TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

CHAPITRE 1

INTRODUCTION

Un projet est un ensemble fini d'activités entreprises dans le but de répondre à un besoin précis, dans des délais fixés et dans la limite de l'enveloppe budgétaire allouée. C'est donc une action temporaire avec un début et une fin, qui mobilise des ressources identifiées (humaines, matérielles, informationnelles et financières) durant sa réalisation, qui possède un coût et fait donc l'objet d'une budgétisation de moyens et d'un bilan indépendant de celui de l'entreprise qui le réalise. La gestion de projet est une démarche visant à structurer, assurer et optimiser le bon déroulement d'un projet suffisamment complexe pour devoir être planifié dans le temps et être budgétisé. Elle permet de maîtriser les risques afin d'atteindre le niveau de qualité souhaité au cours du projet.

L'un des problèmes les plus souvent rencontrés en gestion de projet consiste en la planification des différentes activités à accomplir. Celle-ci peut être effectuée plus ou moins facilement avec les outils familiers tels que les méthodes CPM (*Critical Path Method*), PERT (*Programm Evaluation and Review Technic*) et le diagramme de GANTT pour des projets de petite taille. Toutefois, ces méthodes ne prennent pas en compte la gestion des ressources qui, dans un projet réel, sont de capacité limitée. Il est donc utile de faire appel à des outils plus sophistiqués lorsque le nombre d'activités à exécuter au sein du projet est important ou qu'il existe des contraintes sur les ressources. C'est dans ce cadre que se situe le problème d'ordonnancement de projet sous contraintes de ressources, encore appelé RCPSP (*Ressource Constrained Project Scheduling Problem*).

Le RCPSP consiste en l'ordonnancement d'un ensemble donné de tâches ou activités, à l'aide d'une ou de plusieurs ressources renouvelables ou non, en quantité limitée. La résolution du RCPSP a pour but la détermination des dates d'exécution des

activités en fonction de la disponibilité des ressources et de façon à satisfaire les objectifs fixés. Ces objectifs peuvent consister, par exemple, en l'optimisation de la durée totale du projet, de la date d'achèvement ou de l'utilisation des ressources. Le RCPSP a plusieurs applications dans le domaine industriel comme, par exemple, l'ordonnancement des processus en informatique distribuée, les problèmes de découpe, les problèmes courants d'ordonnancement d'ateliers. On l'utilise aussi dans le domaine organisationnel pour modéliser certains problèmes à savoir, les problèmes d'emploi du temps et les problèmes de planification de personnel.

Depuis plus de 30 ans, de nombreuses études ont été faites sur le RCPSP. Celui-ci est en quelque sorte une généralisation des problèmes classiques d'ordonnancement comme le *flow-shop*, le *flow-shop* hybride et le *job-shop*. Le RCPSP est un problème d'optimisation NP-difficile en raison de sa nature fortement combinatoire [Baptiste *et al.* 2005]. Plusieurs méthodes ont été utilisées dans la littérature pour adresser ce problème. Ces méthodes regroupent aussi bien les méthodes exactes que les méthodes approchées. Les méthodes exactes comprennent la programmation par contraintes [Baptiste *et al.* 2001], les méthodes par séparation et évaluation [Damay et Sanlaville 2006], les schémas de branchement [Demeulemeester et Herroelen 2002], les bornes inférieures [Carlier et Latapie 1991] et la programmation linéaire [Mingozzi *et al.* 1998; Pritsker *et al.* 1969]. Quant aux méthodes approchées, elles sont constituées des heuristiques telles que les algorithmes série et parallèle [Kolisch 1996b], les règles de priorité [Kolisch 1996a] et les métaheuristiques. Ces dernières incluent aussi bien des méthodes de recherche locale telles que la recherche avec tabous [Pinson *et al.* 1994] et le recuit simulé [Bouleimen et Lecocq

2003], que des méthodes évolutives comme les algorithmes génétiques [Hartmann 2002], l'optimisation par essaims particulaires [Zhang *et al.* 2005], les algorithmes de colonies de fourmis [Merkle *et al.* 2002] et la recherche par dispersion (*Scatter Search*) [Debels *et al.* 2006]. Les algorithmes génétiques demeurent l'une des meilleures méthodes de résolution du RCPSP selon les mesures de performance effectuées par Kolisch et Hartmann [2000; 2006].

