UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À

L'UNIVERSITÉ DU QUÉBEC À CHICOUTIMI

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

par

STEFAN MANUEL GOMES MONTEIRO

MODE DE CONSTRUCTION À REBOURS DANS UN ALGORITHME D'OPTIMISATION PAR COLONIE DE FOURMIS POUR LA MINIMISATION DU RETARD TOTAL

15 juillet 2010

RÉSUMÉ

Le problème de machine unique avec temps de réglage dépendants de la séquence est un problème d'ordonnancement industriel où il y a une machine qui peut exécuter différentes tâches. Le changement de tâches à exécuter entraîne un réglage de la machine comme c'est le cas pour les industries de transformation de papier ou les industries pharmaceutiques. De plus, le temps pour réaliser ce réglage dépend de la tâche courante et de la tâche suivante. Le but du problème d'ordonnancement à l'étude est de déterminer l'ordre des tâches de façon à minimiser le retard total.

Ce problème est de nature NP – difficile et sa résolution passe par l'utilisation des métaheuristiques. Plusieurs métaheuristiques ont été proposées dans la littérature comme les algorithmes d'optimisation par colonie de fourmis (OCF), plus précisément les ACS. Ce mémoire propose un nouveau mode de construction de solutions pour l'ACS pour le problème de machine unique avec temps de réglage dépendants de la séquence pour la minimisation du retard total. En effet, compte tenu de la nature de l'objectif à optimiser, nous privilégions un mode de construction de solution à rebours. Pour amorcer une construction à rebours de la séquence, un nouveau concept de visibilité est proposé, avec l'utilisation d'une marge arrière qui permet de favoriser le placement des tâches en retard à la fin de la séquence. Cette nouvelle visibilité a été intégrée à une version ACS existant dans la littérature et a été nommée ACS à rebours. Des modifications ont été apportées à l'ACS à rebours pour améliorer son efficacité. La première modification a pour objectif de rendre l'ACS à rebours plus intelligent au niveau du choix des paramètres associés à la règle de transition et de rendre ainsi son utilisation plus facile. La seconde modification consiste à utiliser une règle de priorité au moment où les tâches restantes à placer ne sont plus en retard pour tenter d'accélérer la construction d'une solution.

Des expérimentations numériques ont été réalisées pour comparer la performance de l'ACS à rebours avec celle de l'un des ACS présentés dans la littérature pour le problème à l'étude. Pour les instances de problème de petite taille, la performance de l'ACS à rebours est semblable à celle de l'ACS classique. Ceci nous porte à croire que l'idée de base ouvre une voie intéressante pour le développement des travaux futurs avec les ACS qui utilisent un mode de construction à rebours de la séquence. Pour les instances de problème de grande taille, un avantage doit être accordé à l'ACS classique, ce qui atteste que des améliorations peuvent être apportées à l'ACS à rebours surtout au niveau de la diversification des solutions produites.

Ce travail de recherche représente une première exploitation du concept de construction de solutions à rebours de la séquence pour les ACS. Le présent mémoire est

une contribution non seulement à une meilleure connaissance des ACS, mais aussi à la connaissance de nouveaux modes de construction pour les ACS.

REMERCIEMENTS

Je tiens tout d'abord à remercier mes parents, qui m'ont toujours fait confiance et qui m'ont fourni tout ce dont j'avais besoin pour que je puisse franchir cette nouvelle étape de ma vie dans les meilleures conditions. Sans votre appui, je ne serais pas là aujourd'hui. Un grand merci à mon oncle António qui m'a continûment encouragé à poursuivre mes études et qui m'a soutenu tout au long de mon séjour au Canada, malgré les moments très difficiles qu'il a vécus.

Je remercie aussi ma compagne Danize et mon fils Ismael pour les sacrifices qu'ils ont acceptés de faire pour que je puisse poursuivre mes études à l'étranger. Sans votre amour et votre soutien, tout serait plus difficile. J'aimerais également dédier ce travail à mon fils qui a dû se priver de ma compagnie pendant ces deux dernières années.

J'exprime ma sincère gratitude à ma directrice de recherche Mme Caroline Gagné et à mon codirecteur M. Marc Gravel pour leur disponibilité, implication et appui tout au long de ce travail de recherche. Sans leurs conseils avisés et leur dynamisme, ce travail n'aurait sans doute pas vu le jour.

Un travail de recherche ne peut aboutir sans support financier. À ce titre, je voudrais remercier l'Agence Canadienne de Développement International (ACDI) qui, par le biais du programme canadien de bourses de la francophonie (PCBF), a entièrement financé mon séjour durant ces deux années d'études.

Je remercie aussi tous les membres du Group de Recherche en Informatique de l'UQAC (GRI) pour les bons moments passés ensemble, plus particulièrement Aymen

Sioud, mon compagnon de bureau, pour ses conseils avisés et pour son soutien tout au long de ma maîtrise.

Pour terminer, j'ai une pensée spéciale pour ma tante Antonina et pour mon oncle Jancénio qui nous ont quitté pendant mon séjour au Canada. Que la terre leur soit légère.

Table des Matières

Liste des Tableaux

Liste des Figures

CHAPITRE 1

Introduction

L'ordonnancement est un processus de prise de décision qui joue un rôle très important dans les industries de fabrication et les entreprises de prestation de services [Pinedo et Chao 1998]. L'ordonnancement traite de la gestion des ressources qui sont limitées dans les organisations et a comme but d'optimiser une ou plusieurs mesures de performance. On trouve de tels problèmes, par exemple, dans les usines de fabrication de semi-conducteurs, les usines d'assemblage de voitures, les systèmes de réservation ou même dans la gestion des tâches exécutées par les unités centrales de traitement (CPU) des ordinateurs (Pinedo 2002).

Les problèmes classiques d'ordonnancement sont caractérisés par un ensemble de M machines, un ensemble de N tâches à exécuter, une durée de traitement p_{ij} de la tâche i sur la machine j et une date due de la tâche (due date) d_i. Pour faciliter la classification des problèmes d'ordonnancement, Graham, Lawler *et al.* [1979] ont proposé de classer l'ensemble de ces problèmes en utilisant les paramètres α (alpha), β (bêta), γ (gamma) où α définit le modèle de machines, β spécifie les contraintes et γ précise les mesures de performance.

Les problèmes d'ordonnancement sont intimement liés aux modèles de machine. On a ainsi le modèle à machine unique, le modèle à machines parallèles, le modèle sériel (flow-shop), le modèle d'atelier ouvert (open-shop) et le modèle d'atelier général (job-shop) [Pinedo et Chao 1998].

Le modèle à machine unique est un modèle où il y a une machine qui peut exécuter différentes tâches comme c'est le cas pour les ateliers de transformation de papier. Le problème peut aussi être représenté par une machine goulot dans un modèle plus complexe, qui, en cas d'embouteillage, affecte la performance du système. Un problème

complexe peut ainsi être fractionné en plusieurs petits problèmes d'ordonnancement à machine unique. L'ordonnancement de la machine goulot peut alors être appliqué à l'ensemble des autres machines.

Le modèle à machines parallèles est une généralisation du modèle à machine unique. Les machines peuvent être identiques et fonctionner en parallèle, ce qui permet aux tâches d'être exécutées par n'importe quelle machine disponible. Toutefois, dans le cas où les machines ne sont pas identiques, certaines d'entre elles peuvent être plus performantes au niveau de la qualité, de la maintenance ou de la rapidité d'exécution.

L'atelier sériel est un modèle où les tâches sont exécutées par plusieurs machines successives. Ces tâches sont exécutées par toutes les machines disponibles même si l'ordre d'exécution par machine peut varier. Il existe une version flexible de l'atelier sériel qui permet d'avoir plusieurs machines identiques ou différentes à chaque étape, ce qui permet d'exécuter les tâches sur des machines parallèles.

L'atelier général permet d'avoir des tâches qui sont exécutées en suivant des chemins différents. Ce modèle est une généralisation de l'atelier sériel. Il permet à une tâche d'être exécutée par la même machine plusieurs fois, ce qui augmente la complexité du problème à cause des conflits qui peuvent survenir quand plusieurs tâches doivent utiliser la même machine au même moment.

L'atelier ouvert est un modèle qui diffère de l'atelier sériel et de l'atelier général par le fait qu'il n'existe pas de restriction sur l'ordre de traitement de chaque tâche. C'est le cas, par exemple, des procédures de test utilisées par les ateliers d'assemblage où différents équipements sont utilisés pour tester différentes fonctionnalités quand il n'est pas possible d'effectuer deux tests en même temps.

Dans les usines, il peut y avoir aussi des configurations hybrides où les différents modèles peuvent coexister. De plus, dans le monde réel, la grande majorité des environnements de production sont plus complexes que ce qui a été présenté ici.

Le deuxième élément caractérisant un problème d'ordonnancement concerne les contraintes associées aux problèmes. Celles-ci peuvent prendre la forme de contraintes de précédence entre les tâches, de routage pour une tâche, de manutention d'un centre de travail à un autre, d'interruption d'une tâche pour laisser passer une tâche urgente, d'espace d'entreposage limité, d'utilisation d'une machine particulière dans un modèle de machines parallèles, parmi d'autres. Ces contraintes peuvent aussi être reliées aux outils utilisés pour fabriquer ou manipuler les matières premières ou encore relatives au réglage de la machine avant l'exécution d'une tâche [Pinedo et Chao 1998].

Les problèmes d'ordonnancement qui prennent en considération la notion de temps de réglage peuvent être divisés en deux groupes, le premier étant le temps de réglage dépendant de la séquence et le deuxième, le temps de réglage indépendant de la séquence (Allahverdi, Gupta *et al.* 1999).

Le temps de réglage dépendant de la séquence se présente lorsque les tâches doivent être traitées par plusieurs machines et que la durée ne dépend pas seulement de la tâche à traiter, mais aussi de la tâche précédente. Cette situation est courante dans plusieurs industries, comme c'est le cas de l'industrie de fabrication de composés chimiques où l'ampleur du nettoyage dépend à la fois du produit chimique le plus récemment traité et des produits chimiques qui seront traités par la suite. L'industrie d'impression passe aussi par le même problème parce que le nettoyage et le réglage de la machine pour le traitement de la prochaine tâche dépend de la différence de la couleur de

l'encre, de la taille du papier et des types utilisés dans l'emploi précédent (Eren et Güner 2006).

Pour le temps de réglage indépendant de la séquence, la durée de traitement dépend seulement de la tâche à exécuter. Le temps de réglage étant le même pour chaque tâche, celui-ci peut alors être additionné au temps de traitement.

Le troisième élément caractérisant un problème d'ordonnancement concerne les mesures de performance. Parmi les mesures de performance utilisées pour évaluer la qualité d'un ordonnancement, on note, le temps de fin du dernier travail, le retard total pondéré ($\sum_{j=1}^{n} w_j T_j$) ou non pondéré($\sum_{j=1}^{n} T_j$), la somme du temps d'achèvement des tâches pondéré ($\sum_{j=1}^{n} w_j C_j$) ou non pondéré ($\sum_{j=1}^{n} T_j$), le retard maximal $Tmax$, le temps maximum passé en usine $Fmax$, entre autres.

Selon Esquirol et Lopez [1999], les problèmes d'ordonnancement pour les modèles traditionnels sont de nature NP-difficile et ne sont pas de difficulté identique. Ainsi, la solution optimale à un problème d'ordonnancement ne peut que très rarement être déterminée en temps polynomial. Selon Baptiste, Giard *et al.* [2005], il est nécessaire de trouver des méthodes de résolution qui permettent de trouver une solution de bonne qualité dans un intervalle de temps raisonnable. Ce fait peut être atteint en utilisant les heuristiques, o plus précisément une technique plus efficace et générale appelée les métaheuristiques.

Pour les métaheuristiques, il existe trois grandes approches utilisées pour résoudre un problème d'ordonnancement [Lopez et Roubellat 2001]. L'approche constructiviste est une méthode de recherche qui consiste à diminuer la taille du problème en le subdivisant

en plusieurs sous-problèmes, ce qui permet de trouver une solution à partir de la solution de chacun des sous-problèmes. Cette approche est simple et rapide à mettre en œuvre, mais la qualité des solutions n'est généralement pas très bonne à cause de la difficulté de sortir des optimums locaux, ce qui rend difficile l'obtention de solutions optimales. La deuxième approche regroupe les méthodes plus sophistiquées utilisant la recherche locale, comme le recuit simulé, les méthodes d'acceptation à seuil et la recherche avec tabous. La troisième approche est l'approche évolutive qui, contrairement aux méthodes constructives et de recherche locale, manipule un groupe de solutions admissibles à chacune des étapes du processus de recherche en utilisant une population de solutions pour guider efficacement la recherche. Les algorithmes génétiques, la recherche distribuée et l'algorithme d'optimisation par colonie de fourmis sont des méthodes évolutives. L'approche hybride est une méthode de recherche combinée qui permet de joindre différentes approches pour tirer profit des points forts de chaque approche.

Dans ce mémoire, nous nous intéressons plus particulièrement au modèle à machine unique avec temps de réglage dépendants de la séquence dans l'objectif de minimiser le retard total. Ce modèle est noté $1 \mid s_{ij} \mid \sum T_j$ selon la classification de Graham, Lawler *et al.* [1979]. Plusieurs auteurs ont proposé différentes méthodes pour sa résolution. Ragatz (1993) et Gagné, Price *et al.* [2002] ont proposé des ensembles de problèmes tests permettant d'évaluer et de comparer la performance des algorithmes développés.

Au niveau des méthodes exactes, Ragatz (1993) a proposé un algorithme de séparation et d'évaluation progressive (branch and bound) dont la performance dépend de la taille et des caractéristiques du problème. Cette méthode a permis de trouver les solutions optimales pour de petites instances de problèmes. Pour leur part, Bigras,

Gamache *et al.* (2008) ont trouvé la solution optimale pour toutes les instances de problèmes proposées par Ragatz (1993) en utilisant la même méthode avec une relaxation lagrangienne.

Au niveau des métaheuristiques, Rubin et Ragatz [1995] et Armentano et Mazzini [2000] ont proposé des algorithmes génétiques, Tan et Narasimhan (1997) ont développé un algorithme de recuit simulé, França, Mendes *et al.* [2001] ont proposé un algorithme mémétique, Gagné, Gravel *et al.* [2005] ont proposé une hybridation de la recherche avec tabous et de la recherche à voisinage variable (VNS), Gagné, Price *et al.* [2002] et Liao et Juan [2007] ont développé un algorithme d'optimisation par colonie de fourmis, Lin et Ying [2008] ont proposé une hybridation de la recherche avec tabous et le recuit simulé, Gupta et Smith [2006] ont développé une heuristique de recherche locale basée sur l'espace et un «Greedy Randomized Adaptative Search Procedure » (GRASP).

L'objectif principal de ce mémoire vise à proposer une adaptation d'un algorithme d'optimisation par colonie de fourmis au problème $1 \mid s_{ij} \mid \sum T_j$. En effet, nous envisageons de modifier le comportement de l'ACS (Ant Colony System) au niveau de la construction des solutions pour privilégier une construction à rebours de la séquence, de façon à tenir en compte plus particulièrement la mesure de performance qui est le retard total.

Dans le chapitre 2, on aborde les spécificités du problème de machine unique avec temps de réglage dépendants de la séquence, les principaux travaux réalisés dans ce domaine et, plus particulièrement, l'utilisation des algorithmes d'optimisation par colonie de fourmis. Les objectifs de la recherche y sont également précisés. Dans le chapitre 3, on présente un nouveau mode de construction de solution à rebours avec l'ACS pour le problème à l'étude, en plus de proposer des raffinements à ce mode de construction. Le

chapitre 4 présente les expérimentations numériques réalisées avec les algorithmes proposés et une comparaison de performance avec l'ACS. Le mémoire se termine par une conclusion dans le chapitre 5.

CHAPITRE 2

Revue de la littérature

2.1 Introduction

Le modèle à machine unique consiste en un ensemble de *N* tâches en attente de la disponibilité d'une machine, tel qu'illustré à la Figure 2.1. Plusieurs auteurs ont abordé ce problème d'ordonnancement en considérant diverses formes de contraintes et selon plusieurs mesures de performance.

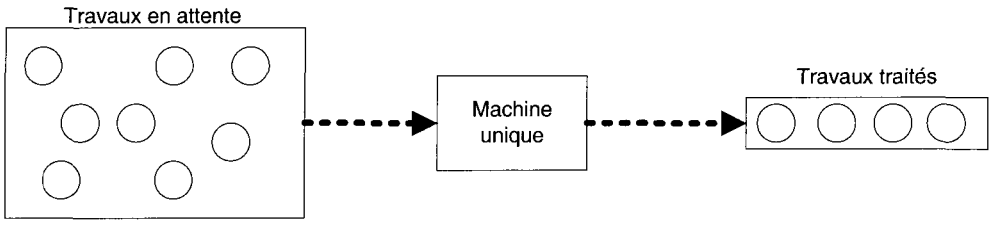

Figure 2.1 – Modèle à machine unique

Pour minimiser le temps de fin du dernier travail (makespan), avec des contraintes de disponibilité et de livraison, Florian, Trepant *et al.* [1971] ont proposé un algorithme de séparation et d'évaluation progressive. Ils ont proposé l'utilisation d'une borne inférieure qui peut être obtenue par l'ajout du temps de traitement de la plus grande tâche au plus petit temps de livraison.

Pour minimiser le temps total passé en usine avec pondération sur les travaux et contraintes de précédence, Conway, Maxwell *et al.* (1967) ont considéré une structure de précédence à chaîne parallèle et ont développé un algorithme pour le cas des pondérations égales. Pour les contraintes de précédence générale, Lenstra et Rinnooy (1980) ont prouvé que le problème est NP-Complet même si tous les temps de traitement sont égaux ou si toutes les pondérations sont égales.

Le nombre de tâches en retard est une autre mesure de performance qui est traitée dans la littérature. L'algorithme optimal pour ce problème a été proposé par Moore (1968)

et Maxwell (1970). Pour leur part, Baker et Nuttle [1980] ont pris en compte la minimisation du nombre de tâches en retard en considérant que la disponibilité des machines varie au fil du temps. Jackson (1955) et Smith (1956) ont démontré que si le temps de disponibilité des tâches est le même pour toutes les tâches, le retard maximal est alors minimisé par la règle de la séquence par les échéances (EDD).

Le retard total est une mesure de performance qui fait en sorte que le problème d'ordonnancement d'une machine unique est considéré comme étant de nature NP-difficile. Dans ses premiers travaux, Emmons (1969) a développé trois théorèmes pour réduire la taille des problèmes à machine unique. Srinivasan (1971) a développé un algorithme hybride à trois phases pour les problèmes à plus de cinquante tâches en utilisant les théorèmes de Emmons (1969) pour réduire la taille des problèmes. Pour sa part, Baker [1977] a présenté un algorithme de chaîne basé sur la programmation dynamique.

La minimisation des temps de réglage est une mesure qui a été largement étudiée en l'associant à un problème de voyageur de commerce [Gilmore et Gomory 1964] dans le cas où ces temps de réglage sont dépendants de la séquence.

Pour les problèmes avec temps de réglages indépendants de la séquence, Graves et Lee [1999] ont assumé que, pour le retard maximal et le temps d'achèvement total pondéré, si les tâches n'ont pas été terminées avant le début de la prochaine date de maintenance, un temps de réglage additionnel doit être ajouté à la reprise de traitement des tâches manquantes. Les problèmes ont été traités par des algorithmes de programmation dynamique à temps pseudo-polynomial.

Gupta et Sivakumar [2005] ont traité les problèmes avec temps de réglage dépendants de la séquence et de la date d'échéance de façon multi-objective. Les objectifs pris en compte sont la minimisation du temps de cycle moyen, la minimisation du retard moyen et la maximisation de l'utilisation des machines. La solution Pareto optimale est obtenue en combinant une approche d'ordonnancement simulée basée sur le contexte et sur un point idéal. La solution proposée est comparable aux heuristiques qui utilisent les règles de répartition comme la séquence par la durée la plus courte (STP) et la séquence par les échéances (EDD).

