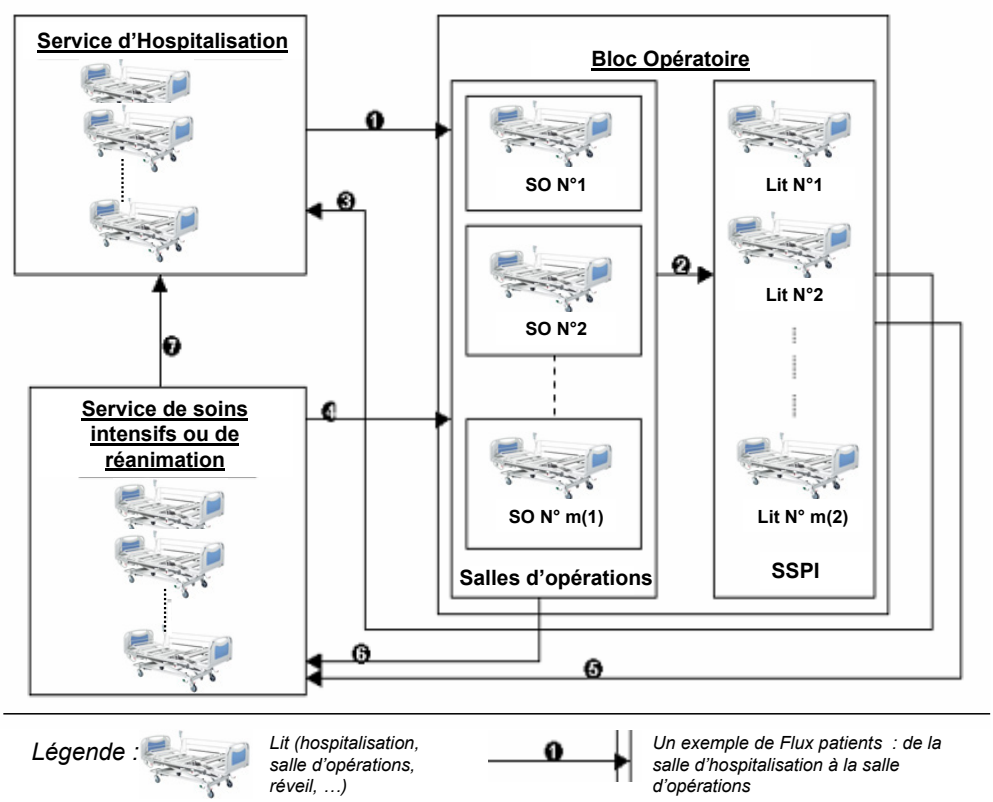

FIG. 4.6 – Provenance et destination des patients avant et après l'intervention chirurgicale

Un patient est transféré au bloc opératoire (1 ou 4) par des brancardiers. Il subit son intervention chirurgicale dans une salle d'opérations (une des salles d'opérations disponibles, quand celles-ci sont polyvalentes, ou une salle d'opérations spécifique quand il s'agit de salles d'opérations dédiées). Une fois l'intervention chirurgicale terminée, le patient est transféré vers un des lits de la SSPI (2). Après son réveil, il est transféré par des brancardiers vers sa chambre dans le service d'hospitalisation (3) ou dans le service des soins intensifs ou de réanimation (5). Ce cheminement est le cheminement le plus classique et c'est celui-ci qui nous intéresse.

Le bloc opératoire est donc composé principalement de M_1 salles d'opérations, qui peuvent être polyvalentes ou dédiées et de M_2 lits dans la SSPI. Il peut être alors assimilé au problème du *flow shop* hybride à deux étages (Chaabane, 2004; Kharraja, 2003).

La problématique de dimensionnement consiste alors, dans notre cas, de déterminer le nombre de salles d'opérations et de lits dans la SSPI, basée sur des plannings opératoires

des scénarios de l'activité chirurgicale prévisionnelle de la nouvelle structure.

Des travaux ont été réalisés pour répondre à ce type de problématique. Marcon et Kharraja (2002) ont étudié le dimensionnement des lits dans la SSPI et l'impact de ce dimensionnement sur les performances du bloc opératoire.

4.5.2.2 Démarche méthodologique d'aide au dimensionnement des blocs opératoires

Nous présentons dans la figure 4.7 l'architecture globale de notre démarche méthodologique appliquée au problème de dimensionnement des blocs opératoires.

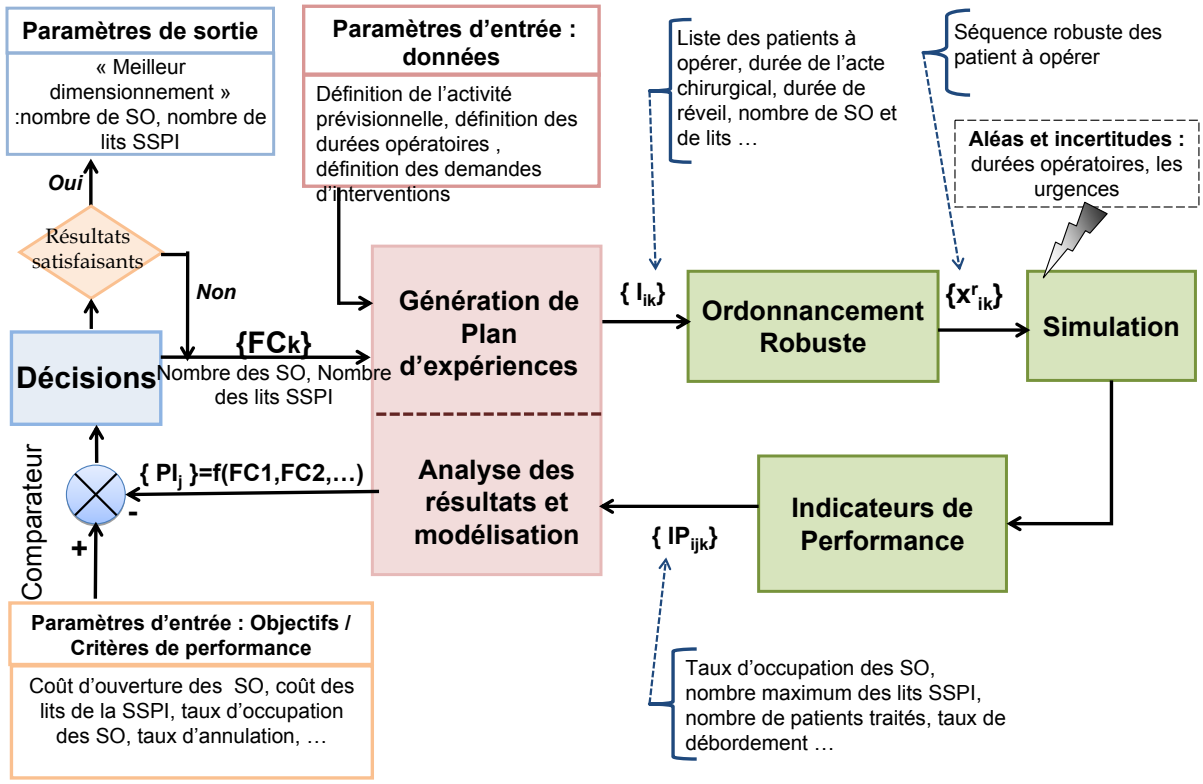

FIG. 4.7 – Architecture globale de la démarche méthodologique d'aide au dimensionnement des blocs opératoires

La démarche méthodologique développée dans la section 4.5.1 est ainsi appliquée au problème de dimensionnement des blocs opératoires.

Nous présentons par la suite l'implémentation des différents modules de l'architecture et les outils utilisés afin d'illustrer son application réelle, dans le cadre de la réalisation d'un outil global d'aide au dimensionnement des blocs opératoires. Nous avons réalisé une application Excel (voir figure 4.8) pour l'interface utilisateur. Cette interface permet de saisir les modules suivants : "Paramètres d'entrées : Données", "Décisions", "Génération de plan d'expériences", "Paramètres d'entrée : Objectifs et critères de performance cibles" dans les différentes feuilles de cette application.

Nous avons programmé en VBA les connexions entre cette interface et les modules : "Ordonnancement Robuste" et "Simulation".