Par sa nature fortement combinatoire, le RCPSP a fait l'objet de nombreux travaux et il possède un large champ d'applications industrielles [Demassey 2003]. En informatique distribuée, l'ordonnancement des processus consiste à exécuter des processus (les activités) sur des machines parallèles ou multiprocesseurs (les ressources) en gérant les exclusions mutuelles et la synchronisation. Dans le domaine des textiles ou de la métallurgie, les problèmes de découpe consistent à débiter de larges rouleaux de taille standard R en rouleaux de moindres largeurs de façon à minimiser le nombre de rouleaux initiaux. Ce problème se modélise en un RCPSP avec une seule ressource de capacité R et où, chaque rouleau à découper correspond à une activité de durée unitaire et de consommation égale à la largeur du rouleau. On peut citer aussi les problèmes courants d'ordonnancement d'ateliers (*job-shop*, *flow-shop* et *open-shop*) [Allahverdi *et al.* 2008] où des travaux, décomposés en séquences d'opération, doivent être exécutées par des machines disjonctives n'exécutant qu'une opération à la fois, chaque opération ayant sa machine dédiée. Tous les problèmes d'ateliers sont des cas particuliers du RCPSP, où les activités sont les opérations et la ressource est unitaire et correspond à une machine. La consommation d'une activité est de *1* sur la ressource dédiée correspondante et de *0* sur toutes les autres ressources. Vignier

et al. [1999] proposent une version hybride du problème d'atelier à cheminement unique (*flow-shop*) qui sert à modéliser de nombreux cas industriels. La version classique est constituée d'un ensemble de travaux possédant tous le même nombre d'opérations traitées dans le même ordre. La version hybride étend le problème d'atelier à cheminement unique en permettant, qu'à chaque étape, les opérations soient exécutées sur l'une des machines identiques disponibles. C'est à tout point de vue assimilable à un problème d'ordonnancement de projet où, à chaque étape correspond une ressource d'une capacité donnée, et où les activités sont les opérations. Ces activités sont de consommation unitaire sur la ressource requise, nulle sur les autres, et liées par les contraintes de préséance en chaînes. Une définition équivalente existe pour le problème d'atelier à cheminements multiples (*job-shop*).

Le RCPSP a aussi des applications dans le domaine organisationnel comme, par exemple, les problèmes d'emploi du temps (planification d'horaires) [Baptiste *et al.* 2005]. Un exemple simple d'emploi du temps est celui de l'université, où il est question de répartir les cours par enseignants, par salles et par créneaux horaires. Le problème se modélise en un RCPSP, pratiquement de la même manière qu'un problème d'atelier, mais où les ressources (salles et enseignants) ont des capacités dépendantes du temps, les enseignants n'étant disponibles qu'à certaines périodes.

Le RCPSP possède plusieurs variantes basées sur le modèle classique (SMRCPSP ou *Single-mode RCPSP*). On distingue entre autres, le GRCPSP (*Generalized RCPSP*), le PRCPSP (*Preemptive RCPSP*), le MMRCPSP (*Multi-Mode RCPSP*) et le RCPSPR (*RCPSP with Rework*).

Le GRCPSP permet une meilleure modélisation du problème étudié. En effet, les contraintes de préséance incluses dans le modèle du GRCPSP sont plus précises. Le modèle classique contient uniquement des relations de préséance du type *finish-to-start (FS)*. Quant au GRCPSP, il intègre également des contraintes de préséance dites généralisées *(GPR, Generalized Precedence Relationships)* : *start-to-start (SS)*, *finish-to-finish (FF)* et *start-to-finish (SF)*. Dans ce modèle, une activité peut partiellement s'exécuter en même temps que l'un de ces prédécesseurs. Ce modèle inclut également des contraintes de ressources généralisées : ressources partiellement renouvelables, ressources continuellement divisibles, ressources dédiées, ressources dont la disponibilité est variable. Pour résoudre le GRCPSP, Sampson et Weiss [1993] ont utilisé des techniques de recherche locale ainsi que Klein [2000] qui a utilisé la recherche avec tabous, avec des règles de priorité. Demeulemeester et Herroelen [1997], quant à eux, ont préféré utiliser les schémas de branchement.