Dans ce mémoire, nous nous intéressons plus particulièrement à la minimisation du retard total, qui prend en compte le temps de réglage entre deux tâches qui se succèdent et qui est noté comme étant $1 \mid s_{ij} \mid \sum T_j$. La prochaine section présente une revue de la littérature des principaux travaux réalisés sur ce problème.

2.2 Modèle à machine unique avec temps de réglage dépendants de la séquence

Pour le problème d'ordonnancement à machine unique, le temps de réglage peut être défini comme une structure spéciale où la tâche i est associée à deux paramètres a_i et b_i. Après avoir complété la tâche i, la machine est en état b_i et pour qu'elle soit prête à initier la tâche j, la machine doit être mise en état a_j. Le temps de réglage total pour amener la machine de l'état b_i à l'état a_j est proportionnel à la différence absolue entre les deux états qui est notée $s_{ij} = | a_j - b_i |$. Cette variable d'état peut être, par exemple, la température d'un four ou une autre configuration de la machine (Pinedo 2002).

Selon Baker (1992), le retard total (tardiness) est une mesure de performance régulière où les résultats peuvent changer avec l'augmentation ou la diminution des temps d'achèvement des tâches. C'est un problème de nature combinatoire où même un petit problème a un nombre important de solutions possibles, ce qui fait de lui un problème de nature NP-difficile. L'inclusion des séquences dépendantes dues aux temps de réglage augmente la complexité du problème, ce qui signifie qu'il s'agit d'un problème de nature NP-difficile plus complexe [Rubin et Ragatz 1995].

Si on a N tâches toutes disponibles pour l'exécution à $t = 0$ et une machine unique qui fait le traitement d'une tâche à la fois, pour chaque tâche j, il y a un temps d'exécution séquentiel p_j, une date due d_j et un temps de réglage s_{ij} lorsque la tâche j suit la tâche i dans la séquence d'exécution. La séquence des tâches est définie par $Q = [Q(0), Q(1), \ldots, Q(N)]$, où $Q(j)$ représente la $j^{\text{ième}}$ tâche de la séquence, ce qui nous permet de dire que le temps d'achèvement de la tâche est $C_{Q(j)} = \sum_{k=1}^{j}[s_{Q(k-1)Q(k)} + p_{Q(k)}]$. Le retard de la $j^{\text{ième}}$ tâche de la séquence est alors défini par $t_{Q(j)} = max\{0, C_{Q(j)} - d_{Q(j)}\}$ et le retard total de la séquence est $T_Q = \sum_{j=1}^{N} t_{Q(j)}$ Rubin et Ragatz [1995] et Tan, Narasimhan *et al.* [2000].

Dans la littérature, il existe plusieurs travaux qui abordent le problème de machine unique avec temps de réglage dépendants de la séquence. (Ragatz 1993) et [Gagné, Price *et al.* 2002] ont proposé des ensembles de problèmes tests permettant d'évaluer et de comparer la performance des algorithmes développés. Ces 64 instances de problèmes comportant entre 15 et 85 tâches ont été utilisées à maintes reprises par différents auteurs.

Les problèmes proposés par Ragatz (1993), nommés problèmes de petite taille, sont divisés en quatre groupes et chaque groupe est constitué de huit instances par problème comportant respectivement 15, 25, 35 et 45 tâches. Chaque instance est définie par trois facteurs, à savoir, la variabilité du temps de traitement, le facteur de retard et l'intervalle des dates d'échéance. Chaque facteur a deux niveaux qui sont forts et faibles. La variété du temps de traitement a été fixée comme étant basse ou élevée, le facteur de retard a été fixé comme étant bas ou modéré et l'intervalle des dates d'échéance a été fixé comme étant étroit ou large. Les temps de traitement des tâches sont normalement distribués avec une moyenne de 100 unités de temps et les temps de réglages sont uniformément distribués avec une moyenne 9.5 unités de temps.

Les problèmes proposés par Gagné, Price *et al.* [2002] ont été générés aléatoirement en utilisant les procédures définies par Ragatz (1993) et sont nommés problèmes de grande taille. Ces instances sont aussi divisées en quatre groupes constitués de huit instances, mais comportent, dans ce cas, 55, 65, 75 et 85 tâches respectivement.

Rubin et Ragatz (1995) ont développé un algorithme génétique avec une population de 40 individus. La population initiale est générée aléatoirement et le choix de la nouvelle population est fait de façon élitiste et utilise aussi un «random migrant». L'opérateur de croisement utilisé consiste à sélectionner des blocs d'allèles des parents pour composer un nouveau fils. Cette opération est faite de façon à garantir que les enfants produits sont toujours des solutions réalisables. En ce qui concerne la mutation, elle est faite en utilisant l'échange de paires nommé «random start». L'algorithme génétique est plus rapide et a produit de bons résultats par rapport à l'algorithme de séparation et d'évaluation

progressive proposé par Ragatz (1993) pour les problèmes avec facteur de retard modéré et pour les problèmes avec un large intervalle par rapport aux dates d'échéance.

Tan et Narasimhan [1997] ont utilisé le recuit simulé et ont défini à priori trois paramètres classiques de l'algorithme en fonction des expérimentations précédentes de Rubin et Ragatz [1995]. Ces paramètres sont: la température initiale, la température finale et le degré de refroidissement. Au début, les valeurs de la température initiale et le degré de refroidissement sont calibrés de façon que les niveaux d'acceptation et de rejet des solutions moins bonnes soient égaux. Au fur et à mesure que la température baisse, le taux d'acceptation des solutions moins bonnes s'approche de zéro. La solution initiale a été générée aléatoirement et, à partir de là, il y a un échange de deux éléments qui est effectué où les tâches à échanger sont tirées aléatoirement. Après chaque échange, la solution est évaluée et si elle permet d'améliorer le résultat de la fonction objectif, elle est automatiquement acceptée. Sinon, la solution est acceptée en fonction d'une probabilité et de la température actuelle. Pour le problème avec un niveau de retard modéré, une faible variance pour le temps de traitement et un large intervalle pour les dates d'échéancier, le retard total est de 10,4% inférieur à l'algorithme de séparation et d'évaluation progressive proposé par Ragatz (1993) et de 10,04% mieux que ceux produits pas l'algorithme génétique de Rubin et Ragatz [1995].

França, Mendes *et al.* [2001] ont conçu un algorithme mémétique pour le problème en introduisant une structure ternaire pour la population. Cette structure est constituée de plusieurs «clusters» contenant chacun une solution «leader» et trois solutions «supporters». La solution «leader» a une évaluation plus importante que celle de ses «supporters», c'est-à-dire que les individus mieux adaptés se trouvent dans la partie

supérieure des «clusters». De nouveaux individus sont constamment générés et des ajustements sont faits pour maintenir la structure ordonnancée. En ce qui concerne le croisement, les auteurs ont conclu que le croisement par ordre (OX) est la méthode la plus performante et que la mutation ne joue pas un rôle important dans le processus évolutionnaire si le schéma de croisement est bien adapté au problème. L'algorithme mémétique a produit de meilleurs résultats que l'algorithme génétique proposé par Rubin et Ragatz [1995] et que l'ATCS de Lee, Bhaskaran *et al.* [1997]. Pour les instances des problèmes les plus grands, l'algorithme mémétique a besoin de plus de temps pour atteindre de bons résultats.

Gagné, Price *et al.* [2002] ont abordé le problème avec la version ACS de l'algorithme d'optimisation par colonie de fourmis. Celle-ci utilise une règle de transition à la fois probabiliste et déterministe selon un seuil fixé. Cette règle de transition utilise deux visibilités à savoir les réglages normalisés et les marges normalisées. Elle intègre également une fonction de regard vers l'avant (lookahead) qui est une borne inférieure de retard total pour une solution partielle. Les résultats produits par l'ACS sont comparés au RSPI de Rubin et Ragatz [1995] et, pour les problèmes de 15 et 25 tâches, le RSPI produit de meilleurs résultats. En contrepartie, pour les problèmes de grande taille, l'ACS est plus performant. En termes de temps d'exécution, l'ACS est plus rapide que le RSPI pour toutes les instances de problèmes.

Gagné, Gravel *et al.* [2005] ont proposé une hybridation avec la recherche avec tabous et le VNS. L'algorithme est inspiré de la recherche avec tabous et profite de la méthode de gestion des voisins utilisée par le VNS. Les problèmes ont été testés en fonction de deux objectifs qui sont la minimisation du retard total et la minimisation du

temps de réglage. Plusieurs structures de voisinage comme le 3opt-restreint, le or-opt, le hyper-opt et le RSPI ont été implémentées pour chacun des objectifs. Le or-opt s'est avéré le plus performant par rapport aux autres structures et a démontré être très adapté au problème à l'étude.

Pour les instances de problèmes proposées par Ragatz (1993), les résultats produits par la recherche avec tabous ont été comparés à l'algorithme de séparation et d'évaluation progressive conçu par les propres auteurs et, la recherche tabou a donné de meilleurs résultats pour 11 instances de problèmes. La recherche avec tabous a été aussi testée pour les instances de problèmes de grande taille proposées par Gagné, Price *et al.* [2002] avec l'ACS de Gagné, Price *et al.* [2002] et elle a donné de meilleurs résultats pour presque toutes les instances de problèmes. Il faut aussi dire que, jusqu'à présent, c'est l'algorithme le plus performant de la littérature.

Liao et Juan [2007] ont aussi résolu le problème avec un OCF dans l'objectif de minimisation du retard total pondéré et non pondéré. La version proposée est un ACS qui utilise un seuil minimal de phéromone tel que proposé pour le max-min ant system (MMAS). La règle de construction déterministe et probabiliste prend en compte deux matrices qui sont la matrice de phéromone et une matrice construite à partir de l'heuristique «Apparent Tardiness Cost with Setup» (ATCS). La recherche locale est une combinaison de l'échange de paires (IT) et de l'insertion dans le voisinage (IS). Le IT consiste à échanger les tâches en position y et z et le IS permet d'insérer la tâche qui est à la position y à la position z. Les résultats obtenus sont comparés au RSPI de (Rubin and Ragatz 1995), à l'OCF de Gagné, Price *et al.* [2002], et au Tabou-VNS proposé par

Gagné, Gravel *et al.* [2005]. L'algorithme proposé a donné de meilleurs résultats que l'ACS de Gagné, Price *et al.* [2002] pour huit instances de problèmes.

Lin et Ying (2008) ont proposé une hybridation avec le recuit simulé et la recherche avec tabous pour le problème. L'objectif est d'utiliser les listes taboues et la fonction de probabilité d'acceptation d'une solution utilisée par le recuit simulé. La solution initiale est générée aléatoirement et, à partir de cette solution l'algorithme génère un voisin. Si la solution générée n'est pas taboue, elle est évaluée. Dans le cas où la solution est meilleure, elle est acceptée. Sinon, la fonction d'acceptation du recuit simulé est utilisée pour accepter ou refuser la solution. Si la solution est taboue, le critère d'aspiration est utilisé. Les résultats produits par l'algorithme sont très compétitifs car les résultats obtenus pour les instances de petite et moyenne tailles sont très semblables à ceux du Tabou-VNS de Gagné, Gravel *et al.* [2005]. Pour les instances de grande taille, elles sont un peu moins performantes mais l'écart n'est pas très significatif.

Bigras, Gamache *et al.* [2008] ont développé un algorithme de séparation et d'évaluation progressive pour trouver la solution optimale pour le problème à l'étude en utilisant la relaxation lagrangienne. L'algorithme a réussi à trouver les solutions optimales pour les instances de petite et moyenne tailles proposées par Ragatz (1993). En ce qui concerne les temps d'exécution, l'optimum pour les problèmes de 15 tâches est obtenu dans un intervalle de temps qui varie entre une et vingt-sept secondes. Pour les problèmes de 25 tâches, le temps maximal est de 582 secondes et, pour les instances de 35 et 45 tâches, les temps d'exécution pour certaines instances de problèmes sont parfois supérieurs à 100 000 secondes.

Plusieurs métaheuristiques ont été développées pour le problème $1 \mid s_{ij} \mid \sum T_j$ mais nous nous intéressons, dans ce mémoire, plus particulièrement, aux algorithmes d'optimisation par colonie de fourmis. À la section suivante, les différentes versions de l'algorithme d'optimisation par colonie de fourmis sont présentées en mettant en évidence la version ACS. On présente aussi deux adaptations de l'ACS qui ont été faites pour le problème à l'étude.

2.3 Les algorithmes de fourmis

Les fourmis sont des insectes qui vivent en colonie et de façon très organisée. Chaque fourmi semble avoir son propre agenda pour la recherche des aliments [Bonabeau, Dorigo *et al.* 1999]. Les fourmis quittent leur nid et se déplacent principalement de manière aléatoire. Lorsqu'une fourmi trouve l'emplacement d'une source de nourriture, elle dépose sur le chemin menant au nid une substance chimique appelée phéromone. Celle-ci guide les autres fourmis vers la même source de nourriture et ces dernières déposent à leur tour de la phéromone pour communiquer et créer un effet de renforcement du marquage de la piste empruntée. À long terme, il est possible d'observer une optimisation de la collecte de nourriture par le marquage de plus en plus important de la piste la plus fréquentée. Les autres chemins présentant une distance plus longue sont abandonnés et la phéromone a tendance à s'évaporer. La convergence vers le chemin le plus court est possible par la communication de l'information au sein de la collectivité. L'algorithme d'optimisation par colonie de fourmis (OCF) tente de reproduire ce comportement évolutif par l'échange d'information passé dans une mémoire globale, la matrice de trace de phéromone, qui est alimentée par chacun des membres de la communauté. Cette mémoire guide les fourmis dans la construction de nouvelles solutions

et celles-ci alimentent à nouveau la mémoire globale proportionnellement au degré de succès obtenu.

2.3.1 Les différentes versions d'algorithmes d'optimisation par colonie de fourmis

La métaheuristique d'optimisation par colonie de fourmis a été proposée initialement par Colorni, Dorigo *et al.* (1991) pour résoudre un problème de voyageur de commerce (VC). L'objectif du VC est de trouver le chemin le plus court reliant n villes données et chaque ville ne peut être visitée qu'une seule fois. C'est un problème de nature NP-difficile.

Il y a trois comportements des fourmis naturelles qui ont été transposées aux fourmis artificielles. Le premier est la préférence des chemins avec haut niveau de phéromone, le deuxième est le niveau élevé de phéromone sur le chemin le plus court et troisièmement, la trace de phéromone sert de moyen de communication entre les fourmis [Dorigo et Gambardella 1997].

L'algorithme d'optimisation par colonie de fourmis est présenté à la Figure 2.2 et prend en compte deux procédures de base qui sont la construction de m fourmis qui deviennent m solutions pour le problème considéré et la procédure d'actualisation des traces de phéromone où la quantité de phéromone déposée sur chaque arête du graphe est modifiée au fur et à mesure que les solutions sont construites.

La construction des solutions est faite de manière probabiliste et la probabilité d'ajout de nouveaux éléments à la solution en construction dépend de l'heuristique d'opportunité η et du dépôt des traces de phéromone τ laissé antérieurement (Bonabeau, Dorigo *et al.*

1999). La modification des traces de phéromone prend en considération le taux d'évaporation des phéromones ρ et la qualité des solutions produites.

/ * Initialisation */
POUR toutes les arêtes (i,j) **FAIRE**
  $\tau_{ij}(0) = \tau_0$
FIN POUR
POUR $k = 1$ jusqu'à m **FAIRE**
    Placer les k fourmis dans des sommets aléatoirement
FIN POUR
/* Cycle principal */
POUR $t = 1$ jusqu'à *tmax* **FAIRE**
    POUR $k = 1$ jusqu'à m **FAIRE**
      Construire une solution $S^k(t)$ en appliquant $n-1$ fois une règle probabiliste où le choix est une fonction de la trace de phéromone τ et d'une heuristique d'opportunité η.
    FIN POUR
    SI une meilleure solution est trouvée **ALORS**
      Actualiser la meilleure solution trouvée
    FIN SI
    POUR toutes les arêtes (i,j) **FAIRE**
      Actualiser les traces de phéromone en appliquant la règle d'actualisation des traces de phéromone
    FIN POUR
FIN POUR
Imprimer la meilleure solution

Figure 2.2 – Description de haut niveau d'un OCF générique [Bonabeau, Dorigo *et al.* 1999]

L'algorithme d'optimisation par colonie de fourmis a fait l'objet de nombreux travaux qui ont permis de définir différentes versions de l'algorithme. Dans l'AS [Dorigo, Maniezzo *et al.* 1991] et (Dorigo 1992), qui est la première version de l'OCF, les informations collectées pour chaque fourmi sont utilisées à la fin de chaque cycle pour actualiser les traces de phéromone. Ainsi, toutes les fourmis peuvent être guidées au prochain cycle par ces informations.

L'EAS [Dorigo, Maniezzo *et al.* 1991], (Dorigo 1992) et [Dorigo, Maniezzo *et al.* 1996] est la première amélioration de l'AS au niveau du dépôt de la phéromone. Ce dépôt est fait par les fourmis à la fin d'un cycle. Il permet d'avoir un renforcement des traces de phéromone pour les parties de solution appartenant seulement à la meilleure solution trouvée jusqu'à maintenant.

La deuxième amélioration de l'AS est l'ASrank [Bullnheimer, Hartl *et al.* 1999]. Cet algorithme permet l'actualisation des phéromones aux $(W-1)$ fourmis qui ont obtenu les meilleurs résultats, c'est-à-dire que l'actualisation est faite en fonction de la performance et que la quantité de phéromone déposée dépend de sa classification à la fin d'un cycle.

Pour sa part, dans le MMAS (Stützle et Hoos 2000), l'actualisation des phéromones dans la matrice est faite seulement par la fourmi qui a obtenu le meilleur résultat dans le cycle courant. De plus, la trace accumulée dans la matrice de phéromone est limitée par τ_{min} et τ_{max}, qui sont les quantités minimales et maximales de phéromone qui peuvent être déposées.

La version la plus récente de l'OCF est l'ACS [Dorigo et Gambardella 1997]. L'actualisation des phéromones est faite par la fourmi qui a trouvé le meilleur chemin au cours du cycle courant. On note également une mise à jour locale de la trace de phéromone après chacun des choix effectués lors de la construction d'une solution. Également, une liste de candidats est incluse dans l'algorithme lors de l'application de la règle de transition.