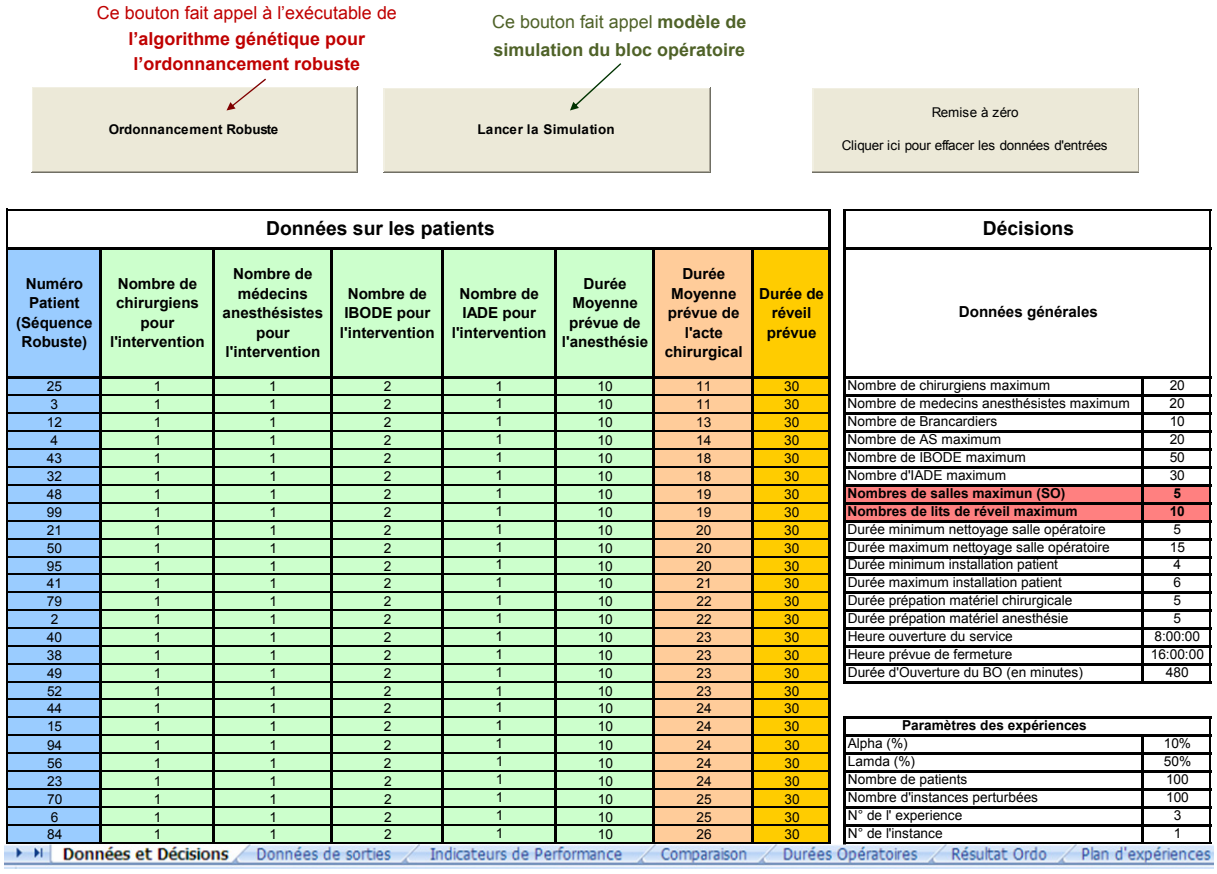

FIG. 4.8 – Interface de saisie des données et des décisions

Nous sommes partis d'un exemple inspiré d'un centre hospitalier. Nous avons construit une instance de 100 patients. Les durées opératoires ont été générées aléatoirement dans l'intervalle [10, 120]. En l'absence de référentiels normalisés au niveau des temps d'induction, de nettoyage et de préparation, nous sommes partis sur la base des travaux de Kharraja *et al.* (2002), où des temps ont été estimés par un expert médical (voir tableau 4.2).

Durées opératoire	Durée de réveil	Durée d'induction	Durée de nettoyage	Durée de préparation
$<$ 60 minutes	30	10	15	10
[60, 90]	60	20	15	10
[90, 120]	90	20	15	20
$\geq$ 120 minutes	120	30	30	20

TAB. 4.2 – Les temps nécessaires du bloc opératoire

Les décisions prises concernent le nombre de salles d'opérations, le nombre de lits dans la salle de réveil et les valeurs de λ et α.

Nos expériences sont définies comme suit :

- les valeurs de λ sont $0; 0,3; 0,5; 0,7; 1$
- les valeurs de α sont 10 % ; 25 % ; 50 %
- le nombre de Salles d'Opérations (SO) est égal à [5 ; 10 ; 15]
- le nombre de lits dans la SSPI sont multiples du nombre de SO [5 ; 8 ; 10], [10 ; 15 ; 20], [15 ; 23 ; 30] (donc à chaque niveau de SO correspondent 3 niveaux de lits dans la SSPI).

La méthode utilisée dans le module " Ordonnancement Robuste " est **l'algorithme génétique pour l'ordonnancement robuste présenté dans la section 2.6 et appliqué à l'ordonnancement des patients**. Nous avons utilisé le programme en Microsoft Visual C++ 6.0 réalisé pour les expérimentations du chapitre 3 et nous avons obtenu un exécutable appelé avec un code VBA en cliquant directement sur le bouton "Ordonnancement Robuste" (voir figure 4.8).

Le décideur doit fixer le degré d'incertitude α sur les durées opératoires. Pour trouver un ensemble de solutions pour ce problème, il peut faire varier la valeur λ entre 0 et 1 et appliquer l'algorithme génétique pour l'ordonnancement robuste pour chaque valeur de λ. Plus la valeur de λ est importante, plus la priorité est donnée à la minimisation de la date de fermeture du bloc opératoire $Cmax_{I_{ik}}$ par rapport à celui associé à la déviation entre les scénarios perturbés et le scénario initial (ici le scénario initial est représenté par I_{ik}). Par exemple, pour trouver une solution robuste, ce qui est le cas de $\lambda = 0$, nous favorisons la minimisation de la déviation entre les scénarios perturbés et le scénario initial I_{ik}. Pour trouver une solution efficace, ce qui est le cas de $\lambda = 1$, nous favorisons la minimisation de la date de fermeture du bloc opératoire du scénario initial ($Cmax_{I_{ik}}$).

Ceci a permis d'obtenir la séquence robuste x^r_{ik} présentée dans la colonne "Numéro Patient" et qui constitue l'ordre d'arrivée des patients dans le bloc opératoire. La figure 4.8 présente alors le résultat de l'ordonnancement robuste obtenu pour l'expérience N°3, le nombre de salles d'opérations SO= 5, le nombre de salles de réveil dans la SSPI égal à 10, $\lambda = 0.5$ et $\alpha = 0.25$. Cette séquence, les données et les décisions sont utilisées par la suite dans le modèle de simulation.

La figure 4.9 présente le modèle de simulation du bloc opératoire composé d'un nombre maximum de 18 salles d'opérations et de 30 lits dans la SSPI. Ce modèle a été construit sous ARENA 12.0 (inspiré du futur bloc du CH de Valenciennes).

Nous avons intégré par la suite les résultats des indicateurs de performance dans cette même application, en récupérant tous les résultats obtenus par simulation. Ceci permettra à l'utilisateur d'exploiter les résultats, soit en les modélisant, soit en les comparant directement avec les objectifs et les indicateurs de performance cibles. L'utilisateur peut par la suite fournir ses paramètres de sortie, une fois la (ou les) décision(s) prise(s).

Nous avons défini cinq indicateurs de performance : nombre de patients traités, taux d'utilisation des salles d'opérations durant les heures régulières, taux de débordement (pourcentage de temps utilisé en heures supplémentaires), nombre maximum et nombre moyen de lits de réveil utilisés durant les heures d'ouverture régulières.