Le PRCPSP autorise la préemption des activités. Le temps d'exécution d'une activité est divisé en plusieurs périodes (l'activité est divisée en plusieurs sous-activités) au sein desquelles il n'y a pas de préemption. L'objectif est alors de minimiser la date de début au plus tôt de la dernière opération de la dernière activité. Des contraintes de préséance doivent être rajoutées entre chaque sous-activité, contraintes de type *FS*. Les problèmes d'ateliers, dans lesquels une opération peut être divisée en plusieurs sous-activités effectuées sur la même machine, sont des exemples de PRCPSP. Vanhoucke et Debels [2008] présentent l'état de l'art des procédures utilisées pour la résolution du PRCPSP.

Demeulemeester et Herroelen [1996] ont aussi résolu le problème en utilisant les schémas de branchement.

Le MMRCPSP intègre la possibilité d'avoir plusieurs modes opératoires par activité, à l'opposé du SMRCPSP qui oblige chaque activité à s'exécuter dans un unique mode opératoire. La notion de mode d'exécution permet de spécifier les diverses façons avec lesquelles une activité utilise une ou plusieurs ressources. Par exemple, une activité qui nécessite une seule unité de ressource pour être complétée en 8 jours (mode 1), peut être exécutée en 4 jours avec deux unités de ressource (mode 2). On peut y retrouver à la fois le GRCPSP et le PRCPSP. Kolisch et Sprecher [1997] présentent une synthèse des méthodes utilisées aussi bien pour le SMRCPSP que pour le MMRCPSP. Selon les auteurs, les méthodes les plus utilisées sont la programmation linéaire et les méthodes par séparation et évaluation (*Branch-and-Bound*) pour les instances de petite taille. Pour des instances de plus grande taille, les méthodes privilégiées sont les heuristiques et les algorithmes stochastiques.

Le RCPSPR consiste à tenir compte des incertitudes dans l'ordonnancement de projet sous contraintes de ressources. La plupart des méthodes de résolution du RCPSP, partent du principe que le travail à faire est fait correctement sans qu'il ne soit nécessaire de revenir là-dessus. Il est supposé aussi que les conditions de travail sont idéales et que les délais sont respectés. Malheureusement, tel n'est pas souvent le cas en gestion de projet où la gestion de la qualité prend une part importante [Tukel et Rom 1998]. Il est souvent nécessaire pour une raison ou une autre, de reprendre une ou plusieurs activités au cours d'un projet, ou bien d'en prolonger la durée. Des retards de livraison ou l'indisponibilité de

certaines ressources peuvent aussi être à la base d'un non-respect de l'ordonnancement initial. Cette variante du problème se retrouve très peu dans la littérature. Tukel et Rom [1997] démontrent qu'il est plus intéressant de tenir compte au cours de l'ordonnancement, de la possibilité de reprendre une ou plusieurs activités au cours du projet. Ils utilisent la routine d'optimisation OSL (*Optimization Subroutine Library*) d'IBM [Wilson et Rudin 1992] pour résoudre le problème. Toutefois, selon Bean et Hadj-Alouane [1993], cette méthode n'est pas efficace pour résoudre les problèmes d'optimisation lorsque le nombre de variables est important (plus de 50 variables). Le terme 'robuste' est utilisé pour qualifier un algorithme qui prend en compte les incertitudes au cours de l'ordonnancement.

Ce mémoire traite plus particulièrement du RCPSP classique avec minimisation de la durée totale du projet. Un algorithme d'optimisation par essaims particulaires est proposé pour le résoudre. Une nouvelle méthode de croisement pour les algorithmes génétiques est aussi conçue en se basant sur la méthode d'optimisation par essaims particulaires. Le contenu du mémoire est organisé en cinq chapitres.

Dans le Chapitre 2, le RCPSP est présenté de façon plus détaillée. Un état de l'art est aussi effectué quant aux méthodes utilisées dans la littérature pour le solutionner.

Le Chapitre 3 est consacré aux méthodes utilisées dans ce mémoire pour résoudre le RCPSP, en l'occurrence les algorithmes génétiques et l'optimisation par essaims particulaires. La structure et les différents opérateurs qui composent ces deux méthodes sont décrits. Diverses méthodes de sélection, de croisement et de mutation sont présentées pour les algorithmes génétiques.