/ * Initialisation */
POUR toutes les arêtes (i,j) **FAIRE**
$\tau_{ij}(0) = \tau_0$
FIN POUR
POUR $k = 1$ jusqu'à m **FAIRE**
Placer les k fourmis dans des sommets aléatoirement
FIN POUR
LAISSER T^+ être le chemin le plus court trouvé depuis le début et L^+ sa distance
/* Cycle principal */
POUR $t = 1$ jusqu'à t_{max} **FAIRE**
POUR $k = 1$ jusqu'à m **FAIRE**
Construire une tournée $T^k(t)$ en appliquant $n - 1$ fois l'étape suivante :
S'il existe au moins une ville $j \in$ liste candidate **ALORS**
Choisir la prochaine ville $j, j \in J_i^k$, parmi les cl villes de la liste candidats comme présenté ci-dessous

$$j = \begin{cases} \arg max_{u \in J_i^k} \left\{ [\tau_{ij}(t)] \cdot [\eta_{ij}]^\beta \right\} & \text{If } q \leq q_0; \\ J & \text{If } q > q_0; \end{cases} \qquad [2.1]$$

Où J $\in$ J_i^k est choisie en fonction de la probabilité :

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)] \cdot [\eta_{ij}]^\beta}{\sum_{l \in J_i^k} [\tau_{il}(t)] \cdot [\eta_{il}]^\beta} \qquad [2.2]$$

et où i est la ville actuelle
SINON
Choisir le $j \in j_i^k$ correspondant à la ville la plus proche
FIN SI
Après chaque transition la fourmi k applique la règle d'actualisation locale

$$\tau_{ij}(t) \leftarrow (1 - \rho) \cdot \tau_{ij}(t) + \rho \cdot \tau_0 \qquad [2.3]$$

FIN POUR
POUR $k = 1$ jusqu'à m **FAIRE**
Calculer la distance L^K(t) de la tournée $T^k(t)$ produite par la fourmi k
FIN POUR
SI un meilleur chemin est trouvé **ALORS**
Actualiser T^+ et L^+
FIN SI
POUR toutes les arêtes $(i,j) \in T^+$ **FAIRE**
Actualiser les traces de phéromone en appliquant la règle
$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij}(t) + \rho \cdot \Delta\tau_{ij}(t)$ où $\Delta\tau_{ij}(t) = {}^1/_{L^+}$ [2.4]
FIN POUR
POUR toutes les arêtes (i,j) **FAIRE**
$\tau_{ij}(t+1) = \tau_{ij}(t)$
FIN POUR
FIN POUR
Imprimer la meilleure solution

Figure 2.3 – Description de haut niveau d'un ACS (adapté de Bonabeau, Dorigo *et al.* [1999])

La Figure 2.3, présentée ci-dessus, décrit la première version de l'ACS qui a été conçue pour résoudre les problèmes de grande taille, notamment le problème du voyageur de commerce, qui est un problème classique dans ce domaine d'étude. Cette première version est considérée avec le MMAS, comme l'une des plus performantes (Dorigo et Stützle 2004) et sert de base aux travaux présentés dans ce mémoire.

L'ACS introduit une règle de transition qui permet une exploration explicite. Cette règle dépend du paramètre q_0 qui permet de balancer l'intensification et la diversification. Une fourmi k sur une ville i choisit une ville j à partir de l'Équation 2.1 dans laquelle q est une variable aléatoire distribuée de façon uniforme sur [0,1], q_0 un paramètre réglable $(0 \leq q_0 \leq 1)$ et $J \in J_i^k$ est une ville sélectionnée aléatoirement en fonction de la probabilité définie par l'Équation 2.2.

En fonction de la valeur de q_0, il y a deux comportements possibles : si $q > q_0$, le choix se fait de la même façon que l'algorithme AS et tend à effectuer une diversification; si $q < q_0$, le système tend vers l'intensification, c'est-à-dire que l'algorithme exploite davantage l'information récoltée par le système.

En ce qui concerne la gestion des traces de phéromone, elle est faite à deux niveaux qui sont la mise à jour locale et la mise à jour globale. La mise à jour globale des traces est faite seulement sur les arêtes qui font partie du meilleur chemin trouvé depuis le début de la recherche selon l'Équation 2.3 où (i,j) sont les arêtes qui font partie de T^+, ρ un paramètre qui contrôle l'évaporation de phéromone et L^+ est la distance de la tournée T^+

Les actualisations locales sont effectuées au moment où une fourmi k se déplace d'une ville i à une ville $j \in J_i^k$ sélectionnée et la concentration de phéromone de (i,j) est

actualisée selon l'Équation 2.4. La valeur de τ_0 est la même que la valeur initiale des traces de phéromone. La règle de mise à jour locale permet d'évaporer la phéromone pour éviter de faire toujours le même choix dans la construction des solutions.

L'ACS exploite une liste candidate qui contient un nombre limité de villes pour réaliser le choix de la prochaine ville à visiter. La liste de candidats contient les cl villes qui sont les villes les plus proches. Avant de choisir la prochaine ville, les fourmis vérifient la liste des villes candidates pour voir quelles sont celles qui n'ont pas encore été traitées.

2.3.2 L'ACS pour le problème de machine unique avec temps de réglage dépendants de la séquence pour minimiser le retard total

L'ACS de base a été conçu pour résoudre le problème de VC. Toutefois, deux auteurs ont proposé des versions adaptées au problème à l'étude.

Gagné, Price *et al.* [2002] ont proposé une version de l'ACS qui prend spécifiquement en compte les temps de réglage entre les travaux et les dates d'échéance dans la règle de transition présentée à l'équation 2.5.

$$p_{ij}^{k}(t) = \frac{\left[\tau_{ij}(t)\right]^{\alpha} \cdot \left[\frac{1}{\overline{S}_{ij}}\right]^{\beta} \cdot \left[\frac{1}{\overline{m}_{ij}}\right]^{\delta} \cdot \left[\frac{1}{\overline{B}_{ij}}\right]^{\phi}}{\sum_{j \notin Tabu_k}\left[\tau_{ij}(t)\right]^{\alpha} \cdot \left[\frac{1}{\overline{S}_{ij}}\right]^{\beta} \cdot \left[\frac{1}{\overline{m}_{ij}}\right]^{\delta} \cdot \left[\frac{1}{\overline{B}_{ij}}\right]^{\phi}} \qquad [2.5]$$

Le premier type de visibilité est une matrice de réglages normalisée construite à partir des réglages associés à chaque instance de problème. Elle a une taille de $[n, n+1]$ car elle considère l'état initial de la machine. Les éléments de la matrice normalisée sont définis par $\overline{S}_{ij} = S_{ij}/Max\ S_{ij}$.

Le deuxième type de visibilité est une matrice de marge normalisée. La marge de la tâche j est le temps disponible avant sa date d'échéance pour qu'elle ne soit pas en retard si elle est traitée immédiatement après une tâche i. La marge est définie comme étant $m_{ij} = max\{d_j - p_j - S_{ij}, 0\}$ où d_j est la date due de la tâche j, p_j est le temps de traitement et S_{ij} le temps de réglage. Les éléments de la matrice normalisée sont définis par $\overline{m}_{ij} = m_{ij}/Max\ m_{ij}$.

La règle de transition intègre également une fonction de regard vers l'avant représentée par la matrice B_{ij}. Il s'agit d'une borne inférieure pour le retard total pour la solution partielle en assumant que la tâche j est la tâche candidate choisie après la tâche i. Cette borne est établie par $B_{ij} = T_{Q_h} + T_{Q'_h}$ où T_{Q_h} est la valeur du retard obtenue à partir de la solution courante Q_h et $T_{Q'_h}$ une estimation pour la partie restante Q'_h de la solution. La matrice normalisée est obtenue par $\overline{B}_{ij} = B_{ij}/Max\ B_{ij}$.

Les exposants α, β, γ et ϕ permettent de diriger la recherche en fonction de la nature du problème en privilégiant ainsi une composante au détriment de l'autre. Selon les auteurs, ces paramètres sont calibrés en fonction de la nature de chaque problème.

Après avoir terminé un cycle, la solution produite par la meilleure fourmi est améliorée par une méthode de recherche locale. Les auteurs ont utilisé deux méthodes différentes et le choix d'une méthode s'est fait de façon aléatoire. Les méthodes utilisées sont le 3-opt restreint qui permet de faire des échanges sans inverser complètement la séquence et le «Random Search Pairwise Interchange».

Liao et Juan (2007) ont aussi proposé un ACS mais celui-ci est plutôt conçu pour le problème de machine unique avec retard total pondéré. Les auteurs proposent ainsi

d'introduire le concept de valeur minimale pour les traces de phéromone qui est utilisé par le MMAS.

L'initialisation de la matrice de phéromone se fait en utilisant le paramètre τ_0 qui est défini comme $\tau_0 = K/nWT_{ATCS}$ où K est un paramètre, n le nombre de tâches disponibles et nWT_{ATCS} représente l'application de la règle «Apparent Tardiness Cost with Setup» (ATCS) pour le retard pondéré.

La règle de transition, présentée à l'équation 2.6 prend en compte deux matrices, la matrice de phéromone et une matrice η construite à partir de l'application de l'ATCS.

$$p(i,j) = \frac{[\tau_t(i,j)] \cdot [\eta(i,j)]^{\beta}}{\sum_{u \in U}[\tau_t(i,u)] \cdot [\eta(i,u)]^{\beta}} \qquad [2.6]$$

Le paramètre $\eta(i,j)$ est une heuristique d'opportunité qui place la tâche à la position i et β est un exposant qui détermine l'importance de l'information produite par l'ATCS.

La méthode de recherche locale est une combinaison de la permutation (IT) et de l'insertion dans le voisinage (IS). La permutation consiste à déplacer la tâche à la position i avec celle de la position j. L'insertion dans le voisinage consiste à placer la tâche de la position i à la position j. Les variantes possibles pour la combinaison des deux techniques sont IT/IS et IS/IT. Le choix d'une des techniques est fait de façon aléatoire et la recherche locale est utilisée à chaque fois qu'une meilleure solution est trouvée.

La mise à jour globale de la trace de phéromone, présentée à l'Équation 2.7, est faite par la meilleure fourmi à la fin d'un cycle.

$$\tau_{t+1}(i,j) = (1-\alpha)\cdot\tau_t(i,j) + \alpha\Delta\tau_t(i,j) \qquad [2.7]$$

Le paramètre α varie entre zéro et un et représente l'évaporation de phéromone. La quantité d'évaporation est définie par $\Delta\tau_t(i,j) = 1 / WT^*$, où WT^* est le retard pondéré de la meilleure solution. Pour éviter le piège des optima locaux qui peut être causé au cas où l'évaporation atteint la valeur zéro, les auteurs ont introduit une borne inférieure pour les traces de phéromone, définie par $\tau_t(i,j) = (1 / 5)\tau_0$.

Dans le cadre de ce mémoire, les travaux sont basés principalement sur l'ACS proposé par Gagné, Price *et al.* [2002]. La fonction de regard vers l'avant (look-ahead) n'est toutefois pas utilisée.

2.4 Objectifs de la recherche

Les problèmes d'ordonnancement industriel sont des problèmes de nature NP – difficile, qui ne peuvent être résolus en temps polynomial. Plusieurs métaheuristiques ont été proposées dans le but de trouver le meilleur compromis entre la qualité de solution et les temps de calcul.

L'objectif général de ce mémoire est de contribuer à l'amélioration des méthodes de résolution pour le problème d'ordonnancement à machine unique avec temps de réglage dépendants de la séquence pour minimiser le retard total afin de favoriser l'avancement des connaissances dans le domaine.

À ce jour, deux propositions d'ACS ont été présentées dans la littérature pour résoudre le problème $1 \mid s_{ij} \mid \sum T_j$. Leur mode de construction s'inspire de la version classique de l'ACS en plaçant les tâches les unes après les autres.

Le premier objectif spécifique consiste à exploiter un principe de construction à rebours dans la règle de transition. En effet, nous envisageons de modifier le comportement de l'ACS au niveau de la construction des solutions pour privilégier une construction à rebours de la séquence. Ce principe nous parait adapté aux caractéristiques du problème car le placement des tâches en retard à la fin de la séquence en premier lieu facilitera le placement des tâches restantes. La construction à rebours est un apport novateur qui n'a jamais été exploité dans la littérature pour résoudre les problèmes d'ordonnancement.

Le deuxième objectif spécifique est d'évaluer la performance de cet ACS par rapport à l'ACS de Gagné, Price *et al.* [2002] sur les instances tests proposées par Ragatz (1989) et Gagné, Price *et al.* [2002].

La démarche utilisée dans ce travail consiste, tout d'abord, à reproduire l'ACS de Gagné, Price *et al.* [2002] en ignorant la fonction de regard vers l'avant et la recherche locale. La fonction de regard vers l'avant est ignorée parce que sa principale fonction est de prévoir les tâches qui doivent être placées par la suite dans une méthode de construction traditionnelle, condition qui ne pourra pas être appliquée dans une construction à rebours. La recherche locale est aussi ignorée car on veut mettre en évidence le potentiel d'une construction à rebours. Ainsi, cette reproduction de l'ACS de Gagné, Price *et al.* [2002] permettra d'avoir une base de comparaison équitable pour l'ACS à rebours.

CHAPITRE 3

Un nouveau mode de construction de solutions avec l'ACS pour le problème de machine unique avec temps de réglage dépendants de la séquence

3.1 Introduction

L'ACS a été proposé dans la littérature par Dorigo et Gambardella [1997] et plusieurs travaux ont porté sur différents problèmes d'optimisation combinatoire dont le problème d'ordonnancement à machine unique avec temps de réglage dépendants de la séquence. Toutefois, aucun de ces travaux ne s'est penché sur un mode de construction à rebours d'une solution pour résoudre un problème. Dans ce chapitre, un mode de construction à rebours d'une solution dans un ACS est proposé ayant comme objectif la minimisation du retard total. Nous estimons que cet objectif se prête très bien à une telle proposition d'algorithme car une construction classique favorise le respect des dates dues pour les tâches placées en début de séquence et les retards sont généralement obtenus pour les tâches restant à placer en fin de séquence.

3.2 L'ACS à rebours (ACS – R)

Compte tenu de l'objectif à optimiser et de la nature du problème étudié, plusieurs éléments de l'ACS doivent être revus pour une construction de solution commençant par la fin de la séquence. Les sections suivantes décrivent ces éléments.

3.2.1 Estimation de la date de fin

L'ACS classique construit les solutions en plaçant les tâches les unes après les autres en fonction des visibilités et de la trace accumulée. La date de fin de la dernière tâche détermine alors le «makespan» de l'ordonnancement et le calcul du retard total est obtenu en cumulant le retard de chacune des tâches.

Dans l'ACS – R, la construction d'une solution s'opère en plaçant les tâches en débutant par la fin de la séquence. Il faut, à priori, connaître la date de fin de la dernière tâche ou estimer celle-ci avant d'amorcer la construction d'une solution. Nous avons choisi

d'estimer la date de fin d'un ordonnancement en construisant deux séquences. La première séquence est établie en utilisant la règle de priorité favorisant la plus petite date d'échéance (EDD) alors que la deuxième séquence est générée de façon entièrement aléatoire. Ces deux séquences permettent d'estimer un intervalle dans lequel la date réelle de fin de l'ordonnancement peut potentiellement se retrouver. Pour la construction de chaque fourmi, on tire la date de fin aléatoirement dans cet intervalle.

À chaque fois qu'une tâche est placée à l'aide de la règle de transition de l'ACS – R, la mise à jour de la date de fin est réalisée en soustrayant le temps de traitement de la tâche placée à la date de fin actuelle. On a donc $date\ de\ fin = date\ de\ fin - p_j$ où p_j est le temps de traitement de la tâche déjà placée. Le temps de réglages entre les tâches n'a pas été pris en compte dans la mise à jour de la date de fin parce que l'ordre de grandeur des valeurs associé aux réglages n'est pas significatif par rapport à la date de fin et au temps de traitement.

3.2.2 Définition d'un nouveau concept de visibilité

Pour le problème d'ordonnancement à machine unique avec temps de réglage dépendants de la séquence, l'ACS proposé par Gagné, Price *et al.* [2002] prend en compte, à titre de visibilité, le temps de réglage normalisé entre les tâches, la marge normalisée d'une tâche et la fonction de regard vers l'avant.

Pour reproduire cet ACS aux fins de comparaison, nous n'avons pas inclus la fonction de regard vers l'avant. La règle de transition utilisée, présentée à l'équation 3.1, introduit le concept de marge tel que proposé par les auteurs et qui doit être redéfini dans une implémentation à rebours de l'ACS. Rappelons que la marge d'une tâche est le délai d'une tâche par rapport à sa date d'échéance avant qu'elle ne soit en retard. Cette marge est

calculée au début de l'algorithme pour chacune des tâches sans être mise à jour à chaque pas de la construction d'une fourmi.

$$p_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{\alpha} \cdot \left[\frac{1}{\bar{S}_{ij}}\right]^{\beta} \cdot \left[\frac{1}{\bar{m}_{ij}}\right]^{\delta}}{\sum_{j \notin Tabu_k} [\tau_{ij}(t)]^{\alpha} \cdot \left[\frac{1}{\bar{S}_{ij}}\right]^{\beta} \cdot \left[\frac{1}{\bar{m}_{ij}}\right]^{\delta}} \qquad [3.1]$$

Inspiré de la règle de transition présentée à l'Équation 3.1, le concept de marge arrière (MA_i) a été développé de façon à privilégier les tâches à placer à la fin de la séquence. La marge arrière est le délai disponible pour placer une commande avant que celle-ci devienne en retard.

La marge arrière est ainsi définie comme étant $MA_j = \mathrm{d_j} - \text{date de fin} + \mathrm{s_{ji}}$, où d_j correspond à la date due de la tâche à placer, $date\ de\ fin$ à la date de fin de la dernière tâche placée et s_{ji} au temps de réglage nécessaire lorsque la tâche j est placée avant la tâche i déjà placée.

Le résultat du calcul du *MA* peut être positif, négatif ou nul. Un résultat positif exprime le fait que si la tâche est insérée dans la séquence, elle ne génère aucun retard. Si on prend l'exemple de la Figure 3.1, la tâche 15 est déjà placée dans la séquence et on calcule la marge de la tâche 8 pour l'insérer dans la séquence.

On peut constater que si la tâche 8 est placée avant la tâche 15, elle ne sera pas en retard car sa date de fin sera à 1509 compte tenu du réglage 5 entre les tâches 8 et 15 et de sa date d'échéance de 1580. On a donc une *MA* de 71 pour la tâche 8. Si toutes les tâches non placées ont des marges arrières positives, on va privilégier celles qui ont la

plus grande marge positive parce qu'elles seront placées le plus éloigné possible du début de la séquence sans qu'elles ne soient en retard.

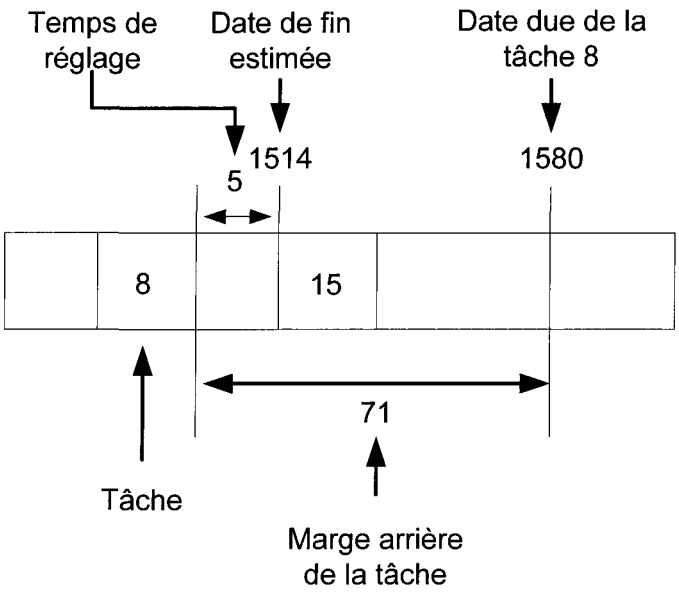

Figure 3.1 – Marge arrière positive

Si le résultat du calcul de la *MA* est négatif, la tâche à placer est déjà en retard par rapport à sa date due. Si on reprend l'exemple précédent avec une date due inférieure à la date de fin estimée, on peut constater que si la tâche 8 est placée immédiatement avant la tâche 15, elle sera livrée en retard. Ce cas est illustré à la Figure 3.2 où la *MA* de la tâche 8 est alors de -208. Il est donc intéressant de retarder l'insertion de cette tâche dans la séquence pour diminuer le retard total. Si les *MA* sont négatives pour toutes les tâches non placées, l'objectif est de privilégier les tâches qui ont la plus petite marge arrière négative pour essayer de minimiser au maximum le retard engendré par ces tâches.