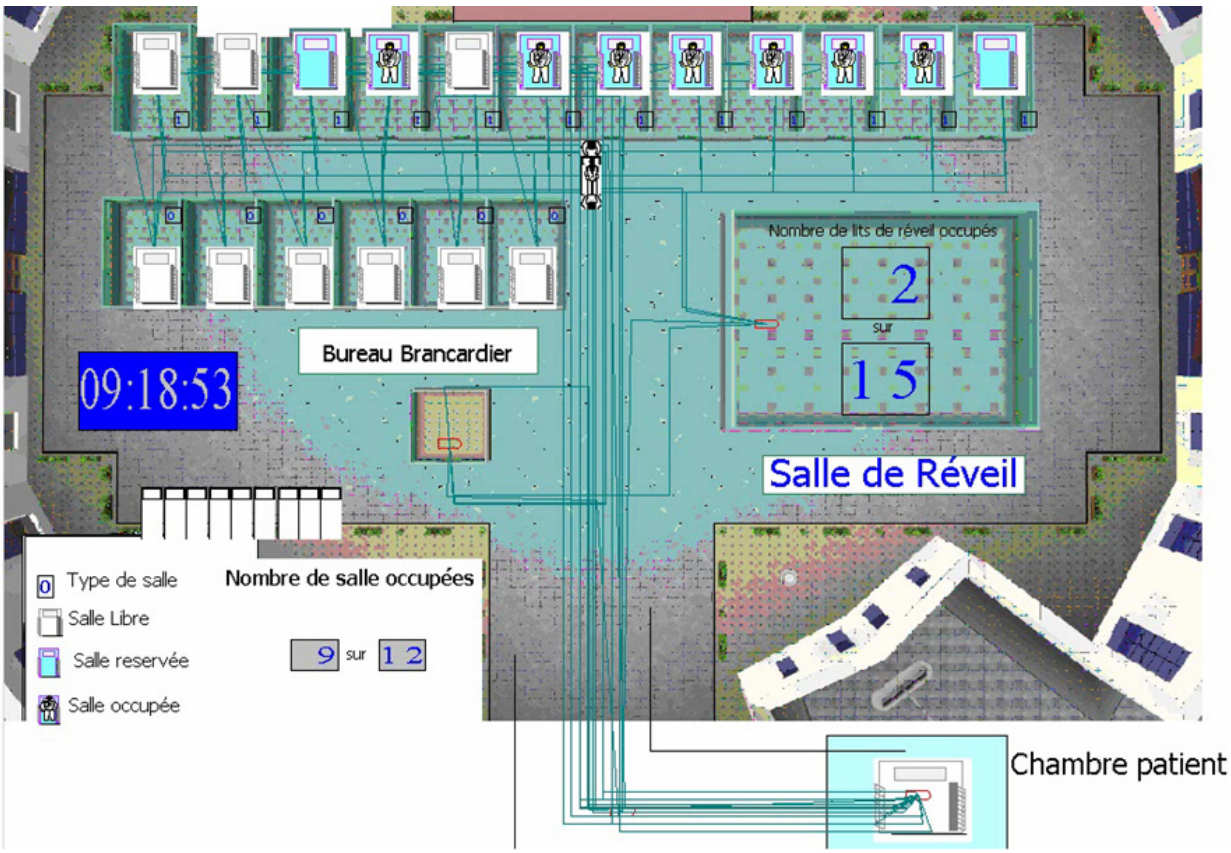

FIG. 4.9 – Modèle de simulation sous ARENA 12.0

Pour l'analyse et la modélisation des résultats, nous pouvons définir par exemple la fonction (linéaire) reliant le nombre de patients traités par rapport aux nombre de salles d'opérations (voir figure 4.10).

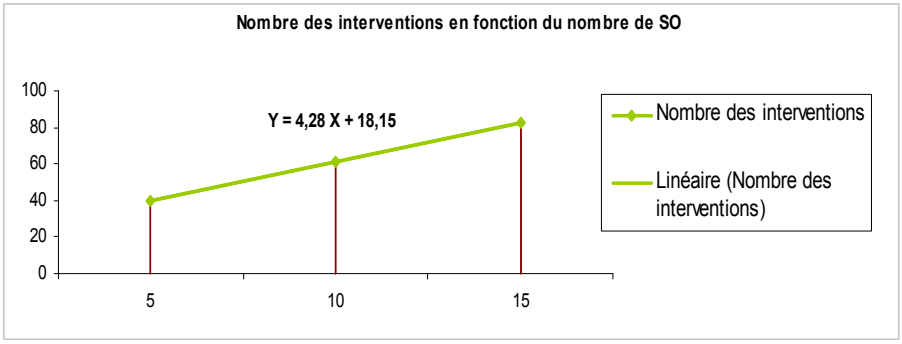

FIG. 4.10 – Estimation du nombre de patients par rapport aux nombre des SO

Une comparaison entre les heures réelles et les heures simulées des patients pour calculer le nombre des patients en retard à l'entrée et à la sortie du bloc opératoire (voir figure 4.11) peut être également réalisée.

	Patients en retard de plus de (min):		30
	A l'entrée du bloc	A la sortie de SO	A la sortie du bloc
Nombre	2	5	11
Taux	2%	5%	11%

FIG. 4.11 – Comparaison des résultats

On peut fournir, pour une valeur donnée de λ (ici $\lambda = 0,5$) et α (ici $\alpha = 0,25$) un tableau comme celui présenté dans la figure 4.12. Nous pouvons faire appel à des techniques d'analyse multicritère pour choisir la meilleure alternative.

	Facteurs		Indicateurs de Performances : Résultats mesurés			
Essais	Nombre de SO	Nombre de lits en SSPI	Nombre de patients traités	Taux d'utilisation des SO durant les heures d'ouverture régulières	Taux d'utilisation des SO durant les heures supplémentaires	Nombre maximum de lits de réveil utilisés durant les heures d'ouverture régulières
1	5	5	38,4	87,1%	2,8%	5
2	5	8	38,8	86,9%	2,7%	5
3	5	10	37,4	86,8%	2,8%	5
4	10	10	62,2	87,8%	3,2%	10
5	10	15	64,8	87,5%	2,4%	10
6	10	20	67,2	87,3%	2,2%	10
7	15	15	79,4	88,1%	0,9%	13,2
8	15	23	80,2	89,1%	0,3%	13,2
9	15	30	82,4	89,6%	0,5%	13,2

FIG. 4.12 – Résultats d'un plan d'expériences

En s'appuyant sur ces résultats, en les comparant avec les objectifs et les critères de performance cibles, ou en les analysant par des techniques multicritère ou en appliquant les modèles reliant les indicateurs de performance aux facteurs contrôlés, le décideur est amené à choisir le meilleur dimensionnement parmi les différentes expériences réalisées.

Nous remarquons ici la richesse des résultats que peut obtenir le décideur sur le comportement futur de son système en intégrant des informations de la phase d'exploitation. Ainsi, le dimensionnement tiendra compte des détails et sera au plus juste. En revanche, le décideur doit avoir des outils rigoureux pour l'analyse des résultats et leurs exploitations. Cette multitude d'informations peut être un problème en soit, car le décideur doit bien sélectionner les informations des plus importantes aux moins importantes, et donner des poids aux différentes décisions. C'est pour cette raison qu'il doit bien définir son plan d'expérience et quelle méthode appliquer pour l'exploitation des résultats. Cet outil est un outil d'aide au dimensionnement et en aucun cas un outil de dimensionnement, c'est le décideur qui doit prendre les décisions finales.

4.6 Conclusion

Nous avons proposé dans ce chapitre une démarche méthodologique intégrant des techniques d'optimisation et de simulation pour l'aide à la conception des systèmes de production basée sur des ordonnancements robustes, face à un ensemble de perturbations. Cette démarche est générique et peut être appliquée aux différents types d'organisation (*flow shop* hybride, *job shop* flexible, etc.). Nous avons développé les différents modules de la démarche sous forme d'un outil d'aide au dimensionnement dans le cadre d'un cas applicatif réel, celui du bloc opératoire dans le secteur hospitalier. Ceci constitue une des applications possibles de notre démarche. L'objectif est de minimiser le coût d'investissement des salles d'opérations et des lits dans la salle de réveil et de minimiser la date de fermeture du bloc opératoire. Cette démarche méthodologique permettra d'accompagner le décideur dans son projet de restructuration ou de conception d'un nouveau bloc opératoire. L'outil est ainsi réalisé et implémenté. Nous avons présenté les différents outils développés dans ce cadre pour illustrer notre démarche. L'algorithme génétique pour l'ordonnancement robuste a été intégré et utilisé dans un des modules de cette démarche.

Conclusion générale & perspectives

La plupart des méthodes d'ordonnancement sont déterministes, elle ne prennent pas en considération le caractère incertain des données du problème. De ce fait, un ordonnancement déterministe deviendra rapidement inefficace avec les incertitudes. Pour pallier ce problème, des méthodes d'ordonnancement, dites robustes, ont été développées, ayant la capacité de résister aux perturbations de l'environnement tout en assurant une bonne performance. Le travail présenté dans cette thèse s'inscrit dans ce domaine de recherche.

Une nouvelle classification des différentes approches de l'ordonnancement dans un milieu incertain a été proposée dans la première partie de ce travail. Trois approches ont été distinguées, à savoir les approches proactives, les approches réactives et les approches hybrides, qui sont elle mêmes composées de deux approches : les approches prédictives-réactives et les approches proactives-réactives. Pour chacune de ces approches, un état de l'art des méthodes de résolution, ainsi que des avantages et des inconvénients, a été présenté.