Dans le Chapitre 4, un algorithme d'optimisation par essaims particulaires est proposé. Cet algorithme se base sur des concepts empruntés à un autre algorithme d'optimisation par essaims particulaires conçu pour le problème de minimisation du retard total pondéré sur une machine unique. La gestion des contraintes de préséance a été intégrée à l'algorithme et certains paramètres ont été modifiés. Une reproduction d'un algorithme génétique de la littérature est aussi effectuée dans ce chapitre.

Dans le Chapitre 5, les algorithmes génétiques sont hybridés avec la méthode d'optimisation par essaims particulaires. Une forme particulière d'hybridation est utilisée. Dans un premier temps, le concept de l'optimisation par essaims particulaires sert à la conception d'une méthode de croisement pour l'algorithme génétique. Ensuite, cette méthode de croisement est utilisée en association avec une autre méthode de croisement au sein de l'algorithme afin d'améliorer les résultats obtenus.

Ce travail de recherche permet de mettre en évidence l'intérêt que pourrait constituer la méthode émergente que constitue l'optimisation par essaims particulaires, initialement conçue pour l'optimisation continue, pour les problèmes à variables discrètes et plus particulièrement pour le RCPSP. Dans la conclusion de ce mémoire, le bilan du travail accompli est effectué et de futures avenues de recherche sont aussi identifiées.

CHAPITRE 2

LE PROBLEME D'ORDONNANCEMENT DE PROJET SOUS CONTRAINTES DE RESSOURCES DANS LA LITTÉRATURE

2.1 Introduction

Le RCPSP classique, encore appelé SMRCPSP (*Single-mode RCPSP*), consiste en l'ordonnancement d'un ensemble donné d'activités, sur une ou plusieurs ressources renouvelables ou non, en quantité limitée, dans le but d'optimiser un objectif donné. Les objectifs les plus souvent rencontrés sont la minimisation de la durée du projet ou de sa date de fin (*makespan*), la minimisation du coût total du projet et la maximisation de la NPV (*Net Present Value*).

Les activités d'un projet sont non-interruptibles, i.e. qu'une activité commencée ne doit pas être interrompue tant qu'on ne l'a pas terminée. Chaque activité nécessite une quantité donnée de chacune des ressources tout au long de son exécution. Les activités sont liées entre elles par deux sortes de contraintes : les contraintes de préséance et les contraintes de ressources. Au niveau des contraintes de préséance, une activité j ne peut pas commencer tant que l'activité i n'est pas finie; on dit que l'activité i précède l'activité j. Quant aux contraintes de ressources, à tout instant et pour chacune des ressources, la somme des besoins des activités en cours ne doit pas dépasser la capacité de la ressource. Les ressources d'un projet sont dites cumulatives, c'est-à-dire qu'il est possible d'exécuter plusieurs activités de façon simultanée. De même, il est possible d'utiliser plusieurs ressources simultanément pour l'exécution d'une activité. Les ressources peuvent être soumises à d'autres contraintes telles que la disponibilité des ressources non stockables seulement sur une période bien déterminée (heures de travail du personnel ou d'équipement); ces ressources seront perdues si elles ne sont pas utilisées au cours de la période concernée. Il existe aussi des contraintes de localisation temporelle qui peuvent

indiquer qu'une activité ne peut débuter avant une date précise, en raison d'une indisponibilité de la ressource correspondante.

2.2 La représentation du RCPSP dans la littérature

Plusieurs notations sont utilisées dans le RCPSP. Celles que nous utilisons dans ce document ne proviennent pas d'un auteur en particulier, mais constituent une synthèse des notations que nous avons rencontrées.