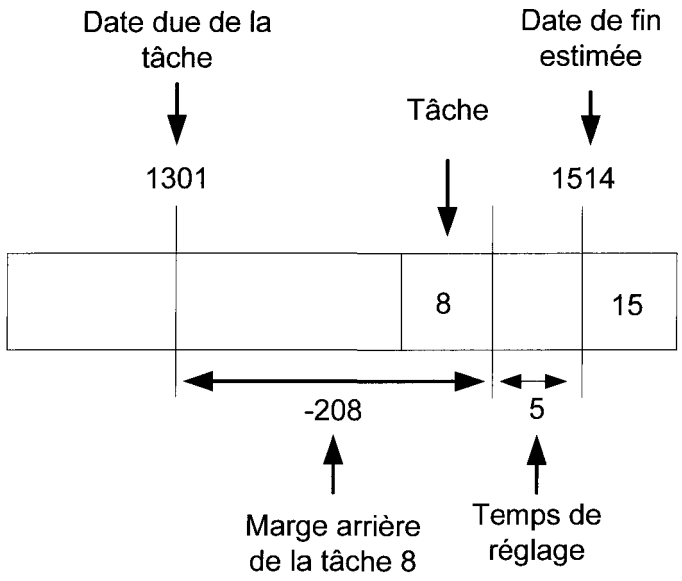

Figure 3.2 – Marge arrière négative

Le calcul de la marge arrière peut être positif pour certaines tâches, négatif pour d'autres et même nul. Supposons un exemple où la date de fin estimée est de 1305, la dernière tâche placée est la tâche 12 et la liste des tâches non placées, leur date d'échéance et les temps de réglage entre chacune des tâches et la tâche 12 sont présentés dans le Tableau 3.1.

Tâche →	6	7	8	9	10	11	14
Date Due →	1236	1240	1301	1323	1353	1368	1432
Temps de réglage →	19	0	16	4	5	14	2

Tableau 3.1 – Informations sur les tâches non placées

En appliquant le calcul de la marge arrière pour toutes les tâches non placées, on constate que la marge arrière est positive pour certaines tâches et négative pour d'autres, tel que présenté dans le Tableau 3.2.

Tâche →	6	7	8	9	10	11	14
MA →	-50	-65	12	22	53	77	129

Tableau 3.2 – Résultat du calcul de la marge arrière

Les tâches à privilégier sont celles avec la *MA* la plus grande. Dans l'Équation 2.8, une *MA* normalisée est utilisée dans la nouvelle règle de transition. Alors, si une ou plusieurs valeurs de la *MA* sont négatives, les résultats sont décalés en ajoutant la valeur absolue du plus petit nombre négatif au résultat *MA* de toutes les tâches candidates, ce qui ramène le plus petit nombre négatif à zéro, tel que présenté dans le Tableau 3.3.

MA décalé →	15	0	77	87	118	142	194

Tableau 3.3 – *MA* décalée

Comme toutes les valeurs utilisées par la règle de transition doivent être ramenées au même ordre de grandeur pour que le choix puisse être équitable, les valeurs décalées sont alors normalisées dans l'intervalle entre un et deux. Les valeurs non normalisées des *MA* décalées sont présentées dans le Tableau 3.4. Pour ce cas particulier, les tâches qui ont la plus grande valeur positive ont plus de chances d'être choisies par la règle de transition.

MA normalisé →	1.07	1	1.39	1.43	1.6	1.73	2

Tableau 3.4 – *MA* normalisée

Contrairement à l'ACS de Gagné, Price *et al.* [2002], la *MA* normalisée est recalculée à chaque pas de la construction d'une séquence en considérant la nouvelle date de fin.

3.2.3 La liste de candidats

La liste de candidats est un concept introduit dans les OCF pour réduire le temps de calcul en limitant les choix pour la prochaine tâche à placer. La liste de candidats proposée par Gagné, Price *et al.* [2002] dans l'ACS classique est constituée des tâches ayant la plus petite marge. Cette liste est triée par ordre croissant et sa taille est définie comme étant 30% du nombre des tâches du problème traité.

Pour le cas de l'ACS – R, la liste de candidats a été définie en fonction des caractéristiques du problème à traiter. Pour les instances de problèmes avec un intervalle étroit des dates d'échéance, la liste de candidats est établie en considérant la marge de chaque tâche. On a $m_j = d_j - p_j$ où d_j est la date d'échéance de la tâche candidate et p_j son temps de traitement. Comme l'intervalle des dates d'échéance est petit, il faut favoriser les tâches qui ont la plus petite marge. L'objectif est de placer les tâches qui ont les grands temps de traitement à la fin de la séquence. Pour les instances de problème avec un intervalle des dates d'échéance large, on prend seulement en compte leur date d'échéance (d_j) et on favorise ainsi le placement des tâches qui doivent terminer le plus tôt. Dans les deux cas, la liste des candidats est triée par ordre décroissant et la taille de la liste utilisée est la même que celle proposée par Gagné, Price *et al.* [2002].

3.2.4 La règle de transition

Compte tenu des éléments présentés aux sections 3.2.1 à 3.2.3, la règle de transition de l'ACS à rebours est définie par à l'équation 3.2.

$$p_{ji}^{k}(t) = \frac{\left[\tau_{ji}(t)\right]^{\alpha} \cdot \left[\frac{1}{\bar{S}ji}\right]^{\beta} \cdot {MA_j}^{\delta}}{\sum_{i \notin Tabu_k}\left[\tau_{ji}(t)\right]^{\alpha} \cdot \left[\frac{1}{\bar{S}ji}\right]^{\beta} \cdot {MA_j}^{\delta}} \qquad [3.2]$$

La première matrice de visibilité représente le temps de réglage normalisé entre les tâches, tel que proposé par Gagné, Price *et al.* [2002] et la deuxième matrice est la marge arrière introduite à la section 3.2.2.

La matrice des temps de réglage normalisé a une dimension de $[n+2, n]$ car on ajoute, non seulement les réglages avec la dernière tâche traitée à la période précédente, mais aussi une tâche associée à la transition vers la période de planification suivante. Dans ce cas, on assure un temps de réglage nul. Les éléments de cette matrice sont normalisés de la manière suivante : $\bar{S}_{ij} = S_{ij}/MAX\ S_{ij}$.

Les exposants α, β et δ associés à chaque matrice de la règle de transition permettent de privilégier certains éléments en fonction des caractéristiques du problème.

La partie déterministe de la règle de transition est présentée à l'équation 3.3 et retient la tâche qui retourne la plus grande valeur.

$$arg\,max_{u \in i_i^k}\left\{\left[\tau_{ji}(t)\right]^{\alpha} \cdot \left[\frac{1}{\bar{S}ji}\right]^{\beta} \cdot {MA_j}^{\delta}\right\} \qquad [3.3]$$

/ * Initialisation de plusieurs paramètres* /

SQ est la meilleure séquence construite à partir de la règle EDD, RT le retard de la meilleure solution et Bi la borne supérieure où $Bi = Cmax(SQ)$;

/ * Initialisation des traces de phéromone*/

POUR chaque paire de tâches (*i,j*) **FAIRE**

Attribuer une valeur initiale $\tau_{ij}(0) = \tau_0$;

FIN POUR

POUR chaque itération $t = 1$ à $tmax$ **FAIRE**

Construire une solution aléatoire Al et calculer la borne supérieure $Bs = Cmax(Al)$;

$date\ de\ fin = aleatoire(Bi, Bs)$;

POUR chaque tâche $j = n$ à 1 par pas de -1 **FAIRE**

POUR chaque fourmi $k = 1$ à m **FAIRE**

POUR tous les candidats cn, $cn \notin Tabou_k$ **FAIRE**

Calculer la MA_{cn},

$SA_{cn} = d_i - date\ de\ fin + s_{ji}$;

FIN POUR

Choisir la prochaine tâche j, $j \in cn$;

$$j = \begin{cases} \arg\max_{u \in i_i^k} \left\{ [\tau_{ji}(t)]^{\alpha} \cdot \left[\frac{1}{\bar{s}_{ji}}\right]^{\beta} \cdot MA_j{}^{\delta} \right\} & \text{Si } q \le q_0; \\ J & \text{Si } q > q_0; \end{cases}$$

où J est choisi en fonction de la probabilité .

$$p_{ji}^k(t) = \frac{[\tau_{ji}(t)]^{\alpha} \cdot \left[\frac{1}{\bar{s}_{ji}}\right]^{\beta} \cdot MA_j{}^{\delta}}{\sum_{j \notin Tabu_k} [\tau_{ji}(t)]^{\alpha} \cdot \left[\frac{1}{\bar{s}_{ji}}\right]^{\beta} \cdot MA_j{}^{\delta}}$$

Garder l'information en $Tabou_k$;

Faire la mise à jour locale des traces de phéromone pour les paires de tâches (j, i);

FIN POUR

FIN POUR

POUR $k = 1$ à m **FAIRE**

Calculer le retard total $RT_k(t)$ pour la séquence $SQ_k(t)$ produite par la fourmi k;

SI $RT_k(t) < RT$ faire la mise à jour de RT et SQ

FIN POUR

Faire la mise à jour globale des traces de phéromone;

Vider $Tabou_k$;

FIN POUR

Figure 3.3 – Pseudo-code de l'ACS – R

Le paramètre q définit la probabilité d'utiliser la partie déterministe ou probabiliste de la règle de transition. Comme proposé par Gagné, Price *et al.* [2002], la valeur de $q_0 = 0.9$ est utilisée et permet à la partie déterministe de la règle de transition d'avoir une probabilité de 90% d'être choisie.

Le lecteur peut consulter le pseudo-code de l'ACS – R à la Figure 3.3 qui intègre l'ensemble des éléments présentés précédemment.

3.3 Modifications proposées à l'ACS – R

Les sous-sections qui suivent présentent deux modifications proposées pour améliorer le fonctionnement de l'ACS - R. D'une part, l'ACS – R utilise un calibrage des paramètres de la règle de transition selon les caractéristiques de l'instance de problème traité. Nous visons à uniformiser le choix des paramètres quel que soit le problème traité. D'autre part, nous cherchons à accélérer la construction de la solution dans le cas où les dates d'échéance n'ont plus d'influence directe dans le choix des tâches.

3.3.1 ACS – R avec changements dynamiques des paramètres (ACS – RD)

Dans l'ACS – RD, l'exposant δ associé à la marge normalisée dans la règle de transition est initialement fixé à une valeur plus élevée que l'exposant β associé au réglage normalisé. Nous visons ainsi à privilégier davantage le respect des dates d'échéance pour les premières tâches placées dans la séquence. En cours de construction d'une séquence, il est possible que toutes les *MA* des tâches non placées soient positives. Toutes ces tâches ne peuvent, par conséquent, être en retard quelle que soit leur position dans la séquence à compléter. La *MA* n'a alors plus d'influence directe sur les prochains choix à effectuer par la règle de transition et il est préférable d'axer les prochains choix de façon à minimiser les réglages entre les tâches. À partir de ce moment,

l'exposant associé au réglage normalisé prendra une valeur plus importante et on réduira la valeur de l'exposant associé à la marge normalisée pour la construction du reste de la séquence.

3.3.2 ACS – R avec règle de priorité (ACS- RP)

En cours de construction d'une séquence, lorsque toutes les *MA* des tâches non placées deviennent positives, une règle de priorité sera utilisée pour compléter la séquence à la place de la règle de transition de l'ACS-R. La règle de priorité «cheap insertion» [Rosenkrantz, Stearns *et al.* 1974] est utilisée à cette fin dans l'objectif de minimiser le temps de réglage entre les tâches pour la partie de la séquence à compléter. Cette règle consiste à choisir aléatoirement une tâche à placer et à vérifier toutes les positions d'insertion possibles de façon à minimiser le temps total de réglage de la sous-séquence. Pour appliquer cette règle, on a besoin non seulement de la dernière tâche placée dans la séquence partielle, mais aussi de la dernière tâche traitée à la période précédente.

Pour démontrer le fonctionnement de cette règle, on tire aléatoirement une tâche dans la liste des tâches non placées. La tâche tirée aléatoirement (tâche 3) est placée entre la dernière tâche traitée à la période précédente, représentée par zéro, et la dernière tâche placée dans la séquence en construction (tâche 11), tel qu'illustré à la Figure 3.4. Le réglage total de la sous-séquence est égal à 32 et correspond au réglage nécessaire entre les tâches 0 et 3 ainsi qu'entre les tâches 3 et 11.

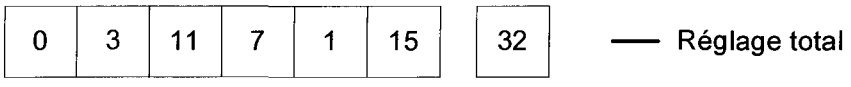

Figure 3.4 – Première insertion dans la sous-séquence avec la règle de priorité

Ensuite, on tire aléatoirement une autre tâche (tâche 8) et on évalue de nouveau le résultat de la somme des temps de réglage selon les deux possibilités présentées à la Figure 3.5. On retient alors la sous-séquence avec le plus petit temps de réglage total.

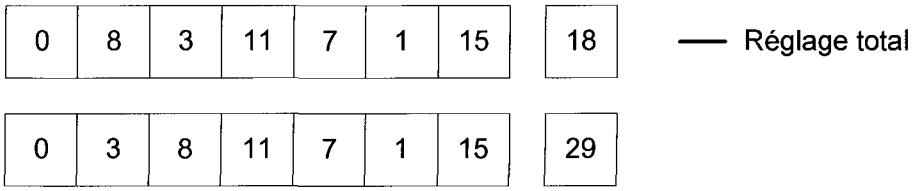

Figure 3.5 – Deuxième insertion dans la sous-séquence avec la règle de priorité

Cette séquence sert de base pour continuer la construction de la séquence dans l'objectif de minimiser les temps de réglage, tel que présenté à la Figure 3.6 où la tâche 5 est choisie.

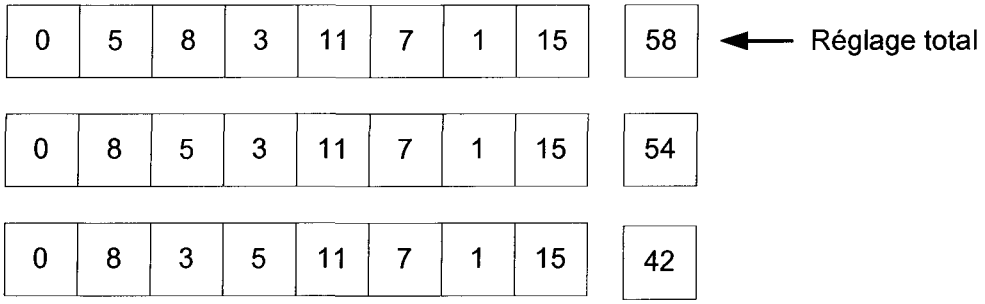

Figure 3.6 – Troisième insertion dans la sous-séquence avec la règle de priorité

La procédure se poursuit jusqu'à ce que toutes les tâches non placées soient placées de façon à compléter la séquence. Cette fourmi est ensuite évaluée sur l'objectif du retard total.

3.4 Conclusion

Dans ce chapitre, nous avons proposé trois versions de l'ACS avec une construction à rebours. Premièrement, nous avons présenté la version ACS – R, qui a été mise en

œuvre à partir de la version ACS de Gagné, Price *et al.* (2002). Des modifications ont été apportées avec la proposition d'un nouveau concept de visibilité permettant ainsi la construction des solutions à rebours de la séquence. Cette nouvelle visibilité a été nommée marge arrière (*MA*) et a été développée de façon à privilégier les tâches à placer à la fin de la séquence afin de limiter leurs retards. La deuxième proposition a été l'ACS–RD qui permet un paramétrage dynamique des exposants associés à chaque matrice de visibilité dans la règle de transition. Le but de cet algorithme est de privilégier les réglages dès que toutes les tâches non placées ne génèrent plus de retard. L'ACS – RP est la troisième proposition d'algorithme qui utilise la règle de priorité «cheap insertion» pour minimiser le «makespan» à partir du moment où les tâches à placer ne sont pas en retard. Dans le prochain chapitre, nous évaluons la performance de ces 3 algorithmes sur les problèmes tests de la littérature.

CHAPITRE 4

Expérimentations numériques et comparaison du mode de construction à rebours vis-à-vis l'ACS

4.1 Introduction

Afin d'évaluer la performance des algorithmes présentés dans le chapitre précédent, des essais numériques ont été effectués sur les 32 problèmes proposés par Ragatz (1993) et sur les 32 problèmes proposés par Gagné, Price *et al.* [2002]. Ceux-ci ont été décrits à la section 2.2 et sont disponibles sur le web à l'adresse https://www.msu.ed/~rubin/data/c&ordata.zip et à l'adresse http://wwwdim.uqac.ca/~c3gagne/DocumentRech/ProblemDataSet55to85.zip *respectivement*.

Tous les essais numériques ont été réalisés sur la base de 50 000 évaluations de solution et chaque instance a été exécutée 20 fois. Ils ont été exécutés sur une machine XEONTM avec 2,4 GHz de processeur et 512MB de RAM. Les résultats sont présentés en % de déviation par rapport aux meilleures solutions connues dans la littérature. Ces solutions ont été produites par l'algorithme de séparation et d'évaluation progressive proposé par Bigras, Gamache et al. (2008).

Dans ce chapitre, une analyse comparative des résultats produits par l'ACS et l'ACS – R est d'abord présentée. Par la suite, nous évaluons la performance des deux variantes de l'ACS – R, soit l'ACS –RD et l'ACS – RP par rapport à l'ACS.

4.2 Analyse comparative entre l'ACS classique et l'ACS – R

La première analyse consiste à évaluer la performance de l'ACS – R par rapport à l'ACS sur les 32 problèmes tests de Ragatz (1993) et les 32 problèmes tests de Gagné, Price *et al.* [2002].

4.2.1. Comparaison de la performance pour les problèmes de petite taille

Le Tableau 4.1 présente les résultats obtenus par les deux ACS. La première colonne du tableau indique le numéro de l'instance, les colonnes 2 à 5 présentent la déviation à l'optimal obtenue par l'ACS pour la meilleure solution obtenue (MS), la médiane des 20 exécutions pour chaque instance de problèmes (SM), la pire solution obtenue (PS) et le temps de traitement pour chaque instance traitée (TT) tandis que les colonnes 6 à 9 présentent une information similaire pour l'ACS – R.

En ce qui concerne les meilleures solutions (MS) obtenues par les 2 algorithmes, l'ACS trouve la meilleure solution pour 19 des 32 instances, c'est-à-dire pour plus de la moitié des instances de problèmes évaluées. L'écart minimal à l'optimal pour ces 19 instances est de 1,92% et l'écart maximal est de 20,58%. Les deux algorithmes atteignent l'optimum pour les mêmes 11 instances de problème et l'ACS - R est meilleur pour seulement deux instances de problèmes. On note toutefois une amélioration très importante pour les instances 601 et 605.

Si on compare la médiane des résultats obtenus (SM), on peut constater, à partir d'une analyse empirique, que l'ACS classique est meilleur que l'ACS – R pour 18 instances de problèmes. Les deux algorithmes atteignent l'optimum pour les mêmes 11 instances de problèmes et l'ACS – R est meilleur pour trois instances de problèmes. Encore une fois, on note une amélioration importante pour les instances 601 et 605.