Certaines méthodes d'ordonnancement utilisent des techniques d'optimisation efficaces permettant de fournir des solutions de bonne qualité, dans un temps de calcul raisonnable. La recherche de cette solution revient à utiliser les métaheuristiques. Dans ce cadre, parmi les techniques les plus employées, nous trouvons les algorithmes génétiques, qui présentent des solutions de qualité intéressante pour la résolution des problèmes combinatoires complexes, leurs applications en ordonnancement sous incertitudes sont limitées. En effet, nous avons proposé un algorithme génétique permettant d'obtenir une solution robuste de bonne performance et qui est peu sensible aux incertitudes. Une organisation de type

flow shop hybride constitue notre cas d'étude. Pour représenter les incertitudes sur les caractéristiques du problème (temps d'exécution incertain), nous avons choisi la modélisation par scénarios. Le contenu de ces scénarios est le reflet d'hypothèses concernant l'incertain. Nous avons développé un critère de robustesse permettant de minimiser simultanément la date d'achèvement des *jobs* et la déviation entre les scénarios perturbés et le scénario initial. Ce dernier permettant de trouver une solution de bonne performance et peu sensible aux incertitudes. Nous avons développé aussi un nouveau mécanisme intégré dans l'algorithme génétique. Ce mécanisme consiste à évaluer, à chaque génération de l'algorithme génétique, tous les chromosomes sur un ensemble de scénarios perturbés.

La seconde étape de notre étude est consacrée, en premier lieu, à tester et vérifier l'algorithme génétique pour l'ordonnancement déterministe. Lors de cette phase, nous avons testé l'algorithme génétique pour l'ordonnancement déterministe sur des *benchmarks* existant dans littérature, dont l'objectif est de minimiser seulement la date d'achèvement des *jobs* (*makespan*). Cette série d'expérimentations est effectuée de manière statique, sans intégrer des perturbations. Les données utilisées sont déterministes et ne prennent pas en compte les incertitudes. En deuxième lieu, nous avons validé notre algorithme génétique pour l'ordonnancement robuste par la simulation. Un modèle de simulation a été implémenté sur ARENA 12.0. Les résultats de simulation montrent que la méthode proposée peut produire de meilleures solutions, dites de compromis, entre efficacité et robustesse pour un degré d'incertitude très élevé. Cette robustesse est traduite par la résistance aux perturbations intégrées dans le modèle de simulation et causées par les incertitudes des temps d'exécution des *jobs*.

La dernière phase de notre étude consiste à appliquer la robustesse dans le domaine de la conception et plus particulièrement le problème du dimensionnement. L'objectif est de travailler conjointement le dimensionnement et l'ordonnancement, dès la phase de conception. L'application en ordonnancement robuste constitue la seconde phase, puisque la qualité du dimensionnement sera basée sur la qualité des ordonnancements proposés et leur robustesse face aux perturbations. En effet, nous avons développé une approche statique (déterministe) qui consiste à dimensionner les machines dans un atelier de production, en tenant compte de la performance et du type des ordonnancements prévisionnels. Nous

avons ensuite développé une démarche méthodologique intégrant des techniques d'optimisation et de simulation pour l'aide au dimensionnement des systèmes de production basé sur des ordonnancements robustes. Les différents modules de la démarche sont développés sous forme d'un outil d'aide au dimensionnement, dans le cadre d'un cas applicatif réel, celui du bloc opératoire dans le secteur hospitalier. Cette démarche méthodologique permettra d'accompagner le décideur dans son projet de restructuration ou de réalisation d'un nouveau bloc opératoire.

Par ailleurs, ces travaux ont permis d'ouvrir de nouvelles perspectives pour les études futures sur le plan théorique (recherche), méthodologique ou pratique (application).

La première perspective scientifique concerne le type d'incertitude. Notre état de l'art montre que la plupart des travaux de recherche s'intéressent seulement à un type d'incertitude : l'incertitude au niveau opérationnel. Cependant, en réalité, d'autres types d'incertitudes existent aux plus hauts niveaux stratégique et tactique. Les incertitudes stratégiques sont celles qui affectent principalement la prise de décision à long terme. Ces incertitudes venant des conditions environnementales, des changements de technologie et des règlements gouvernementaux. Les incertitudes tactiques peuvent changer la prise de décision à moyen terme, telles que les paramètres du marché, les perturbations dans l'information et les flux des matières. Il est donc nécessaire de traiter la combinaison de ces types d'incertitudes.

La deuxième perspective théorique concerne la recherche d'une solution robuste. Il est intéressant d'implémenter un nouveau mécanisme intégré dans l'algorithme génétique pour caractériser les scénarios perturbés. Ce mécanisme permet d'évaluer chaque chromosome sur un ensemble de scénarios perturbés (c'est-à-dire que chaque chromosome est évalué sur son propre ensemble de scénarios perturbés). Cet ensemble de scénarios change pour chaque évaluation d'un seul chromosome. Une autre piste possible, est d'utiliser une méthode d'optimisation multiobjectif permettant d'optimiser les deux objectifs. Cette méthode est représentée par un ensemble de points. Cet ensemble de points est constitué par des solutions non dominantes, encore appelées optimales dans le sens de Pareto. Un autre aspect, qui peut être abordé, est la recherche d'une solution de compromis entre efficacité

et robustesse. Cette solution peut être obtenue en deux étapes. La première étape consiste à trouver par l'algorithme génétique pour l'ordonnancement déterministe un ensemble de solutions initiales (population) qui est efficace (minimisation du *makespan*). Cette population est évaluée sur un ensemble de scénarios perturbés par une fonction d'évaluation mono-objectif (par exemple la déviation entre le scénario initial et les scénarios perturbés).

La troisième perspective se situe au niveau méthodologique et concerne le développement et l'intégration d'un système d'aide multicritère à la décision pour aider le décideur à choisir la meilleure configuration qui répond mieux à ses objectifs. Parmi ces méthodes, nous trouvons les méthodes de surclassement, les méthodes interactives, etc.

Au niveau pratique, l'application de notre méthodologie au niveau des blocs opératoires peut s'étendre, par l'intégration de nouvelles connaissances sur l'arrivée des patients en urgence, qui reste un problème plus compliqué. Une approche proactive-réactive permettra de résoudre ce type de problème. D'autres applications de notre démarche méthodologique sont envisageables, dans le cas des entreprises manufacturières. D'autres applications de l'ordonnancement robuste peuvent être réalisées comme par exemple les algorithmes génétiques embarqués dans les produits intelligents pour calculer des ordonnancements robustes.

Bibliographie

Aissani, N., B. Beldjilali et D. Trentesaux, 2008. Use of machine learning for continuous improvement of the real time heterarchical manufacturing control system performances. *International Journal of Industrial and Systems Engineering*, 3(4) :474–497. 24

Aissani, N., B. Beldjilali et D. Trentesaux, 2009. Dynamic scheduling of maintenance tasks in the petroleum industry : a reinforcement approach. *Engineering Applications of Artificial Intelligence*, 22(7) :1089–1103. 34

Akturk, M. et M. Gorgulu, 1999. Match-up scheduling under a machine breakdown. *European Journal of Operational Research*, 112 :81–97. 31

Albadawi, Z., B. Boulet, R. DiRaddo, P. Girard, A. Rail et V. Thomson, 2006. Agent-based control of manufacturing processes. *International Journal of Manufacturing Research*, 1(4) :466–481. 24

Allaoui, H. et A. Artiba, 2006. Scheduling two-stage hybrid flow shop with availability constraints. *Computer and Operations Research*, 33(5) :1399–1419. 45

Aloulou, M. et M. Portmann, 2002. A genetic algorithm to achieve scheduling flexibility for a single machine problem. *RAIRO-Operations Research*, page 19. 32

Artigues, C., C. Briand, M. Portmann et F. Roubellat, 2002. Pilotage des systèmes de production. *chapitre Pilotage d'atelier basé sur un ordonnancement flexible, Edition Hèrmes*. 10

Artigues, C. et F. Roubellat, 2000. A polynomial activity insertion algorithm in a multi-

resource schedule with cumulative constraints and multiple modes. *European Journal of Operational Research*, 127 :297–316. 28

Askin, R. et C. Standridge, 1993. Modeling and analysis of manufacturing systems. *John Wiley and Sons, New York*. 85

Aubry, A., M. Espinouse et M. Jacomino, 2006. Robust load - balanced configuration with fixed costs for the parallel multi-purpose machines problem. *International Conference Service Systems and Service Management*, pages 990–995. 86

Aubry, A., M. Espinouse, M. Jacomino et P. Mrozicki, 2005. Compromis entre robustesse et coût pour la configuration d'un parc de machines parallèles partiellement multifonctions. *MajecSTIC 2005, Manifestation des Jeunes Chercheurs francophones dans les domaines des STIC, Rennes*, pages 331–338. 86

Baker, A., 1998. A survey of factory control algorithms that can be implemented in a multi-agent heterarchy dispatching scheduling and pull. *Journal of Manufacturing Systems*, 17(4) :297–320. 24

Baybars, I., 1986. A survey of exact algorithms for the simple assembly line balancing. *Management science*, 32 :909–932. 87

Bean, J. C., J. R. Birge, J. Mittenthal et C. E. Noon, 1991. Match-up scheduling with multiple resources, release dates and disruptions. *Operations Research*, 39(3) :470–483. 31

Bierwirth, C. et D. Mattfeld, 1999. Production scheduling and rescheduling with genetic algorithms. *Evolutionary computation*, 7(1) :1–17. 28