Soient I le nombre de activités, K le nombre de ressources, d_i la durée de l'activité i. $r_{i,k}$ la quantité de la ressource k nécessaire à l'exécution de l'activité i, R_k la quantité de la ressource k disponible et H l'horizon de planification, i.e. la durée maximale de l'ordonnancement. On ajoute au projet deux activités fictives 0 et $I+1$, 0 étant la première activité et $I+1$ la dernière activité du projet. Ces deux activités sont de durées nulles. Soit V la relation d'ordre partiel sur l'ensemble des activités, i.e. l'ensemble des couples (i, j) tels que l'activité i précède l'activité j. En désignant par S_i la date de début de l'exécution de l'activité i, le problème d'ordonnancement de projet sous contraintes de ressources consiste à déterminer un ordonnancement de durée minimale, soit le vecteur $S = (S_0, S_1, \ldots, S_{I+1})$ tel que S_{I+1} soit minimisée et que les contraintes de préséance et de ressources soient respectées. $X_{i,t}$ est une variable binaire qui sert à déterminer la date de fin ordonnancée t de la activité i.

Ces notations sont définies dans le Tableau 2.1 où l'on retrouve dans la première colonne les variables utilisées et, dans la seconde, leur description.

Tableau 2.1 : Définition des variables utilisées dans le RCPSP

Variables	Description
I	Nombre d'activités au sein du projet.
K	Nombre de ressources, $k = 1, \ldots K$.
d_i	Durée de l'activité i.
$r_{i,k}$	Quantité de la ressource k nécessaire à l'exécution de l'activité i.
R_k	Quantité de la ressource k disponible pour le projet.
V	Ensemble des couples (i, j), tels que l'activité i précède l'activité j.
H	Durée maximale de l'ordonnancement.
t	Instant de décision $t = 1, \ldots H$.
$X_{i,t}$	Variable binaire : égale à 1 si l'activité i se termine à la date t et 0 dans le cas contraire.
S_i	Date de début de l'exécution de l'activité i.

L'objectif principal du RCPSP est de minimiser la durée totale du projet (*makespan*). La modélisation mathématique du RCPSP est donnée ci-dessous. L'Équation 2.1 représente la fonction 'objectif' qui minimise la date de fin de la dernière activité du projet. Les Équations 2.2 à 2.5 traduisent les contraintes sur les activités et les ressources. L'Équation 2.2 traduit le fait qu'il existe une date de fin unique par activité au sein du projet. L'Équation 2.3 représente la contrainte selon laquelle la date de fin de toute activité est inférieure à l'horizon. L'Équation 2.4 montre la relation de préséance entre deux activités i et j. Enfin, l'Équation 2.5 traduit le respect des limites de ressources par les activités s'exécutant de façon simultanée.

$$f = Min\left(\sum_{t=0}^{H} t * X_{I+1,t}\right) \quad \text{où } f \text{ est la fonction 'objectif'} \qquad \textit{(Équation 2.1)}$$

$$\sum_{t=0}^{H} X_{i,t} = 1 \qquad \forall\, i \qquad \textit{(Équation 2.2)}$$

$$\sum_{t=0}^{H} t * X_{i,t} \leq H \qquad \forall\, i \qquad \textit{(Équation 2.3)}$$

$$\sum_{t=0}^{H} t * X_{i,t} + d_j - \sum_{t=0}^{H} t * X_{j,t} \leq 0 \qquad \forall\, (i,j) \in V \qquad \textit{(Équation 2.4)}$$

$$\sum_{i=1}^{I} \sum_{u=t}^{t+d_i-1} r_{i,k} X_{i,u} \leq R_k \qquad \forall\, k \text{ et } \forall\, t \leq H - d_i + 1 \qquad \textit{(Équation 2.5)}$$

On représente habituellement un projet par le graphe potentiels-tâches, un graphe orienté G, où les nœuds représentent les activités du projet et où à tout couple (i, j) de V est associé un arc $i \rightarrow j$ valué par la durée d_i de l'activité i. L'exemple de la Figure 2.1 représente le graphe potentiels-tâches pour un projet de 7 activités et 3 ressources k_1, k_2 et k_3 de capacités respectives R_1, R_2 et R_3. Les nombres dans les nœuds sont les numéros des activités. Les trois nombres à côté de chaque nœud représentent les durées des activités sur chacune des ressources. Chaque arc du graphe est valué par la durée de l'activité source. À côté se trouve un ordonnancement minimal proposé par Baptiste *et al.*[2005]; dans cet ordonnancement, l'activité 1 est ordonnancée à la date 0, 2 et 5 à la date 2, 3 à la date 5, 4 à la date 6, 6 à la date 7 et la date de début de l'activité 7 est la date 10.