PROB	ACS classique				ACS à rebours			
	MS	SM	PS	TT	MS	SM	PS	TT
401	0	0	0	6,73	6,67	15,56	22,22	22,4
402	0	0	0	0	0	0	0	0
403	0,23	0,31	2,02	6,75	2,95	5,56	7,99	22,46
404	3,66	6,84	8,34	6,74	6,75	8,9	18,46	22,43
405	0	0	0	0	0	0	0	0
406	0	0	0	0	0	0	0	0
407	1,4	12,28	21,49	6,75	21,98	28,1	31,54	22,45
408	2,51	3,41	4,06	6,74	6,11	7,89	8,94	22,65
501	4,21	5,56	8,43	14,15	6,13	7,28	9,58	34,97
502	0	0	0	0	0	0	0	0
503	0,77	2,8	3,8	14,01	3,92	4,49	5,43	35,34
504	0	0	0	0	0	0	0	0
505	0	0	0	14,18	0	0	0	35,51
506	0	0	0	0	0	0	0	0
507	9,41	13,19	14,41	14,14	21,33	23,65	26,19	35,4
508	27,73	39,09	50,23	14,18	37,28	45,09	50,81	35,46
601	375	487,5	541,67	23,83	183,33	225	266,67	51,72
602	0	0	0	0	0	0	0	0
603	1,77	3,03	3,92	23,98	4,93	5,46	6,11	51,82
604	2,44	2,94	3,69	23,85	8,33	9,81	10,64	52,55
605	66,23	83,33	101,32	23,58	39,91	68,64	83,77	51,77
606	0	0	0	0	0	0	0	0
607	11,55	14,11	15,91	23,86	20,26	21,89	23,46	52,17
608	18,62	23,58	29,52	23,85	23,63	28,07	32,86	52,15
701	41,24	80,93	91,75	38,11	51,55	123,2	194,85	75,07
702	0	0	0	0	0	0	0	0
703	2,59	3,3	3,75	37,98	4,86	5,84	6,69	75,36
704	6,27	8,31	10,65	38,42	8,84	11,97	13,84	75,75
705	53,5	72	87	38,21	59	66	68	74,89
706	0	0	0	0	0	0	0	0
707	8,52	11,73	13,39	38,23	19,59	23,23	24,6	75,21
708	13,5	15,34	16,36	38,52	23,2	24,61	26,64	75,98

Tableau 4.1 – Les déviations, en % par rapport à l'optimum, de l'ACS et de l'ACS – R pour les problèmes de petite taille

Pour les pires solutions produites (PS), l'analyse empirique des résultats permet d'affirmer que l'ACS classique est meilleur pour 18 instances de problèmes. Les deux algorithmes atteignent l'optimum pour les mêmes 11 instances de problèmes et l'ACS – R est meilleur pour trois instances de problèmes. Une amélioration significative est encore obtenue pour les instances 601 et 605.

Pour poursuivre cette analyse, un test de dominance stochastique a été réalisé. Ce test consiste à définir des intervalles compris entre la déviation maximale et minimale pour ensuite classer les résultats des ACS à partir du nombre d'instances qui appartiennent à ces intervalles. Ce test permet de prendre en considération les écarts qui peuvent exister entre les deux ACS. La génération de graphiques à partir des nombres d'instances qui appartiennent aux intervalles facilite l'interprétation des données. Ainsi, on peut considérer qu'un ACS X surclasse un ACS Y si la courbe produite par l'ACS X est au dessus de celle produite par l'ACS Y. Ces courbes ne se croisent jamais tout au long du graphique.

Les résultats présentés sous forme de graphiques à la Figure 4.1 ne permettent pas d'affirmer que l'ACS classique domine stochastiquement l'ACS – R pour les meilleures solutions obtenues (MS), les résultats médians obtenus (SM) et les pires solutions obtenues (PS). En effet, dans les trois cas, les courbes produites par les deux ACS se croisent au moment où la déviation par rapport à l'optimum atteint les 200%.

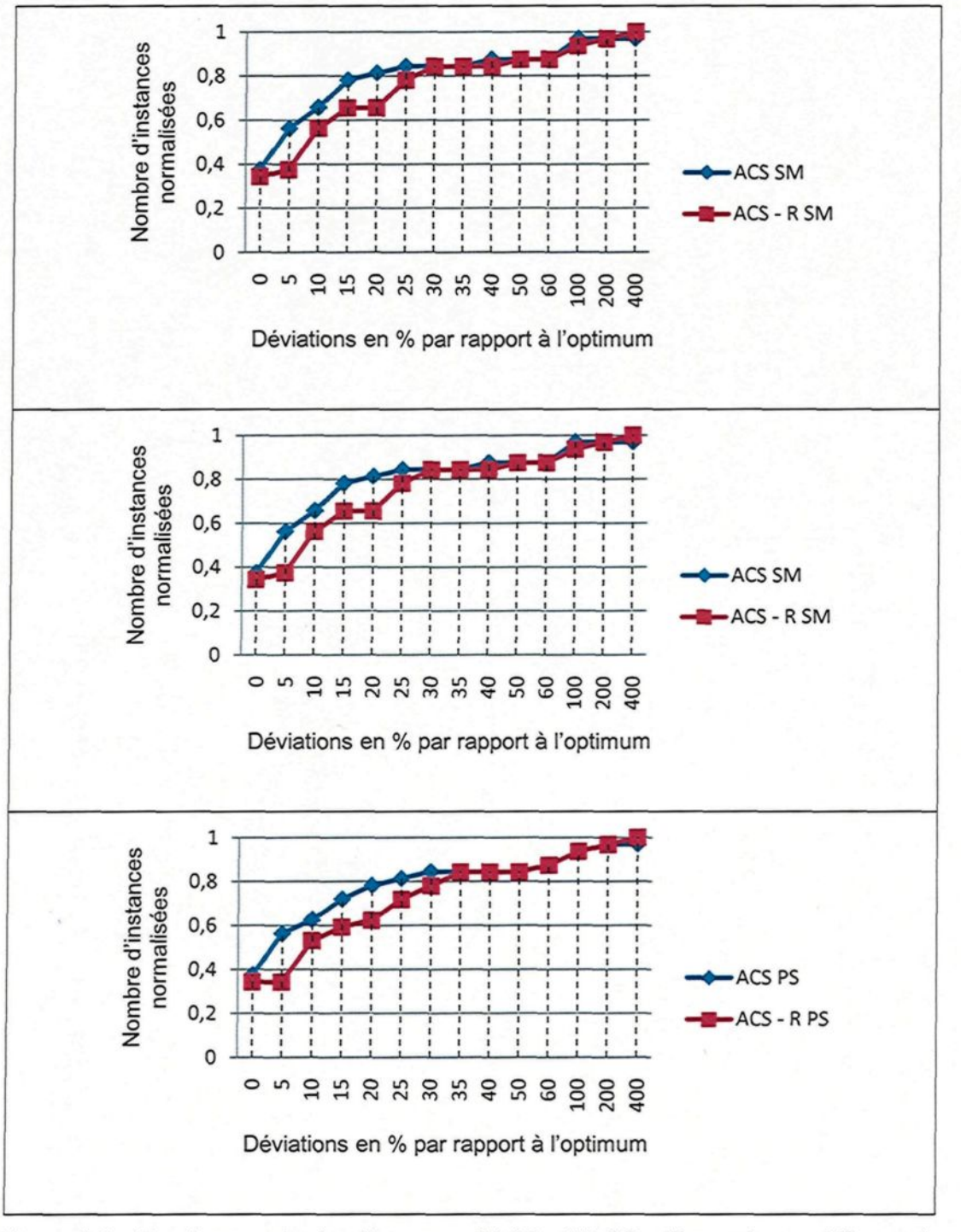

Figure 4.1 – Dominance stochastique pour l'ACS et l'ACS – R pour les problèmes de petite taille

Finalement, le test non paramétrique de rang signé de Wilcoxon a été appliqué à ces résultats de façon à comparer la performance des deux ACS. Ce test consiste à faire la somme des rangs signés positif et négatif séparément. L'hypothèse de base, H_0, est que les deux algorithmes sont équivalents. Le calcul de la valeur calculée du test se fait à partir de la plus petite somme des rangs. Ainsi, pour comparer les solutions des deux ACS, nous avons: $H0 : ACS = ACS - R$ pour l'hypothèse nulle et $H1 : ACS \neq ACS - R$ pour la contre-hypothèse. Ce test est réalisé à un seuil de 99% et toutes les instances pour lesquelles les 2 algorithmes obtiennent un résultat identique sont éliminées.

Les résultats du test de Wilcoxon sont présentés au Tableau 4.2. Dans la deuxième colonne, une valeur de W de -2,589 est obtenue pour la comparaison de la meilleure solution obtenue (MS). Cette valeur correspond à une probabilité de $p = 0{,}01$. Ces résultats permettent d'accepter l'hypothèse nulle et confirment ainsi que la différence de qualité de solution entre les deux ACS n'est pas statistiquement significative pour les meilleures solutions obtenues.

Pour leur part, les résultats du test de rang signé de Wilcoxon pour la médiane des solutions indiquent que l'hypothèse nulle est également acceptée car la valeur calculée de W est de $-2{,}329$ et correspond à une probabilité $p = 0{,}20$. Ceci confirme que statistiquement l'ACS – R produit des solutions médianes d'une qualité semblable à celle de l'ACS classique.

Pour les pires solutions (PS), le résultat du test non paramétrique de rang signé de Wilcoxon indique que la valeur calculée est de $W = -2{,}068$ correspondant à $p = 0{,}39$. Comme la valeur calculée est supérieure à la valeur critique ($W_{1-\alpha} = -2{,}58$), l'hypothèse nulle est acceptée, ce qui démontre encore une fois que statistiquement il n'y a pas une

différence significative entre les deux ACS au niveau de la qualité des pires solutions produites.

	ACS MS - ACS-R MS	ACS SM - ACS-R SM	ACS PS - ACS-R PS
W	-2,589	-2,329	-2,068
Asymp. Sig. (2-tailed)	0,010	0,020	0,039

Tableau 4.2 – Résultats des tests de rang signé de Wilcoxon entre l'ACS et l'ACS – R pour les problèmes de petite taille

Globalement, ces résultats confirment que les deux versions d'algorithmes performent de manière similaire. En ce qui concerne les temps de traitement, l'ACS prend en moyenne 15 secondes pour l'ensemble des instances tandis que le temps moyen pour l'ACS – R est de 32.5 secondes. Ceci est dû au fait que la *MA* est recalculée pour chaque fourmi à chaque fois qu'une tâche doit être choisie, tandis que pour l'ACS classique elle n'est calculée qu'une fois au début de l'algorithme.

4.2.2. Comparaison de la performance pour les problèmes de grande taille

Le Tableau 4.3 présente, de façon similaire, les résultats obtenus par les deux ACS sur le deuxième groupe de problèmes.

Une analyse empirique montre que, pour les meilleures solutions (MS), l'ACS donne de meilleurs résultats pour 19 instances de problèmes avec un écart à l'optimum qui varie de 0,59 % à 42,08%. Les deux ACS atteignent l'optimum pour les mêmes 10 instances de problèmes tandis que l'ACS – R est meilleur pour trois instances de problèmes avec un écart supplémentaire à l'optimum qui varie entre 0,81% et 300%.

PROB	ACS classique				ACS à rebours			
	MS	SM	PS	TT	MS	SM	PS	TT
551	53,55	72,13	87,43	152,74	95,63	115,85	140,98	176,13
552	0	0	0	0	0	0	0	0
553	2,15	3,24	3,73	154,05	3,89	4,64	5,32	178,02
554	6,49	11,92	14,98	154,77	13,01	15,96	18,13	180,04
555	0	0	0	151,16	0	0	0	176,19
556	0	0	0	0	0	0	0	0
557	10,41	12,36	13,58	151,15	16,42	18,49	20,41	180,5
558	14,95	20,77	23,46	154,79	22,22	25,82	27,52	179,73
651	39,18	49,63	68,66	261,3	61,19	67,91	95,9	258,98
652	0	0	0	0	0	0	0	0
653	2,22	2,92	3,64	263,67	4,25	5,2	6,03	258,52
654	6,4	8,06	9,77	263,88	5,29	9,19	11,53	261,09
655	2050	3000	6250	261,53	1750	2200	2550	252,48
656	0	0	0	0	0	0	0	0
657	14,17	16,9	18,58	260,17	20,41	23,21	25,16	255,91
658	22,94	26,79	30,54	263,84	24,51	29,35	31,73	256,65
751	51,45	85,27	124,07	316,38	72,61	83,82	88,8	345,61
752	0	0	0	0	0	0	0	0
753	2,26	2.9	3,23	317,15	4,12	4,62	5,19	347,27
754	8,75	13,13	17,55	318,12	7,94	11,3	15,29	350,24
755	0	0	0	314,46	0	0	0	349,49
756	0	0	0	0	0	0	0	0
757	14,31	16,39	20,11	316	24,17	26,3	27,53	348,34
758	19,81	23,51	28,19	317,8	22,62	25,14	28,78	350,91
851	70,83	89,19	108,07	407,48	84,64	94,14	100,26	445,29
852	0	0	0	0	0	0	0	0
853	2,47	3,11	3,55	412,2	4,89	5,46	6,1	434,68
854	4,48	6,56	8,46	410,55	5,07	6,53	7,87	448,69
855	42,76	74,38	97,53	410,49	82,33	115,19	125,8	441,97
856	0	0	0	0	0	0	0	0
857	14,8	15,9	17,22	413,57	20,66	22,9	24,72	442,32
858	17	20,58	22,58	413,86	21,36	23,15	24,72	450,59

Tableau 4.3 – Les déviations, en % par rapport à l'optimum, de l'ACS et de l'ACS – R pour les problèmes de grande taille

Pour l'instance 655, on note une amélioration substantielle de la qualité de solution produite par l'ACS – R par rapport à l'ACS.

Pour la médiane des solutions (SM), l'ACS donne de meilleures solutions que l'ACS – R sur 18 instances de problèmes. Les deux ACS atteignent l'optimum pour les mêmes 10 problèmes et l'ACS – R est meilleur pour quatre instances de problème, avec un écart supplémentaire à l'optimum de 800% pour le problème 655.

En ce qui concerne les pires solutions (PS), l'ACS est meilleur que l'ACS – R pour 17 instances de problèmes. Les deux ACS atteignent l'optimum pour les mêmes 10 instances de problèmes tandis que l'ACS – R est meilleur pour 5 instances de problèmes. Encore une fois, l'ACS – R réussit à creuser l'écart pour l'instance 655, atteignant cette fois-ci 3700% de déviation supplémentaire.

La Figure 4.2 illustre graphiquement les résultats du test de dominance stochastique pour les MS, SM et PS produits par les deux ACS. Pour les trois cas, ce test permet d'affirmer que l'ACS domine stochastiquement l'ACS – R, car les courbes produites par les deux ACS ne se croisent jamais.

Le test non paramétrique de rang signé de Wilcoxon, présenté au Tableau 4.4, est appliqué pour vérifier l'égalité dans la performance des deux ACS. Ce test est réalisé à un seuil de test à 99% et toutes les instances pour lesquelles les 2 algorithmes obtiennent un résultat identique sont éliminées. L'hypothèse nulle a été définie comme étant $H_0: ACS = ACS - R$ et l'hypothèse alternative est $H_1: ACS \neq ACS - R$.

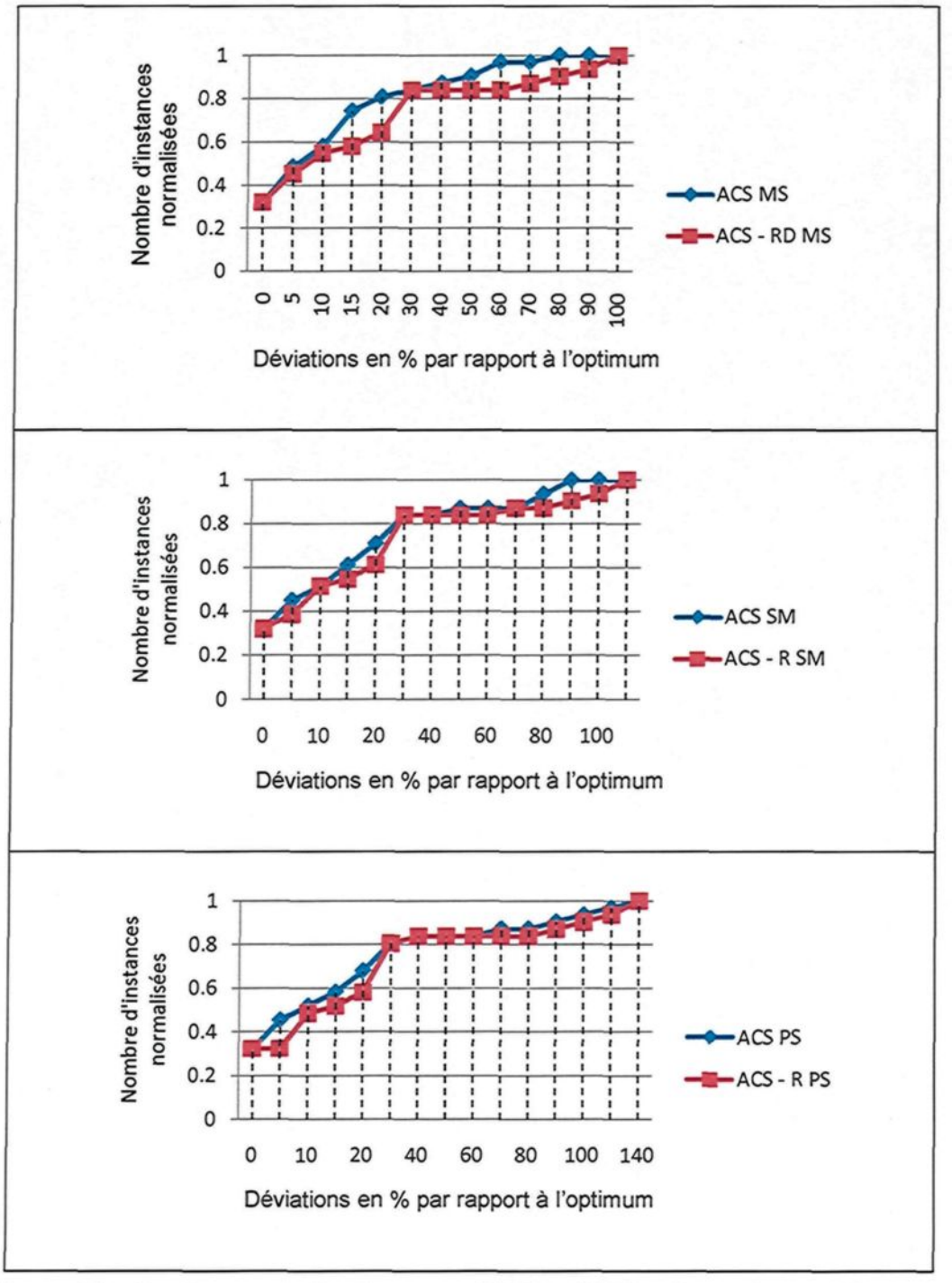

Figure 4.2 – Dominance stochastique pour l'ACS et l'ACS – R pour les problèmes de grande taille

	ACS MS - ACS - R MS	ACS SM - ACS - R SM	ACS PS - ACS - R PS
W	-3,230	-3,003	-1,883
Asymp. Sig. (2-tailed)	0,001	0,003	0,060

Tableau 4.4 – Résultats des tests de rang signé de Wilcoxon entre l'ACS et l'ACS – R pour les problèmes de grande taille

Pour les meilleures solutions, la valeur calculée est $W = -3{,}230$ et correspond à une probabilité $p = 0{,}01$. Comme la valeur calculée de W est inférieure à la valeur critique $W_{1-\alpha} = -2{,}58$, nous pouvons conclure que l'hypothèse nulle est rejetée. Cela nous permet d'affirmer que les deux ACS performent différemment avec l'avantage à l'ACS classique.

Le test non paramétrique de rang signé de Wilcoxon pour la médiane des solutions indique que la valeur calculée est de $W = -3{,}003$ pour une probabilité de $p = 0{,}03$. Comme la valeur critique est de $W_{1-\alpha} = -2{,}58$, on peut affirmer que l'hypothèse nulle est rejetée, ce qui nous permet de conclure que les deux ACS sont statistiquement différents en termes de qualité des solutions produites. Encore une fois, un avantage va à l'ACS classique.

Pour les pires solutions, le test non paramétrique de rang signé de Wilcoxon indique que la valeur calculée est de $W = -1{,}883$ pour une probabilité $p = 0{,}06$. Dans ce cas, l'hypothèse nulle est acceptée, ce qui nous permet de conclure que les deux ACS ont une performance statistiquement semblable relativement aux pires solutions trouvées.