Billaut, J., A. Moukrim et E. Sanlaville, 2005. Flexibilité et robustesse en ordonnancement. *Hèrmes Sciences Publications*. 16

Bonfill, A., 2006. Proactive management of uncertainty to improve scheduling robustness in process industries. *Ph.D. thesis, Universitat Politecnica de Catalunya*. 10

Bouchon-Meunier, B., 1995. La logique floue et ses applications. *Addison-Wesley*, France. 9

Boucon, D., 1991. Ordonnancement d'atelier, aide au choix de règles de priorité. *thèse de doctorat, ENSAE, Toulouse, France*. 26

Bouyssou, D., 1989. Modelling inaccurate determination, imprecision using multiple criteria. *in Improving Decision Making in Organisations, Lockett, A.G. et Islei, G. (Editors), Springer Verlag, Heidelberg*, pages 78–87. 11

Brah, S. et J. Hunsucker, 1991. Branch and bound algorithm for the flow shop with multiple processors. *European Journal of Operational Research*, 51 :88–99. 45

Brautigam, J., C. Esche et A. Mehler-Bicher, 2003. Uncertainty as a key value driver of real options. *Proceedings of the annual international conference on real options, http ://www.realoptions.org/papers2003/BraeutigamUncertainty.pdf*. 10

Carlier, J. et E. Néron, 2000. An exact method for solving the multiprocessor flowshop. *RAIRO - Operations Reseach*, 34(1) :1–25. 59, 60, 61

Caux, C., H. Pierreval et M. Portmann, 1995. Les algorithmes génétiques et leurs applications aux problèmes d'ordonnancement. *Revue d'Automatique de Productique et d'Informatique Industrielle*, 29(4-5) :409–443. 39

Chaabane, S., 2004. Gestion prédictive des blocs opératoires. *Thèse, Informatique et Systèmes Coopératifs pour l'Entreprise, Institut National des Sciences Appliquées de Lyon*. 107

Chaari, T., S. Chaabane, N. Aissani et D. Trentesaux, 2009. Scheduling under uncertainty : a literature survey. *Submitted to Computers and Industrial Engineering*. 12, 14

Chaari, T., S. Chaabane, T. Loukil et D. Trentesaux, 2008a. Optimisation du dimensionnement d'un atelier de production basé sur les ordonnancements des scénarios de production prévisionnels, Cas d'un Flow shop hybride. *7ème conférence internationale francophone (MOSIM), Paris-France*, pages 790-798 :ISBN 978–2–7430–1057–7. 91

Chaari, T., S. Chaabane, D. Trentesaux et C. Tahon, 2008b. Démarche méthodologique et outil d'aide au dimensionnement des blocs opératoires. *Gestion et Ingénierie des SystèmEs Hospitaliers (GISEH), Lausanne-Suisse*, pages 833-840 :ISBN 978–2–8399–0316–5. 105

Cheikhrouhou, N., J. Paris et H. Pierreval, 2002. Une méthode de dimensionnement de lignes de transfert via l'analyse de perturbation finie. *Journal Européen des Systémes Automatisés*, 36(2) :199–221. 91, 102

Chen, C. et Y. Yih, 1996. Identifying attributes for knowledge-based development in dynamic scheduling environments. *International Journal of Production Research*, 34(6) :1739–1755. 27

Chiang, W.-Y. et M. Fox, 1990. Protection againts uncertainty in a deterministic schedule. *Fourth International Conference on Expert Systems in Production and Operations Management, South California, USA*. 17

Ching-Torng, L., H. Chiu et P. Chu, 2006. Agility index in the supply chain. *International Journal of Production Economics*, 100(2) :285–299. 82

Choon, H., 1991. Combining simulation with optimization search techniques for the design of flexible manufacturing systems. *In Proceedings of the International Conference on Computer Integrated Manufacturing, Singapore*, pages 159–162. 89

Chu, C. et M. C. Portmann, 1993. Application of the artificial memory approach to scheduling problems. *Journal of Intelligent Manufacturing*, 4(2) :151–161. 26

Church, L. et R. Uzsoy, 1992. Analysis of periodic and event driven rescheduling policies in dynamic shops. *International Journal of Computer Integrated Manufacturing*, 5 :153–163. 28

Czarneck, H., B. Schroer et M. Rahman, 1997. Using simulation to schedule manufacturing resources. *In Proceedings of the Winter Simulation Conference, Atlanta*, pages 750–757. 88

Daniels, R. et J. Carrillo, 1997. Beta-robust scheduling for single machine systems with uncertain processing times. *IIE Transactions*, 29 :977–985. 19

Davenport, A. et J. Beck, 2000. A survey of techniques for scheduling with uncertainty. *Working paper, http ://www.eil.utoronto.ca/chris/chris.papers.html.* 13, 16

Davenport, A., C. Gefflot et J. Beck, 2001. Slack-based techniques for robust schedules. *Sixth European Conference on Planning (ECP-2001), Toledo, Spain.* 18, 19

De Matta, R., V. Hsu et T. Lowe, 1999. Capacitated selection problem. *Naval Research Logistics*, 46(1) :19–37. 86

Dexter, F., A. Macario et R. Traub, 1999. Which algorithm for scheduling add-on elective cases to maximizes operating room utilization ? use of bin packing algorithms and fuzzy constraints in operating room management. *Anesthesiology*, 91 :1491–1500. 12

Dolgui, A., B. Finel, N. Guschinsky, G. Levin et F. Vernadat, 2006. Mip approach to balancing transfer lines with blocks of parallel operations. *IIE transactions*, 38 :869–882. 87

Dolgui, A., N. Guschinsky et G. Levin, 1999. On problem of optimal design of tranfer lines with parallel and sequential operation. *in Proceedings of the 7th IEEE International Conference on Emerging Technologies and Factor Automation, edited by J.M. Fuertes, Barcelona, Spain*, 1 :329–334. 87

Dréo, J., A. Péreowski, P. Siarry et E. Taillard, 2003. Métaheuristiques pour l'optimisation difficile. *Editions Eyrolles*. 40

Drummond, M., J. Bresina et K. Swanson, 1994. Just - in - case scheduling. *Twelfth National Conference on Artificial Intelligence (AAAI-1994) Seattle USA*. 20

Dumbrava, S., 1997. The design of flexible manufacturing systems using simulations. *Intelligent Manufacturing Systems*, pages 151–155. 88

Dunne, T. et X. Mu, 2008. Investment spikes and uncertainty in the petroleum refining industry. *Working Paper, Paper provided by Federal Reserve Bank of Cleveland in its series*, (Number 0805). 9

Einhorn, H. et R. Hogarth, 1985. Ambiguity and uncertainty in probabilistic inference. *Psychological Review*, 92(4) :433–461. 11

Elkhyari, A., 2003. Outil d'aide à la décision pour des problèmes d'ordonnancement dynamique. *Thèse de doctorat, Université de Nantes.* 31

Engin, O. et A. Doyen, 2004. A new approach to solve hybrid flow shop scheduling problems by artificial immune system. *Future Generation Computer Systems*, 20 :1083–1095. 61, 66

Esquirol, P. et P. Lopez, 1999. L'ordonnancement. *Economica*. 7

Esswein, C., 2003. Un apport de flexibilité séquentielle pour l'ordonnancement robuste. *Thèse de doctorat, Université François Rabelais,Tours, France*. 7

Feyzioglu, O., H. Pierreval et D. Delflandre, 2005. A simulation based optimization approach to size manufacturing systems. *International Journal of Production Research*, 43(2) :247–266. 84, 89

French, S., 1995. Uncertainty and imprecision modelling and analysis. *Journal of the Operational Research Society*, 46 :70–79. 11

Gao, H., M. Fox, W.-Y. Chiang et S. Hikita, 1995. Building robust schedules - An empirical study of single machine scheduling with uncertainty. *Working Paper, http ://www.eil.utoronto.ca/public/uncertainty-paper.ps*. 17

Gerchak, Y., D. Gupta et M. Henig, 1996. Reservation planning for elective surgery under uncertain demand for emergency surgery. *Management Science*, 42(3) :321–334. 12

Ghosh, S., 1996. Guaranteeing fault tolerance through scheduling in real time systems. *Thèse de doctorat à l'université de Pittsburgh*. 17

Gilmore, P. C. et R. E. Gomory, 1964. Sequencing a one state-variable machine : A solvable case of the traveling salesman problem. *Operations Research*, 12 :655–679. 46

Goldberg, D., 1989. Genetic algorithm in search, optimisation and machine learning. *Addison Wesley*. 36

GOThA, 2002. Flexibilité et robustesse en ordonnancement. *Le bulletin de la ROADEF*, 8 :10–12. 7

Gourgand, M., N. Grangeon et S. Norre, 1999. Metaheuristics for the deterministic hybrid flow shop problem. *Proceeding of the International Conference on Industrial Engineering and Production Management, Glasgow, United Kingdom*, pages 136–145. 46