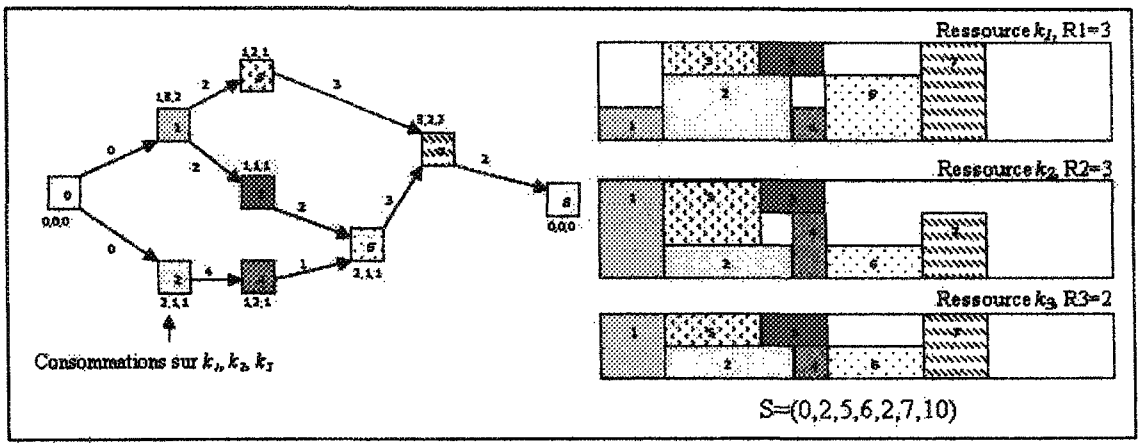


Figure 2.1 : Exemple de projet avec ordonnancement minimal [Baptiste *et al.* 2005]

Plusieurs méthodes ont été utilisées dans la littérature pour résoudre le RCPSP. Sans prétendre que la liste des travaux présentés à la prochaine section soit exhaustive, nous avons développé les principales approches de résolution pour le RCPSP.

2.3 Les méthodes de résolution du RCPSP

De nombreux auteurs ([Herroelen *et al.* 1998; 2001], [Hartmann et Kolisch 2000; 2006], [Demeulemeester et Herroelen 2002], [Artigues *et al.* 2008]) ont publié des études détaillées sur les diverses méthodes employées pour résoudre le RCPSP. Ces méthodes se subdivisent en deux groupes : les méthodes exactes et les méthodes approchées. Nous parlons ici des méthodes les plus usuelles.

2.3.1 Les méthodes exactes

Les méthodes exactes comprennent la programmation par contraintes [Baptiste *et al.* 2001], les méthodes par séparation et évaluation [Damay et Sanlaville 2006], les schémas de branchement [Demeulemeester et Herroelen 2002], les bornes inférieures [Carlier et Latapie 1991] et la programmation linéaire [Mingozzi *et al.* 1998; Pritsker *et al.* 1969].

Les techniques de programmation par contraintes se basent, d'une part, sur une relaxation successive du problème de RCPSP à chacune de ses ressources et, d'autre part, sur la relaxation des contraintes de préséance à des fenêtres temporelles afin d'établir des conditions nécessaires d'existence d'une solution de durée inférieure à H. Ces techniques ont été présentées dans la littérature par Brucker *et al.* [1998] pour l'*edge-finding* disjonctif, par Dorndorf *et al.* [2000] pour l'*edge-finding* disjonctif et le raisonnement énergétique,

Baptiste et Le Pape [1996] pour l'*edge-finding* cumulatif. Baptiste *et al.* [2001] ont mis à jour des dominances entre différentes règles d'ajustement applicables au RCPSP. Ils proposent aussi une étude expérimentale des règles plus complexes de l'*edge-finding* cumulatif et du raisonnement énergétique. La méthode de programmation par contraintes est présentée de façon détaillée par Baptiste *et al.* [2005]. Les auteurs présentent les modèles, les méthodes de résolution et les applications industrielles des divers problèmes de planification et d'ordonnancement.