En termes de temps moyens d'exécution, l'ACS classique prend en moyenne 214,4 secondes pour exécuter une instance de problème tandis que l'ACS – R prend en moyenne 230 secondes. L'ACS classique est plus rapide pour les mêmes raisons énoncées à la section antérieure.

Si pour les problèmes de petite taille, les deux ACS ont obtenu une performance très semblable pour les MS, SM et PS, cela n'est pas le cas pour les problèmes de grande taille. La performance des deux ACS est différente pour les meilleures solutions trouvées et la médiane des solutions, mais elle est équivalente pour les pires solutions produites. Cela nous permet de conclure que l'ACS-R a une performance similaire à l'ACS classique pour les problèmes de petite taille, tandis que l'ACS classique est meilleur pour les problèmes de grande taille.

4.3 Analyse comparative entre l'ACS classique et l'ACS – RD

Une première modification proposée à l'ACS – R consiste à rendre dynamique l'ajustement des paramètres. Nous allons vérifier la performance de l'ACS – RD par rapport à l'ACS sur les 32 problèmes tests de Ragatz (1993) et les 32 problèmes tests de Gagné, Price *et al.* [2002].

4.3.1. Comparaison de la performance pour les problèmes de petite taille

Le Tableau 4.5 présente les résultats produits par l'ACS et l'ACS – RD pour les problèmes de petite taille. Ce tableau a la même structure que celle du Tableau 4.1 présenté à la section 4.2.1.

L'analyse empirique permet de constater que, pour les MS, l'ACS classique donne de meilleurs résultats pour 17 des 32 instances de problèmes traités. L'ACS atteint l'optimum pour 11 instances de problèmes tandis que l'ACS – RD atteint l'optimum pour 12 instances de problèmes. L'ACS – RD est meilleur pour quatre instances de problèmes, avec une amélioration significative pour les instances 601, 605 et 701.

PROB	ACS classique				ACS – RD			
	MS	SM	PS	TT	MS	SM	PS	TT
401	0	0	0	6,73	4,44	15,56	22,22	21,93
402	0	0	0	0	0	0	0	0
403	0,23	0,31	2,02	6,75	4,68	6,7	7,99	22,05
404	3,66	6,84	8,34	6,74	0	8,9	17,06	22,03
405	0	0	0	0	0	0	0	0
406	0	0	0	0	0	0	0	0
407	1,4	12,28	21,49	6,75	21,76	28,16	33,53	22,02
408	2,51	3,41	4,06	6,74	5,67	7,57	8,94	22,29
501	4,21	5,56	8,43	14,15	5,36	7,85	9,58	34,75
502	0	0	0	0	0	0	0	0
503	0,77	2,8	3.8	14,01	3,89	4,86	6,75	34,91
504	0	0	0	0	0	0	0	0
505	0	0	0	14,18	0	0	0	35,19
506	0	0	0	0	0	0	0	0
507	9,41	13,19	14,41	14,14	21,48	24,47	25,97	35,15
508	27,73	39,09	50,23	14,18	35,98	44,18	50,81	35,18
601	375	487,5	541,67	23,83	183,33	233,33	291,67	51,34
602	0	0	0	0	0	0	0	0
603	1,77	3,03	3,92	23,98	4,79	5,36	5,98	51,45
604	2,44	2,94	3,69	23,85	5,5	9,81	11,01	51,99
605	66,23	83,33	101,32	23,58	56,14	64,69	86,4	51,31
606	0	0	0	0	0	0	0	0
607	11,55	14,11	15,91	23,86	17,46	22,14	23,26	51,55
608	18,62	23,58	29,52	23,85	21,53	28,44	31,8	51,82
701	41,24	80,93	91,75	38,11	35,05	122,68	179,38	74,61
702	0	0	0	0	0	0	0	0
703	2,59	3,3	3,75	37,98	4,67	5,69	7,06	74,72
704	6,27	8,31	10,65	38,42	7,87	11,73	15,28	75,01
705	53,5	72	87	38,21	58	64,5	72	74,61
706	0	0	0	0	0	0	0	0
707	8,52	11,73	13,39	38,23	20,9	23,51	24,36	74,57
708	13,5	15,34	16,36	38,52	23,12	25,24	27,35	75,17

Tableau 4.5 – Les déviations, en % par rapport à l'optimum, de l'ACS et de l'ACS – RD pour les problèmes de petite taille

Pour la médiane des solutions obtenues par les deux ACS, l'ACS obtient les meilleurs résultats pour 18 instances de problèmes, les deux ACS ont atteint l'optimum pour les mêmes 11 instances et l'ACS – RD est meilleur pour trois instances de problèmes. On note des gains intéressants sur les instances 601, 605 et 705.

Pour les pires solutions produites par les deux ACS, l'ACS donne de meilleurs résultats pour 18 instances, les deux ACS atteignent l'optimum pour les mêmes 11 instances de problèmes et l'ACS – RD est meilleur pour trois instances de problèmes. Encore une fois, cela concerne les instances 601, 605 et 705.

Le test de dominance stochastique pour les MS, SM et PS ne permet pas de distinguer quel ACS domine stochastiquement l'autre. Comme on peut le constater sur les graphiques de la Figure 4.3, les courbes produites par les deux ACS pour les trois cas se croisent à une déviation donnée.

Le Tableau 4.6 présente les résultats des tests non paramétriques de rang signé de Wilcoxon pour un seuil de 99%. En ce qui concerne les meilleures solutions, la valeur calculée $W = -1{,}894$ correspond à une probabilité $p = 0{,}058$, ce qui indique que l'hypothèse nulle est acceptée. Dans ce cas-ci, nous pouvons dire que l'ACS et l'ACS – RD sont statistiquement équivalents pour les meilleures solutions.

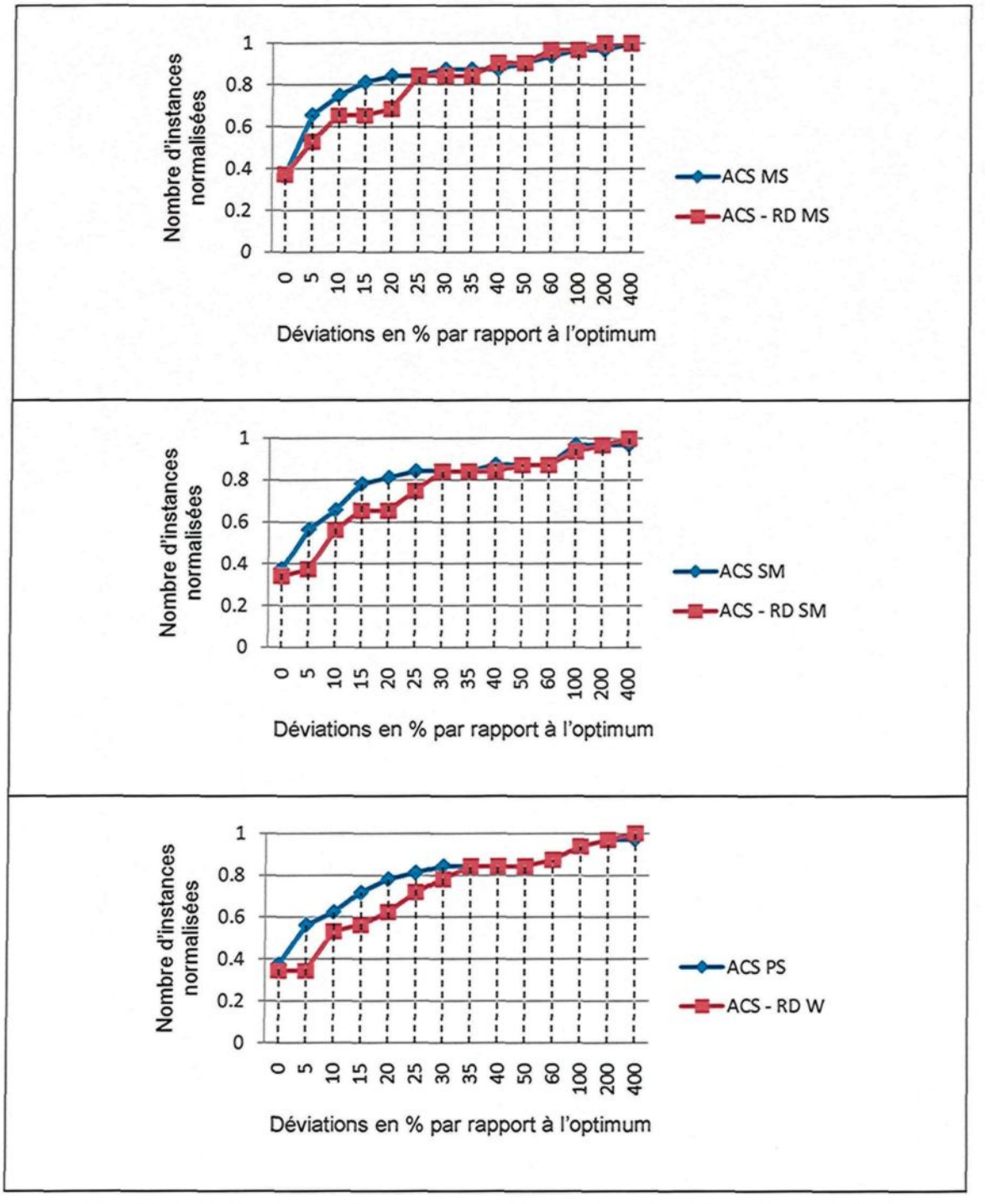

Figure 4.3 – Dominance stochastique pour l'ACS et l'ACS – RD pour les problèmes de petite taille

Le résultat du test non paramétrique de rang signé de Wilcoxon effectué pour la médiane des solutions (SM), indique que l'hypothèse nulle est acceptée car la valeur calculée $W = -2{,}207$, qui correspond à une probabilité $p = 0{,}027$, est inférieure à la valeur critique. Dans ce cas-ci, nous pouvons conclure que les deux ACS sont statistiquement équivalents pour la médiane des solutions.

Pour les pires solutions (PS), le résultat du test non paramétrique de rang signé de Wilcoxon nous permet d'accepter l'hypothèse nulle. La valeur calculée est de $-2{,}068$ pour une probabilité $p = 0{,}039$, ce qui est supérieure à la valeur critique ($W_{1-\alpha} = -2{,}58$). Ceci nous indique que les deux ACS performent de façon similaire pour les pires solutions produites.

	ACS MS - ACS-RD MS	ACS SM - ACS-RD SM	ACS PS - ACS-RD PS
W	-1,894	-2,207	-2,068
Asymp. Sig. (2-tailed)	0,058	0,027	0,039

Tableau 4.6 – Résultats des tests de rang signé de Wilcoxon entre l'ACS et l'ACS – RD pour les problèmes de petite taille

Après avoir comparé l'ACS avec l'ACS–RD pour les meilleures solutions, la médiane des solutions et les pires solutions produites, on constate que les deux ACS ont une performance similaire.

Pour les temps d'exécution des deux ACS, l'ACS est le plus rapide avec une moyenne de 15 secondes pour les 32 instances de problèmes, tandis que l'ACS – RD a une moyenne de 32,6 secondes. Comme référé à la section 4.2.1, cette différence est due surtout au calcul de la *MA*.

L'ACS – RD a été conçu pour rendre l'ACS – R plus intelligent au niveau du choix des paramètres à utiliser pour les problèmes à caractéristiques différentes. Pour les problèmes de petite taille, le test non paramétrique de rang signé de Wilcoxon nous permet de conclure que la différence de qualité de solution entre les deux ACS n'est statistiquement pas significative, même si empiriquement l'ACS – R semble donner de meilleures solutions. En termes de temps d'exécution, on peut considérer que, pour la moyenne des temps d'exécution, l'ACS – RD et l'ACS – R ont la même performance.

4.3.2. Comparaison de la performance pour les problèmes de grande taille

Le Tableau 4.7 présente les résultats produits par l'ACS et l'ACS – RD pour les problèmes de grande taille. Ce tableau a la même structure que celle du Tableau 4.1 présenté précédemment.

L'analyse empirique des résultats pour les MS démontre que l'ACS est meilleur que l'ACS – RD pour 20 instances de problèmes avec des écarts supplémentaires à l'optimum qui varient de 0,71% à 55,47%. Les deux ACS atteignent l'optimum pour les mêmes 10 problèmes et l'ACS – RD est meilleur pour deux problèmes. Comme déjà montré précédemment, l'ACS – RD réussit à améliorer fortement l'instance 655.

En ce qui concerne les SM, l'ACS est meilleur pour 19 problèmes. Les deux ACS atteignent l'optimum pour les mêmes 10 problèmes et l'ACS – RD est meilleur pour trois problèmes. Des écarts plus importants sont obtenus pour les instances 655 et 751. Les instances où l'ACS – RD donne de meilleurs résultats sont les mêmes instances pour lesquelles l'ACS – R a aussi obtenu de meilleurs résultats.

PROB	ACS classique				ACS – RD			
	MS	SM	PS	TT	MS	SM	PS	TT
551	53,55	72,13	87,43	152,74	96,17	114,75	132,24	172,37
552	0	0	0	0	0	0	0	0
553	2,15	3,24	3,73	154,05	3,52	4,69	5,52	173,29
554	6,49	11,92	14,98	154,77	14,62	16,14	17,85	172,78
555	0	0	0	151,16	0	0	0	175,38
556	0	0	0	0	0	0	0	0
557	10,41	12,36	13,58	151,15	15,58	18,2	19,38	174,68
558	14,95	20,77	23,46	154,79	22,41	25,29	28,14	175,75
651	39,18	49,63	68,66	261,3	63,43	69,03	77,99	254,49
652	0	0	0	0	0	0	0	0
653	2,22	2,92	3,64	263,67	4,55	5,07	5,54	253,24
654	6,4	8,06	9,77	263,88	4,5	8,88	11,72	255,96
655	2050	3000	6250	261,53	1950	2300	2650	254,14
656	0	0	0	0	0	0	0	0
657	14,17	16,9	18,58	260,17	19,06	23,38	25	258,34
658	22.94	26,79	30,54	263,84	24,79	28,65	32,09	256,01
751	51,45	85,27	124,07	316,38	75,1	81.95	92.53	340,53
752	0	0	0	0	0	0	0	0
753	2,26	2,9	3,23	317,15	4,21	4,72	5,53	342,72
754	8,75	13,13	17,55	318,12	9,55	12,41	15.22	343,85
755	0	0	0	314,46	0	0	0	344,7
756	0	0	0	0	0	0	0	0
757	14,31	16,39	20,11	316	23,58	25,57	27,54	343,7
758	19,81	23,51	28,19	317,8	23,55	25,75	27,74	345,87
851	70,83	89,19	108,07	407,48	86,46	91,41	101.3	429,31
852	0	0	0	0	0	0	0	0
853	2,47	3,11	3,55	412,2	5,03	5,4	5,89	434,41
854	4,48	6,56	8,46	410,55	5,19	6,58	7,17	437,59
855	42,76	74,38	97,53	410,49	98,23	110,42	125,8	439,45
856	0	0	0	0	0	0	0	0
857	14,8	15,9	17,22	413,57	20,68	22,55	24,33	436,51
858	17	20,58	22,58	413,86	20,84	23,69	25,83	434,77

Tableau 4.7 – Les déviations, en % par rapport à l'optimum, de l'ACS et de l'ACS – RD pour les problèmes de grande taille

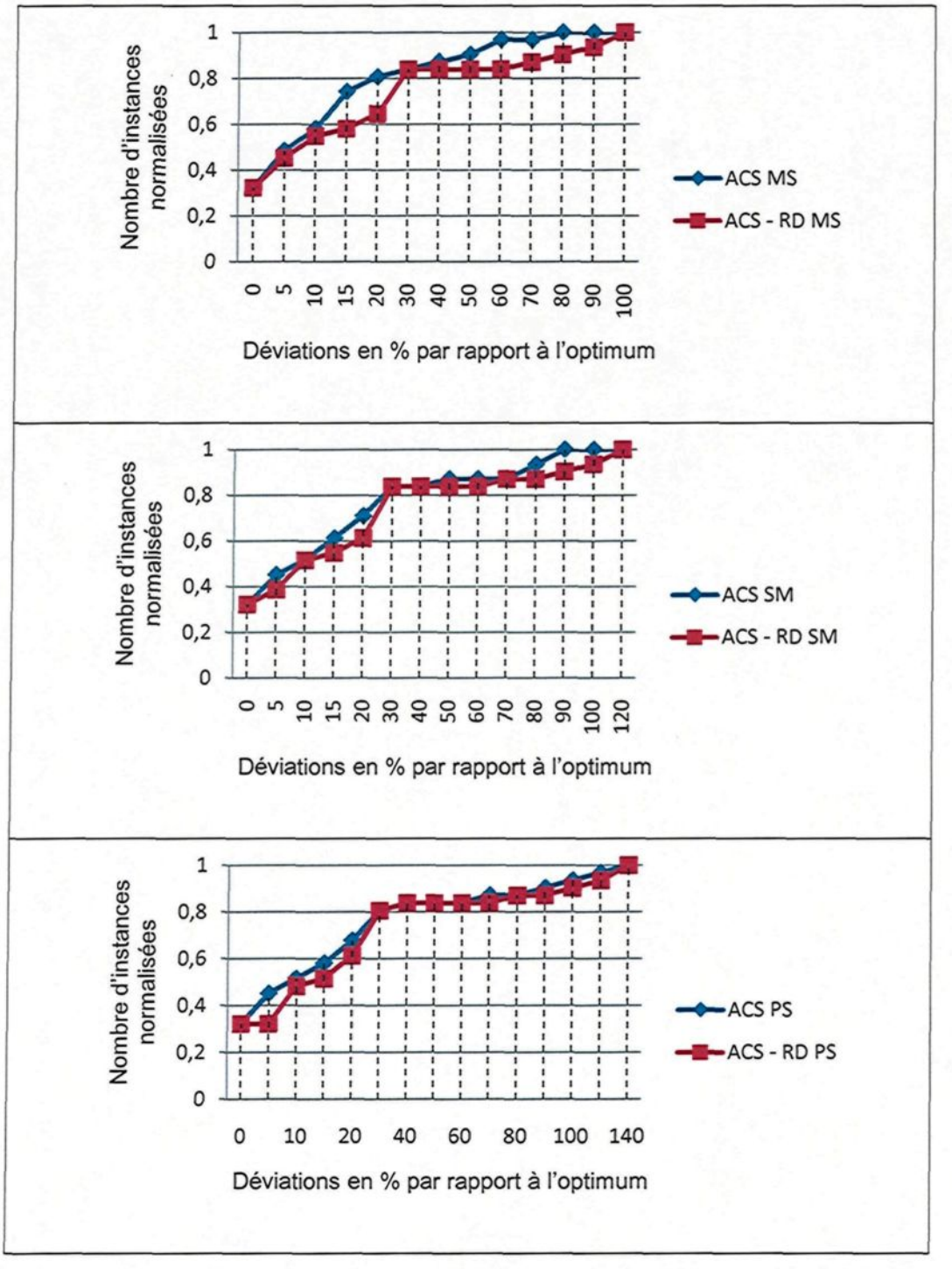

Figure 4.4 – Dominance stochastique pour l'ACS et l'ACS – RD pour les problèmes de grande taille

Pour les pires solutions (PS), l'ACS est meilleur pour 17 instances de problèmes. Les deux ACS atteignent l'optimum pour les mêmes 10 instances de problèmes tandis que l'ACS – RD est meilleur pour cinq instances de problèmes.

Le test de dominance stochastique a été réalisé pour les MS, SM et PS. Les résultats, présentés sous forme de graphiques à la Figure 4.4, nous permettent d'affirmer que l'ACS classique domine stochastiquement l'ASC – RD pour les problèmes de grande taille. Encore une fois, les courbes produites par les deux ACS ne se croisent pas.