Govil, M. et M. Fu, 1999. Queueing theory in manufacturing : a survey. *Journal of Manufacturing Systems*, pages 214–240. 85

Guchinskaya, O. et A. Dolgui, 2009. Configuration des lignes d'usinage à boîtiers multi-broches : une approche mixte. *RAI-RO Operations Research*, 43 :277–296. 87

Guillaume, C., D. Rémy et G. Gilles, 2001. une approche basée sur la simulation pour résoudre le problème du job shop périodique à contraintes linéaires. *Proceedings d'une Conférence Internationale de Modélisation et de Simulation, Troyes, France*, pages 199–203. 39

Gupta, J., 1988. Two-stage hybrid flowshop scheduling problem. *Journal of Operational Research Society*, 39(4) :359–364. 45, 46

Gupta, J., K. Krüger, V. Lauff, Y. Sotskov et F. Werner, 2002. Heuristics for hybrid flow shops with controllable processing times and assignable due dates. *Computers and Operations Research*, 29(10) :1417–1439. 46

Gupta, S. et N. Zhang, 2006. Multiperiod planning of refinery operations under market uncertainty. *The 2006 Annual Meeting San Francisco, CA, Systems and Process Design (10a), Computing and Systems Technology Division*. 11

Hammersley, J. et D. Handscomb, 1964. Monte carlo methods. *Methuen, London*. 12

Herroelen, W. S. et R. Leus, 2005. Project scheduling under uncertainty : Survey and research potentials. *European Journal of Operational Research*, 165(2) :289–306. 13

Huber, P., 1981. Robust statistics. *John Wiley and Sons*. 16

Jensen, M., 2000. Neighbourhood based robustness applied to tardiness and total flowtime jobshops. *Proceeding of sixth Parallel Problem Solving from Nature, Springer*, 1917 :283–292. 16

Jensen, M. T., 2001. Improving robustness and flexibility of tardiness and total flow-time job shops using robustness measures. *Applied Soft Computing Journal*, 1(1) :35–52. 22

Jeong, K. et Y. Kim, 1998. A real-time scheduling mechanism for a flexible manufacturing system : Using simulation and dispatching rules. *International Journal of Production Research*, 36(9) :2609–2626. 26

Jichao, X., 1996. Variability detection and robustness design in complex production system. *Computers Industrial Engineering*, 31(3-4) :775–778. 16

Jin, Z., Z. Yang et T. Ito, 2006. Metaheuristic algorithms for the multistage hybrid flowshop scheduling problem. *International Journal of Production Economics*, 100(2) :322–334. 46

Johnson, S. M., 1954. Optimal two and three-stage production schedule with setup times included. *Naval Research Logistics Quarterly*, 1(1) :61–68. 7

Kenne, J. et A. Gharbib, 2004. Stochastic optimal production control problem with corrective maintenance. *Computers and Industrial Engineering*, 46 :865–875. 8

Kharraja, S., 2003. Outils d'aide à la planification et l'ordonnancement des plateaux-médico-techniques. *Thèse :Génie Industriel, Université Jean-Monnet, Saint-Etienne.* 107

Kharraja, S., S. Chaabane et E. Marcon, 2002. Evaluation de performances pour deux stratégies de programmation opératoire de bloc. *IEEE Conférence Internationale Francophone d'Automatique (CIFA 2002).* 110

Kim, M. et Y. Kim, 1994. Simulation based real-time scheduling in a flexible manufacturing system. *Journal of Manufacturing Management Systems*, 13(2) :85–93. 25

Klaï, R. et C. Lamboray, 2007. L'alpha-robustesse lexicographique : Une relaxation de la Beta-robustesse. *Annales du LAMSADES, Numéro 7, Paris, France.* 23

Korbaa, O., 2003. Contribution à la conception et l'optimisation des systèmes de transport et de production. *Habilitation à diriger des recherches, Université des sciences et technologies de lille.* 82

Kouvelis, P., A. Kurawarwala et G. Gutiérrez, 1992. Algorithms for robust single and multiple period layout planning for manufacturing systems. *European journal of operational research*, 63 :287–303. 22

Kouvelis, P. et G. Yu, 1997. Robust discrete optimization and its applications : Non convex optimization and its applications. *Kluwer Academic Publishers.* 21

Kutanoglu, E. et I. Sabuncuoglu, 1999. An analysis of heuristics in a dynamic job shop with weighted tardiness objectives. *International Journal of Production Research*, 37(1) :165–187. 25

LA, H. T., 2005. Utilisation d'ordres partiels pour la caractérisation de solutions robustes en ordonnancement. *thése de doctorat à l'institut national des sciences appliquées de Toulouse-France*. 14, 23

Law, A. et W. Kelton, 2000. Simulation modeling and analysis. *MacGraw-Hill : New York*. 88

Le Pape, C., 1995. Artificial intelligence in reactive scheduling. *Chapman et Hall*. 16

Leon, V., S. Wu et R. Storer, 1994. Robustness measures and robust scheduling for job shops. *IIE Transactions*, 26(5) :32–43. 16

Lin, Z. et C. Yang, 1996. Evaluation of machine selection by the AHP method. *Journal of Materials Processing Technology*, 57(3-4) :253–258. 85

Linn, R. et W. Zhang, 1999. Hybrid flow shop scheduling : A survey. *Computers and Industrial Engineering*, 37(1-2) :57–61. 46

Lopez, P. et F. Roubellat, 2008. Production scheduling. *ISTE/Wiley, London*. 25

MacCathy, B. et J. Liu, 1993. Addressing the gap in scheduling research : a review of optimization and heuristic methods in production scheduling. *International Journal of Production Research*, 31(1) :59–79. 7

Marcon, E. et S. Kharraja, 2002. Etude exploratoire sur la stratégie de dimensionnement d'une SSPI. *IEEE Conférence Internationale Francophone d'Automatique (CIFA 2002)*. 108

Marik, V. et J. Lazansky, 2007. Industrial applications of agent technologies. *Control Engineering Practice*, 15 :1364–1380. 24

Masuchun, R. et W. Ferrell, 2004. Dynamic rescheduling with stability. *The Proceedings of the 5th Asian Control Conference*, pages 1886–1894. 30

Masuchun, R. et W. Masuchun, 2004. New rescheduling methodology with frozen interval. *2nd International Conference on Autonomous Robots and Agents, Palmerston North, New Zealand*. 31

McHaney, R. et D. Douglas, 1997. Multivariate regression metamodel : a DSS application in industry. *Decision Support Systems*, 19 :43–52. 88

Mehta, S. et R. Uzsoy, 1999. Predictive scheduling of a single machine subject to breakdowns. *International Journal of Computer Integrated Manufacturing*, 12(1) :15–38. 10, 13, 17

Moore, J. M., 1968. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *management science*, 15 :102–109. 7

Mouelhi, W. et H. Pierreval, 2010. Training a neural network to select dispatching rules in real time. *Computers & Industrial Engineering*, 58(2) :249–256. 27

Nearchou, A. C., 2004. The effect of various operators on the genetic search for large scheduling problems. *International Journal of Production Economics*, 88 :191–203. 41, 42, 47

Neiro, S. et J. Pinto, 2006. Lagrangean decomposition applied to multiperiod planning of petroleum refineries under uncertainty. *Latin American Applied Research*, 36(4) :213–220. 11

Nowicki, E. et C. Smutnicki, 1998. the flow shop with parallel machines : A tabu search approach. *European Journal of Operational Research*, 106(2-3) :226–253. 46

O' Donovan, R., R. Uzsoy et K. McKay, 1999. Predictable scheduling of a single machine with breakdowns and sensitive jobs. *International Journal of Production Research*, 37(18) :4217–4233. 17

Ouelhadj, D., S. Petrovic, P. Cowling et A. Meisels, 2005. Inter-agent cooperation and communication for agent-based robust dynamic scheduling in steel production. *Advanced Engineering Informatics*, 18(3) :161–172. 24

Paris, J., H. Pierreval et L. Tautou, 1996. Une méthode d'optimisation-simulation par algorithme évolutionniste en gestion de production en juste-à-temps. *Journal Européen des Systèmes Automatisés*, 30(7) :929–944. 90, 102

Patel, V., J. Ashby et J. Ma, 2002. Discrete event simulation in automotive final process system. *In proceeding of the 2002 Winter Simulation Conference, San Diego, CA,*, pages 1030–1034. 88

Pierreval, H., 1992. Expert system for selecting priority rules in flexible manufacturing systems. *Expert Systems with applications*, 5(1) :51–57. 27

Pierreval, H., 1993. Neural network to select dynamic scheduling heuristics. *Journal of decision systems*, 2(2) :173–190. 27