Le test non paramétrique de rang signé de Wilcoxon, présenté au Tableau 4.8, a été réalisé à un seuil de test de 99% pour les MS, SM, PS de façon à valider l'égalité des ACS au niveau de leurs performances.

Pour les MS, la valeur calculée est de $W = -3{,}230$ et correspond à une probabilité $p = 0{,}001$. Comme la valeur calculée est inférieure à la valeur critique ($W_{1-\alpha} = -2{,}58$) à un seuil à 99%, on rejette l'hypothèse nulle. Ainsi, il a été démontré que l'ACS présente une performance significativement supérieure à celle de l'ACS – R au niveau de la qualité des solutions produites.

En appliquant le test aux résultats médians (SM), on constate que la valeur calculée est de $W = -2{,}938$ et correspond à une probabilité $p = 0{,}03$. Comme la valeur critique pour le seuil de test proposé est supérieure à la valeur calculée, l'hypothèse nulle est rejetée. Ceci amène à conclure que les deux ACS n'ont pas la même performance en termes de qualité de solutions produites. Dans ce cas, l'avantage est accordé à l'ACS classique.

En ce qui concerne les pires solutions (PS), la valeur calculée du test est de $W = -1{,}899$ et correspond à une probabilité $p = 0{,}58$. Pour le seuil de test proposé, l'hypothèse nulle est acceptée car la valeur critique est inférieure à la valeur calculée.

	ACS MS - ACS - RD MS	ACS SM - ACS - RD SM	ACS PS - ACS - RD PS
W	-3,230	-2,938	-1,899
Asymp. Sig. (2-tailed)	0,001	0,003	0,058

Tableau 4.8 – Résultats des testes de rang signé de Wilcoxon entre l'ACS et l'ACS – RD pour les problèmes de grande taille

Après avoir comparé l'ACS et l'ACS – RD pour les problèmes de grande taille, on peut conclure que les deux ACS ont la même performance seulement pour le cas des PS. Pour les MS et SM, les deux ACS performent différemment en termes de qualité de solutions produites, avec un avantage à l'ACS classique. En termes de temps d'exécution, l'ACS classique est plus rapide avec une différence pour la moyenne des temps d'exécution de 12 secondes par rapport à l'ACS classique. Encore une fois, le calcul de la *MA* par l'ACS – RD joue un rôle important comme déjà référé à la section 4.2.1.

En général, l'ACS – RD et l'ACS – R ont une performance très semblable en termes de qualité de solution pour les problèmes de grande taille même si quelques différences sont remarquées. L'ACS – RD donne de meilleurs résultats pour les PS des problèmes 551 et 651 et pour la SM du problème 855. Pour les MS, SM et PS du problème 655, l'ACS – R est meilleur, ainsi que pour la médiane des solutions (SM) du problème 855. Pour le problème 655, l'écart supplémentaire par rapport à l'optimum est de 200% et 100% respectivement, à l'avantage de l'ACS – R. Pour les temps d'exécution, on peut considérer que les deux ACS ont la même performance, même si l'ACS – RD a un avantage de 3,7 secondes pour la moyenne des temps d'exécution.

4.4 Résultats de l'ACS à rebours avec règle de priorité

La deuxième modification proposée à l'ACS – R consiste à utiliser une règle de priorité pour compléter la solution dans le cas où il n'y aurait plus de tâches en retard à placer dans une séquence en construction.

La suite de l'analyse consiste à évaluer la performance de l'ACS – RP par rapport à l'ACS sur les 32 problèmes tests de Ragatz (1993) et les 32 problèmes tests de Gagné, Price *et al.* [2002].

4.4.1. Comparaison de la performance pour les problèmes de petite taille

Le Tableau 4.9 présente les résultats produits par l'ACS et l'ACS – RP. Ce tableau a la même structure que le Tableau 4.1 présenté ci-dessus.

En ce qui concerne les meilleures solutions obtenues par les deux ACS (MS), l'analyse empirique permet d'affirmer que l'ACS produit de meilleures solutions pour 12 instances de problèmes, avec un écart supplémentaire à l'optimum qui varie entre 0,99% et 16,49%. L'ACS atteint l'optimum pour 12 instances de problèmes, tandis que l'ACS – RP atteint l'optimum pour 14 instances de problèmes. L'ACS – RP produit de meilleures solutions pour 8 instances de problèmes et l'écart supplémentaire pour ces 8 instances varie entre 0,58% et 225.

Pour la médiane des solutions (MS), l'ACS produit de meilleures solutions pour 12 instances de problèmes avec un écart supplémentaire à l'optimum qui varie entre 0,05% et 7,17%. Les deux ACS atteignent l'optimum pour les mêmes 12 instances de problèmes, tandis que l'ACS – RP donne de meilleurs résultats pour 12 instances de problèmes avec un écart supplémentaire à l'optimum qui varie entre 0,06% et 299,58.

PROB	ACS classique				ACS – RP			
	MS	SM	PS	TT	MS	SM	PS	TT
401	0	0	0	6,73	0	0	4,44	37,12
402	0	0	0	0	0	0	0	0
403	0,23	0,31	2,02	6,75	1,81	2,66	2,84	24,49
404	3,66	6,84	8,34	6,74	0	6,75	6,75	22,17
405	0	0	0	0	0	0	0	0
406	0	0	0	0	0	0	0	0
407	1,4	12,28	21,49	6,75	7,04	11,93	18	24,72
408	2,51	3,41	4,06	6,74	1,93	3,29	5,05	22,35
501	4,21	5,56	8,43	14,15	0	0,77	1,53	129,44
502	0	0	0	0	0	0	0	0
503	0,77	2,8	3,8	14,01	2	2,69	2,97	68,27
504	0	0	0	0	0	0	0	0
505	0	0	0	14,18	0	0	0	141,93
506	0	0	0	0	0	0	0	0
507	9,41	13,19	14,41	14,14	17,29	19,28	20,93	45,7
508	27,73	39,09	50,23	14,18	28,72	31,44	36,61	36,22
601	375	487,5	541,67	23,83	150	166,67	191,67	328,96
602	0	0	0	0	0	0	0	0
603	1,77	3,03	3,92	23,98	3,74	4.68	5,21	73,42
604	2,44	2,94	3,69	23,85	4,19	5.3	6,83	52,18
605	66,23	83,33	101,32	23,58	33,33	41,23	47,81	263,17
606	0	0	0	0	0	0	0	0
607	11,55	14,11	15,91	23,86	15,99	17,78	19.52	74,35
608	18,62	23,58	29,52	23,85	14,88	21,03	24,87	52,07
701	41,24	80,93	91,75	38,11	57,73	74,23	89,69	616,64
702	0	0	0	0	0	0	0	0
703	2,59	3,3	3,75	37,98	3,72	4,2	4,94	120,63
704	6,27	8,31	10,65	38,42	5,25	6,71	8,49	75,77
705	53,5	72	87	38,21	41	50,5	62	604,73
706	0	0	0	0	0	0	0	0
707	8,52	11,73	13,39	38,23	17,26	18,66	19,8	126,32
708	13,5	15,34	16,36	38,52	19,1	20,95	22,33	75,88

Tableau 4.9 – Les déviations, en % par rapport à l'optimum de l'ACS et de l'ACS – RP pour les problèmes de petite taille

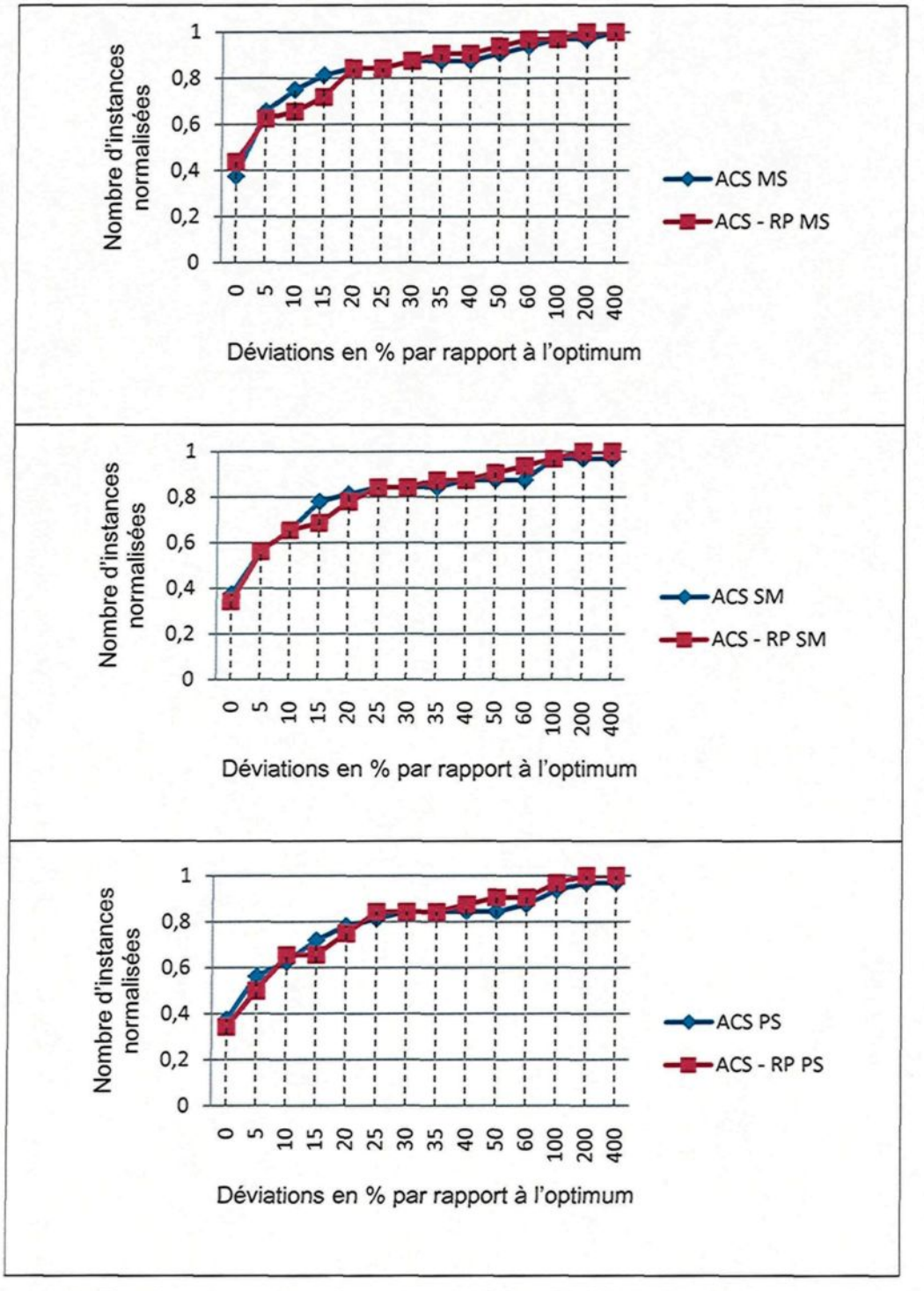

Figure 4.5 – Dominance stochastique pour l'ACS et l'ACS – RP pour les problèmes de petite taille

Pour le cas des pires solutions produites (PS), l'ACS classique donne de meilleures solutions pour 10 instances de problèmes et les écarts varient entre 0,82% et 6,52%. L'ACS atteint l'optimum pour 12 instances de problèmes tandis que l'ACS – RP atteint l'optimum pour 11 instances de problèmes. L'ACS – RP est meilleur pour 11 instances de problèmes et l'écart pour ces 11 instances varie entre 0,83% et 350%.

À partir de l'analyse empirique, il est encore une fois difficile de partager les deux ACS. La dominance stochastique pour les MS, SM et PS, présentée sous forme de graphiques à la Figure 4.5, ne permet pas de conclure lequel des deux ACS domine stochastiquement l'autre car les courbes produites par les deux ACS se croisent à un moment donné.

Les résultats du test non paramétrique de rang signé de Wilcoxon, présentés au Tableau 4.10, indiquent que, pour les MS, l'hypothèse nulle a été acceptée, parce que la valeur calculée ($W = -0{,}56$) à une probabilité $p = 0{,}575$ est supérieure à la valeur critique ($W_{1-\alpha} = -2{,}58$) à un seuil de test à 99%.

Pour la médiane des solutions (SM), le résultat du test non paramétrique de rang signé de Wilcoxon, présenté à la troisième colonne du Tableau 4.10, indique que l'hypothèse nulle est acceptée. La valeur de $W = -0{,}365$ correspond à une probabilité $p = 0{,}715$ et est supérieure à la valeur critique au seuil proposé.

Pour les pires solutions (PS), le test non paramétrique de rang signé de Wilcoxon, présenté à la quatrième colonne du Tableau 4.10, permet, encore une fois, d'accepter l'hypothèse nulle car la valeur calculée $W = -0{,}886$ à une probabilité $p = 0{,}375$ est, une fois de plus, inférieure à la valeur critique.

	ACS MS - ACS-RP MS	ACS SM - ACS-RP SM	ACS PS - ACS-RP PS
W	-0,560	-0,821	-0,886
Asymp. Sig. (2-tailed)	0,575	0,411	0,375

Tableau 4.10 – Résultats des tests de rang signé de Wilcoxon entre l'ACS et l'ACS – RP pour les problèmes de petite taille

En analysant les trois groupes de solutions produites par l'ACS et l'ACS – RP, nous pouvons affirmer que les deux ACS ont la même performance car l'hypothèse nulle a été acceptée pour les trois groupes de solutions. Pour la moyenne des temps d'exécution, l'ACS a une moyenne de 14,8 secondes tandis que l'ACS – RP a une moyenne de 94,2 secondes. L'ACS – RP est moins performant non seulement à cause du calcul de la *MA*, mais aussi à cause de l'application de la règle de priorité utilisée.

L'ACS – RP réussit à améliorer la qualité des solutions produites initialement par l'ACS – R. L'introduction de la règle de priorité s'est avérée intéressante en termes de qualité de solutions produites. En contrepartie, le temps d'exécution pour l'ACS – RP a augmenté considérablement par rapport à l'ACS – R, passant d'une moyenne de 32,9 secondes pour l'ACS – R à 94,2 pour l'ACS – RP. Il faut également noter que l'application de la règle de priorité représente une forme de recherche locale.

4.4.2. Comparaison de la performance pour les problèmes de grande taille

Le Tableau 4.11 présente les résultats produits par l'ACS et l'ACS –RP pour les problèmes de grande taille. Ce tableau a la même structure que le Tableau 4.1 présenté précédemment.

PROB	ACS classique				ACS – RP			
	MS	SM	PS	TT	MS	SM	PS	TT
551	53,55	72,13	87,43	152,74	225,14	335,79	427,87	1637,61
552	0	0	0	0	0	0	0	0
553	2,15	3,24	3,73	154,05	8,79	11,97	12,79	1638,15
554	6,49	11,92	14,98	154,77	31,27	36,01	46,56	1637,64
555	0	0	0	151,16	0	0	0	1638,17
556	0	0	0	0	0	0	0	0
557	10,41	12,36	13,58	151,15	23,85	25,14	26,53	1637,98
558	14,95	20,77	23,46	154,79	52,43	55,32	57,18	1637,84
651	39,18	49,63	68,66	261,3	291,42	443,47	586,19	3207,69
652	0	0	0	0	0	0	0	0
653	2,22	2,92	3,64	263,67	13,22	15,73	16,28	3204,9
654	6,4	8,06	9,77	263,88	51,93	52,86	54,85	3207,82
655	2050	3000	6250	261,53	28300	56575	69000	3207,34
656	0	0	0	0	0	0	0	0
657	14,17	16,9	18,58	260,17	20,93	23,11	25,16	3272,94
658	22,94	26,79	30,54	263,84	82,81	82,81	82,81	3211,71
751	51,45	85,27	124,07	316,38	670,54	837,34	999,17	5270,67
752	0	0	0	0	0	0	0	0
753	2,26	2,9	3,23	317,15	11,58	13,26	14,3	5264,42
754	8,75	13,13	17,55	318,12	62,27	63,76	65,67	5268,31
755	0	0	0	314,46	0	0	0	5260,25
756	0	0	0	0	0	0	0	0
757	14,31	16,39	20,11	316	28,64	32,42	34,3	5266,43
758	19,81	23,51	28,19	317,8	65,88	66,55	67,58	5263,99
851	70,83	89,19	108,07	407,48	640,89	765,23	845,31	7939,99
852	0	0	0	0	0	0	0	0
853	2,47	3,11	3,55	412,2	13,96	15,04	15,55	7935,47
854	4,48	6,56	8,46	410,55	38,7	39,58	39,8	7934,81
855	42,76	74,38	97,53	410,49	608,83	741,17	840,28	7964,62
856	0	0	0	0	0	0	0	0
857	14,8	15,9	17,22	413,57	24,17	26,39	27,94	7930,22
858	17	20,58	22,58	413,86	63,02	63,23	64,31	7926,69

Tableau 4.11 – Les déviations, en % par rapport à l'optimum, de l'ACS et de l'ACS – RP pour les problèmes de grande taille

En faisant une analyse empirique du Tableau 4.11 pour les meilleures solutions (MS), on note que l'ACS est meilleur pour 22 instances de problèmes, avec des écarts supplémentaires qui varient de 6,64% à 26 250%. Les deux ACS atteignent l'optimum pour les mêmes 10 instances et l'ACS – RP n'est meilleur pour aucune instance de problème.

En ce qui concerne la médiane des solutions (SM), l'ACS est meilleur pour 22 instances de problèmes, cette fois-ci avec un écart supplémentaire qui varie entre 6,21% et 53 575%, ce qui est supérieur à l'écart existant pour les MS. Les deux ACS atteignent l'optimum pour les mêmes 10 instances de problèmes et l'ACS – RP n'est meilleur pour aucune instance de problème.

Pour les pires solutions, la situation se répète, c'est-à-dire que l'ACS est meilleur pour 22 instances de problèmes, avec un écart supplémentaire à l'optimum qui varie de 6,58% à 62 750%. Les deux ACS atteignent l'optimum pour les 10 autres instances. L'ACS – RP n'est meilleur pour aucune instance de problème.

L'analyse stochastique, présentée sous forme de graphiques à la Figure 4.6, démontre qu'il y a une nette dominance de l'ACS vis-à-vis de l'ACS – RP pour les MS, SM et PS.

Les courbes produites par les deux ACS sont éloignées l'une de l'autre, ce qui permet de distinguer clairement que l'ACS domine stochastiquement l'ACS – RP pour les problèmes de grande taille.