Pierreval, H. et J. Paris, 2000. Distributed evolutionary algorithms for simulation optimization. *IEEE Transaction on Systems, Man and Cybernetics*, 20(1) :15–24. 90, 102

Pierreval, H. et J. Paris, 2003. From simulation optimization to simulation configuration of systems. *Simulation Modelling Practice and Theory*, 11 :5–19. 90, 102

Pierreval, H. et H. Ralambondrainy, 1990. A simulation and learning technique for generating knowledge about manufacturing systems behavior. *Journal of the Operational Research Society*, 41(6) :461–474. 26

Pierreval, H. et L. Tautou, 1997. Using evolutionary algorithms and simulation for the optimisation of manufacturing systems. *IIE Transactions*, 29(3) :181–189. 90, 102

Pomerol, J., 2001. Scenario development and practical decision making under uncertainty. *Decision Support Systems*, 31 :197–204. 16

Priore, P., D. Garcia et I. Quesada, 1998. Manufacturing systems scheduling through machine learning. *Neural Computation, NC'98, Vienna, Austria*, pages 914 –917. 25, 26

Proth, J. et X. Xie, 1994. Les réseaux de petri pour la conception et la gestion des systèmes de production. *Masson, Paris*. 85

Querrec, R., S. Tarot, P. Chevaillier et J. Tisseau, 1997. Simulation d'une cellule de production : Utilisation d'un modèle à base d'agents contrôlés par réseaux de petri. *AGIS'97*, pages 209–214. 24

Rajendran, C. et O. Holthaus, 1999. A comparative study of dispatching rules in dynamic flow shops and job shops. *European Journal of Operational Research*, 116(1) :156–170. 24, 45

Rangsaritratsamee, R., J. Ferrell et M. Kurz, 2004. Dynamic rescheduling that simultaneously considers efficiency and stability. *Computers and Industrial Engineering*, 46 :1–15. 30

Reaidy, J., P. Massotte et D. Diep, 2006. Comparison of negotiation protocols in dynamic agent-based manufacturing systems. *International Journal of Production Economics*, 99 :117–130. 24

Rekiek, B., A. Dolgui, A. Delchambre et A. Bratcu, 2002. State of art of optimisation methods for assembly line design. *Annual Reviews in Control*, 26 :163–174. 87

Roubellat, F., C. Artigues et M. Villaumie, 1998. Un système interactif d'aide à la décision pour l'ordonnancement en temps réel d'atelier. *Rapport Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS), Numéro 98285*. 32

Roy, B., 1997. Un chaînon manquant en RO-AD : les conclusions robustes. *Cahier du Laboratoire d'Analyse et de Modélisation de Systèmes pour l'Aide à la Décision, Université de Paris Dauphine*, (144). 16

Ruiz, R., J. García-Díaz et C. Maroto, 2005a. Considering scheduling and preventive maintenance in the flowshop sequencing problem. *Computers and Operations Research*, 34(11) :3314 –3330. 8

Ruiz, R., C. Maroto et J. Alcaraz, 2005b. Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. *European Journal of Operational Research*, 165(1) :34–54. 46

Ruiz, R. et J. A. Vázquez-Rodríguez, 2010. The hybrid flow shop scheduling problem. *European Journal of Operational Research*, 205 :1–18. 46

Sabuncuoglu, I., 1998. A study of scheduling rules of flexible manufacturing systems : a simulation approach. *International Journal of Operational Research*, 36(2) :527–546. 25

Sabuncuoglu, I. et S. Goren, 2009. Hedging production schedules against uncertainty in manufacturing environment with a review of robustness and stability research. *International Journal of Computer Integrated Manufacturing*, 22(2) :138–157. 20

Sadeh, N., S. Otsuka et R. Schelback, 1993. Predictive and reactive scheduling with the microboss production scheduling and control system. *Proceedings of the IJCAI-93 workshop on knowledge-based production planning, scheduling and control*, pages 293–306. 29

Sahinidis, N., 2004. Optimization under uncertainty : state-of-the-art and opportunities. *Computers and Chemical Engineering*, 28 :971–983. 14

Santos, D., J. Hunsucker et D. Deal, 1995. Global lower bounds for flow shops with multiple processors. *European Journal of Operational Research*, 80 :112–120. 135

Scholl, A. et C. Becker, 2006. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research*, 168 :666–693. 87

Sevaux, M. et K. Sörensen, 2004. A genetic algorithm for robust schedules in a one-machine environment with ready times and due dates. *4OR, Springer*, 2(2) :129–147. 21

Sharifi, H. et Z. Zhang, 1999. A methodology for achieving agility in manufacturing organization an introduction. *International Journal of Production Economics*, 62 :7–22. 82

Shen, W., L. Wang et Q. Hao, 2006. Agent-based distributed manufacturing process planning and scheduling : A state-of-the-art survey. *IEEE transactions on systems, man, and cybernetics-part c : Applications and Reviews*, 36(4) :563–577. 24

Smith, S., 1994. Reactive scheduling systems. *In Brown, D. and Scherer, W. T. (Eds.), Intelligent Scheduling Systems, Kluwer Academic Publisher*, pages 155–192. 29

Snoek, M., 2001. Anticipation optimization in dynamic job shops. *Genetic and Evolutionary Computation Conference 2001 (GECCO- 2001), San Francisco, USA*, pages 43–46. 29

Spieckermann, S., H. Heinzel, S. Gutenshwager et S. VoB, 2000. Simulation-based optimization in the automotive industry : a case study on body shop design. *Simulation*, 75(5) :276–286. 89

Sörensen, K., 2001. Tabu searching for robust solutions. *Proceedings of the 4th Metaheuristics International Conference Portugal*, pages 707–712. 21

Sriskandarajah, C., 1993. Performance of scheduling algorithms for no-wait flow-shops with parallel machines. *European Journal of Operational Research*, 70(3) :365–378. 46

Strum, D., J. May et L. Vargas, 2000. Modeling the uncertainty of surgical procedure times : Comparison of the Log-normal and normal models. *Anesthesiology*, 92 :1160–1167. 12

Suresh, V. et D. Chaudhuri, 1993. Dynamic scheduling-a review. *International Journal of Production Economics*, 32 :53–63. 13

Sutclife, N., 1999. Leadership behavior and Business Process Reengineering (BPR) outcomes : An empirical analysis of 30 BPR projects. *Information and Management*, 36 :273–286. 82

Tomovic, R., 1963. Sensitivity analysis of dynamical systems. *McGraw-Hill, New York*. 12

Vieira, G., J. Herrmann et E. Lin, 2000. Analytical models to predict the performance of single-machine system under periodic and event-driven rescheduling strategies. *International journal of production research*, 38(8) :1899–1915. 29

Vignier, A., J. Billaut et C. Proust, 1999. Les problemes d'ordonnancement de type Flow-Shop Hybride : Etat de l'art. *RAIRO - Recherche Opérationnelle*, 33 :117–183. 46

Williams, E. et A. Gevaert, 1997. Pallet optimization and throughput estimation via simulation. *In Proceedings of the 1997 Winter Simulation Conference, Atlanta*, pages 750–757. 88

Wu, C., K. Brown et J. Beck, 2009a. Scheduling with uncertain durations : Modeling beta-robust scheduling with constraints. *Computers and Operations Research*, 36 :2348–2356. 19

Wu, L., X. Chen, X. Chen et Q. Chen, 2009b. The research on proactive-reactive scheduling framework based on real-time manufacturing information. *Materials Science Forum*, 626-627 :789–794. 32

Wu, S., R. Storer et P. Chang, 1993. One-machine rescheduling heuristics with efficiency and stability as criteria. *Computers and operations research*, 20 :1–14. 28, 30

Yamada, T. et R. Nakano, 1992. A genetic algorithm applicable to large scale job shop problems. *Proceedings of the Second International Conference on Parallel Problem Solving from Nature (Elsevier Science Publishers)*. 47

Zadeh, L., 1965. Fuzzy sets. *Information and Control*, 8 :338–353. 12

Zuo, X., H. Mo et J. Wu, 2009. A robust scheduling method based on a multi-objective immune algorithm. *Information Sciences*, 179 :3359–3369. 22

Annexe A

Illustration de l'algorithme génétique pour l'ordonnancement robuste à partir d'un exemple académique

Dans cette annexe, nous présentons une illustration du rôle de la fonction d'évaluation robuste présentée dans le chapitre 2 à l'aide d'un exemple académique simple.

Dans notre exemple nous considérons le problème de *flow shop* hybride à 2 étages, dans chaque étage il existe 2 machines parallèles et identiques. Le nombre de *jobs* est égal à 10. Chaque job J_j (avec $j = 1, ..., 10$) doit passer sur une des machines du premier étage, puis sur une des machines du deuxième étage.