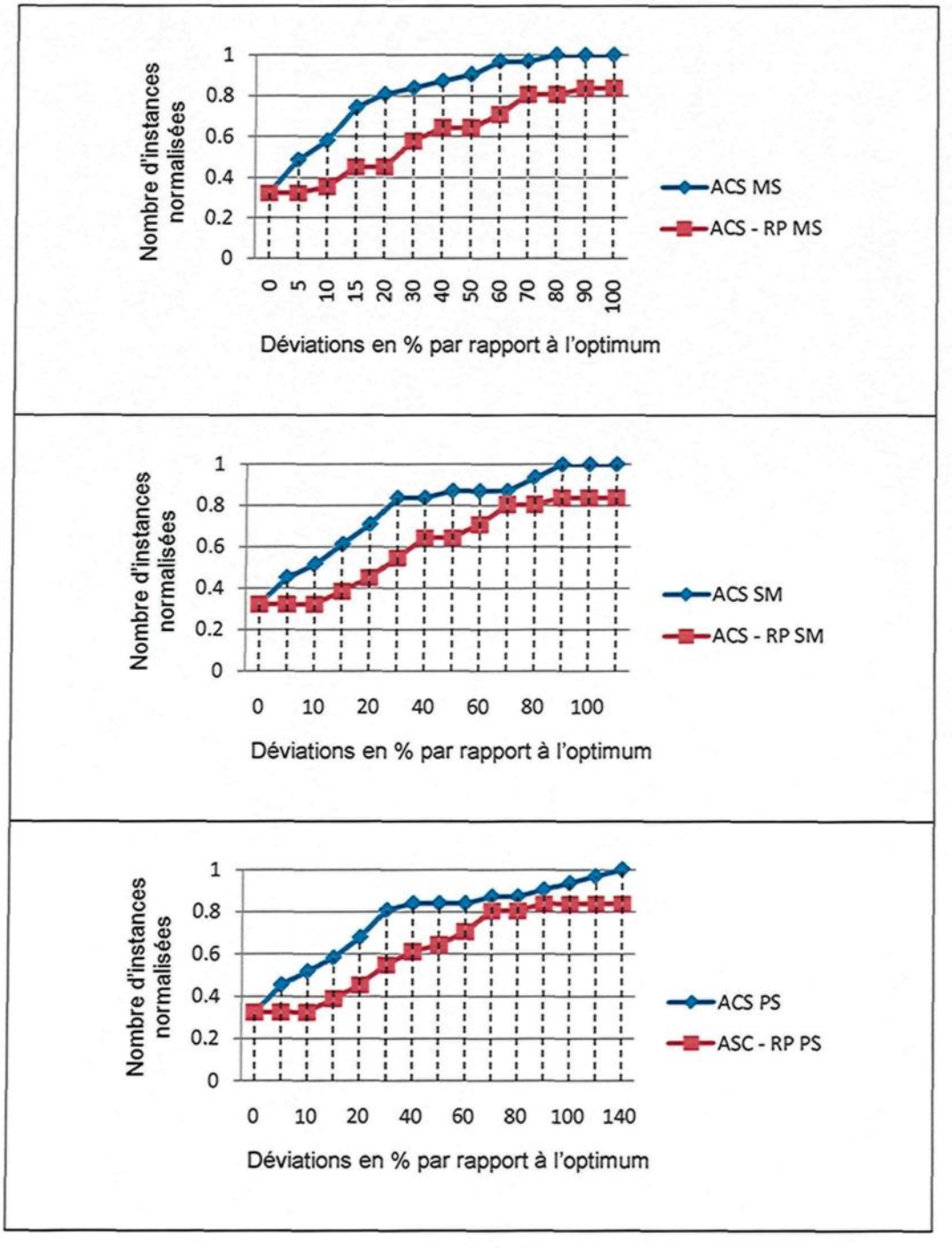

Figure 4.6 – Dominance stochastique pour l'ACS et l'ACS – RP pour les problèmes de grande taille

Le test non paramétrique de rang signé de Wilcoxon, présenté au Tableau 4.12 indique que le test est rejeté pour les MS, SM et PS. Pour les trois cas, la valeur calculée $W = -4{,}107$ a une probabilité $p = 0$, ce qui est inférieur à la valeur critique à un seuil de test à 99%.

	ACS MS - ACS - RP MS	ACS SM - ACS - RP SM	ACS PS - ACS - RP PS
Z	-4,107	-4,107	-4,107
Asymp. Sig. (2-tailed)	0,000	0,000	0,000

Tableau 4.12 – Résultats des tests de rang signé de Wilcoxon entre l'ACS et l'ACS – RP pour les problèmes de grande taille

En analysant les trois groupes de solutions, on constate que la règle de priorité n'est pas adaptée aux problèmes de grande taille. Cette forme de recherche locale n'est pas très efficace et devra être repensée pour mieux couvrir l'espace de solution. Les résultats obtenus sont encore moins intéressants que ceux produits par l'ACS – R et l'ACS – RD. En ce qui concerne le temps de calcul, l'ACS – RP prend en moyenne 3386 secondes pour l'ensemble des instances de problème, ce qui est largement supérieur à celui de l'ACS, l'ACS – R et l'ACS – RD, avec une moyenne de 214,4, 230,3 et 226,5 secondes chacun. Tout cela nous amène à conclure que l'ACS – RP est le moins performant des ACS développés pour les problèmes de grande taille.

4.5 Réflexion sur les résultats obtenus

Les résultats produits par les différents ACS développés donnent lieu à une réflexion qui nous permettra de conclure sur leurs performances.

L'ACS – R, l'ACS – RD et l'ACS – RP ont été comparés à l'ACS classique au niveau de la qualité des solutions produites et au niveau du temps d'exécution pour les problèmes de petite taille et pour ceux de grande taille.

Pour les problèmes de petite taille, les tests non paramétriques de rang signé de Wilcoxon ont démontré que la performance de ces algorithmes est équivalente à celui de l'ACS classique pour les meilleures solutions, la médiane des solutions et les pires solutions.

Les améliorations relatives à la qualité des solutions produites par les ACS proposés pour certaines instances de problèmes sont très significatives, comme c'est le cas pour les instances de problème 601 et 605. Cela nous porte à croire que les concepts proposés sont prometteurs pour le problème à l'étude.

En ce qui concerne les temps de calcul, les ACS proposés sont plus lents que l'ACS classique. Le fait de recalculer la marge arrière à chaque fois qu'une tâche doit être choisie, ajoute un temps de calcul supplémentaire tandis que, pour l'ACS classique, la marge est calculée au début de l'algorithme et elle est statique tout au long de l'exécution de l'ACS – R. Ce calcul est toutefois nécessaire pour guider adéquatement le choix des tâches par la règle de transition. Pour l'ACS – RP, au temps de calcul de la marge arrière s'ajoute l'application de la règle de priorité, ce qui fait augmenter considérablement les temps de calcul. L'application de la règle de priorité est une forme de recherche locale et on s'attendrait à obtenir une meilleure qualité de solution. L'expérimentation d'autres heuristiques plus performantes est une avenue à explorer.

Pour réussir à faire le bon choix des exposants associés à la règle de transition de l'ACS – R, ces exposants doivent être choisis manuellement pour chaque problème. Pour éviter cet inconvénient, on a développé la version RD de l'ACS à rebours qui permet un choix des paramètres de façon dynamique. Même si l'ACS – RD donne des résultats

équivalents à l'ACS – R, cet algorithme est beaucoup plus simple à utiliser car un calibrage des paramètres est non nécessaire.

Pour les problèmes de grande taille, les tests non paramétriques de rang signé de Wilcoxon réalisés pour les meilleures solutions, la médiane des solutions et les pires solutions ont permis de conclure que les différentes versions de l'ACS à rebours n'ont pas réussi à être aussi performantes que l'ACS classique au niveau de la qualité des solutions produites. Au niveau des temps d'exécution, l'ACS classique a, encore une fois, été plus rapide pour les mêmes raisons énoncées précédemment.

Ceci démontre qu'il y a encore des améliorations qui doivent être apportées à l'ACS à rebours pour qu'il devienne aussi performant que l'ACS classique non seulement au niveau de la qualité de solution produites, mais aussi au niveau des temps d'exécution. L'idée de base est intéressante mais mérite d'être analysée davantage. Des essais réalisés avec une version hybride ACS classique et ACS – R appliquée à chaque cycle de manière alternée montrent qu'il est possible d'améliorer encore la qualité des solutions pour les instances de problème 601 et 605. Les deux formes de construction contribuent donc à la performance de l'algorithme. Il s'agit d'une piste intéressante à explorer dans des travaux futurs.

CHAPITRE 5

Conclusion

Le problème d'ordonnancement d'une machine unique avec temps de réglage dépendants de la séquence est un problème NP – difficile qui a été traité dans la littérature par plusieurs auteurs en utilisant différentes métaheuristiques. La minimisation du retard total est l'un des objectifs proposés pour ce type de problème et, parmi les métaheuristiques les plus efficaces pour le résoudre, on retrouve les algorithmes d'optimisation par colonie de fourmis, plus particulièrement, les ACS.

À ce jour, deux propositions d'ACS ont été présentées dans la littérature pour résoudre le problème d'ordonnancement d'une machine unique avec temps de réglage dépendants de la séquence pour minimiser le retard total. La première proposition utilise comme visibilité le temps de réglage normalisé entre les tâches et la marge normalisée des tâches, tandis que la deuxième proposition utilise comme visibilité l'information produite par la règle ATCS [Liao et Juan 2007]. Dans ce mémoire, la première proposition d'ACS sert de point de départ aux travaux réalisés.

Le premier objectif spécifique était d'exploiter un principe de construction à rebours dans la règle de transition. Pour le problème de machine unique avec temps de réglage dépendants de la séquence, ce principe de construction a été intégré à l'ACS proposé par Gagné, Price *et al.* [2002]. Pour amorcer une construction de solution à rebours de la séquence, l'estimation de la date de fin doit être établie en premier lieu. Cette estimation est ensuite utilisée par le nouveau concept de visibilité proposé. Celui-ci passe par la définition d'une marge arrière normalisée qui est utilisée pour la règle de transition et d'une nouvelle liste de candidats définie en fonction des caractéristiques du problème traité. Cette proposition d'ACS a été nommée ACS à rebours (ACS – R).

Deux modifications à l'ACS – R ont également été proposées. La première modification vise à rendre l'ACS – R plus intelligent au niveau du choix des paramètres utilisés par la règle de transition. Cette deuxième version a été nommée l'ACS – RD et le choix des paramètres ne se fait plus de façon empirique, évitant ainsi l'inconvénient de trouver les meilleurs paramètres pour chaque groupe d'instances de problème. Le principe utilisé consiste à démarrer l'algorithme avec un exposant plus important sur la marge normalisée que sur le réglage normalisé. Dès qu'aucune des commandes non placées n'est en retard, on inverse l'importance associée à chacune des visibilités. La deuxième proposition de modification, appelée ACS – RP, consiste à utiliser une règle de priorité pour compléter la séquence au moment où aucune des tâches restantes à placer n'est en retard. Cette règle consiste à positionner les tâches de façon à minimiser le temps de réglage des tâches non placées.

Ces trois propositions d'algorithmes à rebours, présentées dans le chapitre 3, permettent de répondre à notre premier objectif spécifique.

Le deuxième objectif spécifique consistait à évaluer la performance de ces ACS par rapport à l'ACS de Gagné, Price et al. [2002] sur les instances tests proposées par Ragatz (1989) et Gagné, Price *et al.* [2002]. Cet objectif spécifique a été atteint dans le chapitre 4 où on présente les résultats des expérimentations numériques et la comparaison de performance de l'ACS à rebours.

La performance des différentes versions de l'ACS – R a été comparée à la performance de l'ACS de Gagné, Price *et al.* [2002] reproduit pour les problèmes de petite taille et pour les problèmes de grande taille. Les résultats obtenus pour les instances de petite taille ont permis de conclure que les propositions d'ACS sont aussi performantes

que l'ACS classique au niveau de la qualité de solution, ce qui nous porte à croire que l'idée de base est intéressante. Pour les problèmes de grande taille, un avantage doit toutefois être accordé à l'ACS classique.

L'objectif général de ce mémoire consistait à contribuer à l'amélioration des méthodes de résolution pour le problème d'ordonnancement à machine unique avec temps de réglage dépendants de la séquence afin de favoriser l'avancement des connaissances dans le domaine. Ce travail de recherche a non seulement atteint son objectif, mais a aussi ouvert la voie à d'autres avenues non seulement pour l'amélioration de l'ACS – R mais aussi pour la conception d'algorithmes d'optimisation par colonie de fourmis ne faisant pas appel à la recherche locale.

Suite aux résultats obtenus, un premier constat s'impose. Les algorithmes d'optimisation par colonie de fourmis présentent généralement un manque au niveau de la diversification. Dans des travaux futurs, il serait intéressant d'essayer de remédier à cette lacune. Une possibilité consiste à réaliser la construction à rebours des solutions par blocs de tâches au lieu de placer une tâche à la fois. Cette technique, inspirée de l'Or–Opt (Or 1976), consiste à utiliser des blocs de différentes tailles associés à une règle probabiliste qui définit l'emplacement du bloc dans une séquence en construction. Une autre possibilité, inspirée de la méthode de bruitage [Charon et Hudry 2002], consiste à bruiter les données ou à ignorer certaines données au moment de l'application de la règle de transition. Le bruitage des données consiste à ajouter du bruit aux différents types de visibilités présentes dans la règle de transition tandis que dans l'autre cas, certaines informations ne seront pas prises en compte par la règle de transition.

D'autres améliorations pourraient également être apportées sur la base des travaux de l'ACS – RD en intégrant un paramétrage personnalisé pour chaque fourmi, ce qui favoriserait la diversification des solutions.

Les travaux réalisés dans ce mémoire ainsi que les pistes de recherches futures proposées ne s'adressent pas seulement au problème étudié. Ainsi, les concepts proposés dans ce mémoire peuvent s'appliquer à tous les problèmes d'ordonnancement qui visent à optimiser l'objectif du retard total.

CHAPITRE 6

Bibliographie

Allahverdi, A., J. N. D. Gupta, et al. (1999). "A review of scheduling research involving setup considerations." Omega **27**(2): 219-239.

Armentano, V. A. and R. Mazzini (2000). A genetic algorithm for scheduling on a single machine with set-up times and due dates. Production Planning & Control, Taylor & Francis Ltd. **11:** 713-720.

Baker, K. R. (1977). "Computational experience with sequencing algorithme adapted to the tardiness problem." AIIE Transaction **9**: 32-35.

Baker, K. R. (1992). Elements of sequencing and scheduling. Hanover, NH, Amos Tuck School of Business Administration.

Baker, K. R. and H. L. W. Nuttle (1980). "Sequencing independent jobs with a single resource." Naval Research Logistics **Q27**: 499-510.

Baptiste, P., V. Giard, et al. (2005). Gestion de production et ressources humaines. Montrál, Presses internationales Polytchnique.

Bigras, L.-P., M. Gamache, et al. (2008). "The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times." Discrete Optimization **5**(4): 685-699.

Bonabeau, E., M. Dorigo, et al. (1999). Swarm intelligence from natural to artificial systems. New York, Oxford University Press.

Bullnheimer, B., R. F. Hartl, et al. (1999). An improved Ant System algorithm for the Vehicle Routing Problem. Annals of Operations Research, Springer Science & Business Media B.V. **89:** 319-328.

Charon, I. and O. Hudry (2002). Les méthodes de britage. Optimisation approchée en recherche opérationnelle. Lavoisier. Paris, Hèrmes Science**:** 101-128.

Colorni, A., M. Dorigo, et al. (1991). Distributed optimization by ant-colonies. European Conference on Artificial Live (ECAL'91), Cambridge , MASS, USA, MIT Pres.

Conway, R. W., W. L. Maxwell, et al. (1967). Theory of Scheduling. Mass, Addison-Wesley.

Dorigo, M. (1992). Optimization, learning and natural algorithms. Milano, Politecnico di Milano. **PhD**.

Dorigo, M. and L. M. Gambardella (1997). "Ant colonies for the travelling salesman problem." Biosystems **43**(2): 73-81.

Dorigo, M., V. Maniezzo, et al. (1991). Positive feedback as a search strategy. P. d. Milano. Milano, Politecnico di Milano**:** 20.

Dorigo, M., V. Maniezzo, et al. (1996). "Ant system: optimization by a colony of cooperating agents." Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions **26**(1): 29-41.

Dorigo, M. and T. Stützel (2004). Ant Colony Optimization. Cambridge, MA, MIT Press.

Emmons, H. (1969). "One-machine sequencing problem with dual criteria." Operational Research society **17**: 701-715.

Eren, T. and E. Güner (2006). "A bicriteria scheduling with sequence-dependent setup times." Applied Mathematics and Computation **179**(1): 378-385.

Esquirol, P. and P. Lopez (1999). L'Ordonnancement. Paris, Econimica.

Florian, M., P. Trepant, et al. (1971). "An implicit enumeration algorithm for the machine sequencing problem." Management Science **17**: B782-B792.

França, P. M., A. Mendes, et al. (2001). "A memetic algorithm for the total tardiness single machine scheduling problem." European Journal of Operational Research **132**(1): 224-242.

Gagné, C., M. Gravel, et al. (2005). "Using metaheuristic compromise programming for the solution of multiple-objective scheduling problems." Journal of the Operational Research Society **56**: 687-698.

Gagné, C., W. L. Price, et al. (2002). "Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times." Operational Research society **53**: 895-906.

Gilmore, P. C. and R. E. Gomory (1964). "Sequencing a one state-variable machine: a solvable case of thr travelling saleman problem." Operational Research society **23**: 312-324.

Graham, R. L., E. L. Lawler, et al. (1979). "Optimization and approximation in deterministic sequencing and scheduling: A survey." Annals of Discrete Mathematics **5**: 287-326.

Graves, G. H. and C.-Y. Lee (1999). "Scheduling Maintenance and Semiresumable Jobs on a Single Machine." Naval Research Logistics **46**(7): 845-863.

Gupta, A. K. and A. I. Sivakumar (2005). "Multi-objective scheduling of two-job families on a single machine." Omega **33**(5): 399-405.

Gupta, S. R. and J. S. Smith (2006). "Algorithms for single machine total tardiness scheduling with sequence dependent setups." European Journal of Operational Research **175**(2): 722-739.

Jackson, J. R. (1955). "Scheduling a production line to minimize maximum tardiness." Management Science

Res. Project

UCLA.

Lee, Y. H., K. Bhaskaran, et al. (1997). "A heuristic to minimize the total weighted tardiness with sequence-dependent setups." IIE transactions **29**(1): 45-52.

Lenstra, J. K. and K. Rinnooy (1980). "Complexity results for scheduling chain on a single machine " European Journal of Operational Research **4**: 270-275.

Liao, C.-J. and H.-C. Juan (2007). "An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups." Computers Ops. Res. **34**: 1899-1909.

Lin, S.-w. and K.-c. Ying (2008). "A hybrid approach for single-machine tardiness problems with sequence-dependent setup times." The Journal of the Operational Research Society **59**(8): 1109-1119.

Lopez, P. and F. Roubellat (2001). Ordonnancement de la production. Paris, HERMES science Publications.

Maxwell, W. L. (1970). "On sequencing n jobs on one machine to minimize the number of late jobs." Management Science **16**: 295-297.

Moore, J. M. (1968). "An n job, one machine sequencing algorithm for minimizing the number of late jobs." Management Science **15**: 102-109.

Or, I., 1951- (1976). Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking. Industrial Engineering and Management Sciences. Chicago, Northwestern University. **Ph. D.**

Pinedo, M. (2002). Scheduling : theory, algorithms, and systems. Upper Saddle, N.J., Prentice Hall.

Pinedo, M. and X. Chao (1998). Operations scheduling with applications in manufacturing and services. Boston, Mass., Irwin/McGraw-Hill.

Ragatz, G. L. (1989). Scheduling to minimize tardiness on a single machine with sequence dependent setup time. Working Paper. Department of Management, Michigan State University.

Ragatz, G. L. (1993). A branch-and-bound methode for minimum tardiness sequencing on a single processor with sequence dependent setup times Twenty-four annual meeting of the Decision Science Institute.

Rosenkrantz, D. J., R. E. Stearns, et al. (1974). Approximate algorithms for the traveling salesperson problem. 15th Annual Symposium on Switching and Automata Theory (SWAT 1974), Annual IEEE Symposium.

Rubin, P. A. and G. L. Ragatz (1995). "Scheduling in a sequence dependent setup environment with genetic seach." Computers Ops. Res. **22**(1): 85-99.

Smith, W. E. (1956). "Various optimizers for single-stage production." Naval Research Logistics **Q.3**: 59-66.

Srinivasan, V. (1971). "A hybrid algorithm for the one machine sequencing problem to minimise total tardiness." Naval Research Logistics **Q. 18**: 317-327.

Stützle, T. and H. Hoos (2000). "Max-Min Ant System." Future Generation Computer Systems **16**(8): 889-914.

Tan, k.-C., R. Narasimhan, et al. (2000). "A comparison of four methods for minimizing total tardiness on a single processor with sequence dependent setup times." Omega **28**: 313-326.

Tan, K. C. and R. Narasimhan (1997). "Minimizing tardiness on a single processor with sequence-dependent setup times: a simulated annealing approach." Omega **25**(6): 619-634.