Nous partons d'un scénario initial I qui représente les temps d'exécution des *jobs* dans chaque étage. Le degré d'incertitude est $\alpha = 10\%$. Cette perturbation concernera les temps d'exécution des *jobs* qui seront générés selon une loi uniforme $\mathcal{U}(P_{jt_I} - 10\%P_{jt_I}; P_{jt_I} + 10\%P_{jt_I})$ avec P_{jt_I} le temps d'exécution du job j dans l'étage t du scénario initial I. Le scénario initial est décrit dans le tableau A.1.

		J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9	J_{10}
Scénario Initial	Etage1	8	4	6	9	5	10	6	5	7	3
	Etage2	5	3	2	8	6	4	5	9	5	7

TAB. A.1 – Scénario initial du *flow shop* hybride à 2 étages

Notre algorithme génétique pour l'ordonnancement robuste est initialisé par :

- une taille de population égale à 200 ;
- une probabilité de croisement égale à 90% ;
- une probabilité de mutation égale à 20% ;
- un critère d'arrêt qui est le nombre de générations, égal à 1000 ;
- un poids $\lambda \in [0, 1]$ connue. Nous faisons varier la valeur λ entre 0 et 1 avec pas de 25% pour trouver les solutions pour ce problème ;
- $N \in [10, 100]$;
- $\varepsilon = 0.5$.

Tout d'abord, l'algorithme génétique pour l'ordonnancement robuste permet de chercher la solution x^r (x^r est la séquence obtenue après les 1000 générations et pour chaque génération un ensemble de scénarios perturbés est généré selon la loi uniforme $\mathcal{U}$ décrit ci-dessus). A chaque valeur de λ nous associons sa solution x^r. Par exemple, pour $\lambda = 0$, la solution $x^r = 6 - 9 - 1 - 4 - 2 - 5 - 10 - 7 - 8 - 3$ consiste à exécuter dans le premier étage le job 6, puis le job 9, suivi des *jobs* 1, 4, 2, 5, 10, 7, 8 et enfin le job 3. Dans le deuxième étage, la règle FIFO est appliquée pour tous les *jobs*. L'affectation des *jobs* sur les machines est faite par la règle FAM.

Le tableau A.2 donne, pour chaque valeur de λ, les résultats des solutions (x^r). Le $Cmax_I$ représente le *akespan* du scénario initial. La borne inférieure du scénario initial est $LB = 34$. Cette borne inférieure est développée par Santos *et al.* (1995). $DEV\_MAX = 7$ est la déviation maximale entre le *makespan* du scénario initial et les makespan des scénarios minimal et maximal (voir équation 2.5).

λ	LB	DEV_MAX	$Cmax_I$	Séquence (x^r)
$\lambda = 1$	34	7	34	5-8-1-4-10-7-6-9-3-2
$\lambda = 0.75$			36	8-5-4-10-9-7-6-1-3-2
$\lambda = 0.5$			38	8-7-4-5-6-9-10-2-1-3
$\lambda = 0.25$			41	5-1-7-4-6-8-3-9-10-2
$\lambda = 0$			43	6-9-1-4-2-5-10-7-8-3

TAB. A.2 – Les solutions (séquences) x^r obtenues par l'algorithme génétique pour l'ordonnancement robuste

Pour $\lambda = 0$ nous favorisons la minimisation de la déviation entre les scénarios perturbés $\left(\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(Cmax_{\xi_i(I)}(x) - Cmax_I(x)\right)^2}\right)$et pour $\lambda = 1$ nous favorisons la minimisation du *makespan* initial ($Cmax_I$).

Nous avons généré 10 scénarios perturbés qui suivent la même loi(loi $\mathcal{U}$) et le même degré d'incertitude $\alpha = 10\%$. Les temps d'exécution des *jobs* dans chaque étage pour chaque scénario perturbé sont présentés dans le tableau A.3. Par la suite, nous testons (vérifions), pour chaque valeur de λ, les solutions obtenues par l'algorithme génétique pour l'ordonnancement robuste sur ces 10 scénarios perturbés, pour voir l'impact des solutions robustes (x^r) sur ces scénarios perturbés.

		J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9	J_{10}
Scénario $\xi_1(I)$	Etage1	8	4	6	10	5	11	7	5	8	3
	Etage2	6	3	2	9	6	5	5	10	6	7
Scénario $\xi_2(I)$	Etage1	9	5	6	9	5	10	6	6	8	4
	Etage2	5	3	3	9	6	5	5	9	5	8
Scénario $\xi_3(I)$	Etage1	8	5	7	9	5	11	6	6	8	4
	Etage2	6	3	3	9	6	5	5	9	5	8
Scénario $\xi_4(I)$	Etage1	9	5	6	9	5	10	6	5	8	3
	Etage2	5	4	3	8	7	4	5	10	6	7
Scénario $\xi_5(I)$	Etage1	9	5	6	10	6	11	7	6	7	4
	Etage2	6	3	3	8	7	5	5	9	5	8
Scénario $\xi_6(I)$	Etage1	9	5	7	10	5	10	6	5	7	3
	Etage2	6	3	3	9	6	5	5	10	6	7
Scénario $\xi_7(I)$	Etage1	9	5	6	9	5	9	7	6	8	4
	Etage2	6	4	3	9	7	5	6	10	5	8
Scénario $\xi_8(I)$	Etage1	9	5	7	10	5	10	6	6	8	4
	Etage2	5	4	3	8	7	4	6	9	6	7
Scénario $\xi_9(I)$	Etage1	8	5	7	9	6	10	6	6	7	4
	Etage2	5	4	2	9	7	4	6	9	5	8
Scénario $\xi_{10}(I)$	Etage1	9	4	6	9	6	9	7	5	7	4
	Etage2	6	4	3	9	7	4	5	9	5	8

TAB. A.3 – Les scénarios perturbés avec $\alpha = 10\%$ générés à partir du scénario initial

Le tableau A.4 présente, pour chaque valeur de λ, l'évaluation des 10 scénarios perturbés par la solution x^r. C'est-à-dire, pour chaque valeur de λ, nous appliquons la séquence x^r (présentée dans le tableau A.2) sur les 10 scénarios perturbés seulement dans le premier étage. Dans le deuxième étage, nous appliquons la règle FIFO. L'affectation des *jobs*

dans les étages est faite par la règle FAM. Ce fonctionnement est le même que le fonctionnement de l'algorithme génétique pour l'ordonnancement robuste. La dernière colonne représente la déviation entre les 10 scénarios perturbés et le scénario initial.

λ	$Cmax_I$	$Cmax_{\xi_1}$	$Cmax_{\xi_2}$	$Cmax_{\xi_3}$	$Cmax_{\xi_4}$	$Cmax_{\xi_5}$	$Cmax_{\xi_6}$	$Cmax_{\xi_7}$	$Cmax_{\xi_8}$	$Cmax_{\xi_9}$	$Cmax_{\xi_{10}}$	Déviation
$\lambda = 1$	**34**	38	37	38	37	39	38	40	39	37	37	**4.12**
$\lambda = 0.75$	**36**	38	37	38	38	40	38	40	39	38	39	**2.66**
$\lambda = 0.5$	**38**	41	39	40	38	42	39	40	39	40	40	**2.09**
$\lambda = 0.25$	**41**	42	42	43	41	44	42	43	43	43	43	**1.78**
$\lambda = 0$	**43**	43	45	43	43	46	43	45	44	44	43	**1.37**

TAB. A.4 – Évaluation des scénarios perturbés

La fonction d'évaluation robuste $f^r(x)$ développée dans la section 2.6.1 consiste à minimiser simultanément le $Cmax_I$ et la déviation entre les *makespan* des scénarios perturbés et le *makespan* du scénario initial. Nous remarquons que, plus λ augmente, plus le *makespan* du scénario initial diminue, plus la déviation augmente. Ceci résulte de fait que, plus λ augmente, plus la priorité est donnée à la minimisation du scénario initial et donc il s'approche de LB. Inversement pour la déviation, plus λ augmente plus la déviation entre les scénarios perturbés et le scénario initial augmente et donc le coût de la robustesse augmente.

Pour $\lambda = 1$, nous favorisons la minimisation du *makespan* initial ($Cmax_I$). Cette valeur peut être obtenue non seulement par l'algorithme génétique pour l'ordonnancement robuste, mais aussi par l'algorithme génétique pour l'ordonnancement déterministe. Parce que, dans les deux algorithmes, la fonction d'évaluation est la même (minimisation du ($Cmax_I$)) et les perturbations n'interviennent pas. En effet, la solution trouvée est appelée solution efficace.

Pour $\lambda = 0$, nous favorisons la minimisation de la déviation entre les *makespan* des scénarios perturbés et le *makespan* du scénario initial. En effet, la solution trouvée est nommée solution robuste.

Dans le cas où le poids attribué à chaque objectif est le même ($\lambda = 0,5$), nous obtenons des solutions dites de *compromis*, entre efficacité et robustesse.