La programmation linéaire et les autres méthodes exactes sont utilisées le plus souvent, conjointement avec les techniques de programmation par contraintes dans le but d'améliorer l'efficacité de ces algorithmes. Ces méthodes regroupent diverses notions telles que les bornes inférieures, la relaxation lagrangienne, les ressources redondantes, la génération des coupes, qui sont présentées par divers auteurs. Carlier et Néron [2003; 2007] ont largement parlé des bornes inférieures et des fonctions redondantes à travers leurs diverses publications. Dans sa thèse de doctorat, Demassey [2003] présente une revue de la littérature sur les méthodes hybrides de la programmation par contraintes et de la programmation linéaire pour la résolution exacte de problèmes combinatoires. L'auteure présente aussi deux nouvelles méthodes d'évaluation par défaut pour le RCPSP basées sur l'hybridation de la programmation par contraintes et de la programmation linéaire. La première est basée sur l'utilisation conjointe de techniques de propagation de contraintes et d'une relaxation lagrangienne originale du problème. La seconde repose sur la linéarisation des règles ou bien des inférences de la programmation par contraintes pour la génération de coupes ajoutées à des relaxations continues du RCPSP. Une méthode de résolution

optimale du RCPSP appelée « *resolution search* » a aussi été développée dans ce document.

2.3.2 Les méthodes approchées ou heuristiques

Les heuristiques sont des algorithmes de résolution d'un problème mathématique bien défini, par une approche intuitive qui se base sur l'interprétation et l'exploitation de la structure du problème afin d'en déterminer une solution raisonnable [Silver *et al.* 1980]. Elles comprennent, par exemple, les méthodes de construction progressive et les métaheuristiques.

Les méthodes de construction progressive d'une séquence d'activités en RCPSP sont constituées des algorithmes série et parallèle et des règles de priorité. Ces méthodes suivent le même principe et comprennent I étapes au cours desquelles une activité est ordonnancée à chaque étape. L'ordre de sélection des activités est déterminé par une règle de priorité. Les algorithmes série et parallèle prennent en entrée une séquence d'activités classées dans un ordre compatible avec les contraintes de préséance et leur assignent une date de début ou de fin. La différence fondamentale entre l'algorithme série et l'algorithme parallèle est que l'algorithme série sélectionne les activités dans l'ordre de la séquence et ordonnance l'activité le plus tôt possible en tenant compte des contraintes de préséance, de ressources, et des activités ordonnancées aux étapes précédentes. Quant à l'algorithme parallèle, il ordonnance les activités en incrémentant un instant de décision t initialisé à 0; à chaque étape, l'algorithme parcourt la séquence des activités non encore ordonnancées et place successivement au temps t les activités possibles sans violer les contraintes de

préséance ou de ressources. Les deux algorithmes génèrent des ordonnancements actifs, i.e. qu'on ne pourrait programmer aucune des activités plus tôt sans déplacer les autres activités. De plus, l'algorithme parallèle génère des ordonnancements sans délai, ce qui veut dire qu'à aucun moment une ressource n'est laissée inactive alors qu'elle pourrait démarrer l'exécution d'une activité. Il existe obligatoirement un ordonnancement actif optimal, et l'algorithme série est capable de le trouver si la "bonne" règle de priorité est utilisée pour établir la séquence en entrée. Par contre, rien n'indique qu'il peut exister un ordonnancement sans délai optimal; l'algorithme parallèle peut donc ne jamais trouver la meilleure solution quelque soit la séquence d'activités fournie en entrée [Hartmann et Kolisch 2000]. L'illustration de la Figure 2.2 montre le résultat obtenu par chacun des deux algorithmes pour l'exemple présenté à la Figure 2.1 avec la séquence d'activités (1, 2, 3, 4, 5, 6, 7).

L'algorithme série ordonnance les activités dans l'ordre de la séquence. Ainsi, après l'activité *3*, l'activité *4* est ordonnancée à la date *6*, le plus tôt possible sans violer les contraintes de préséance et de ressources et l'activité *5* suit à la date *7*. Quant à l'algorithme parallèle, il ordonnance à chaque instant *t*, dans l'ordre de la séquence, toutes les activités possibles sans violer les contraintes de préséance et de ressources. À la fin de l'activité *3*, l'activité *4* ne peut être ordonnancée car l'activité *2*, qui la précède, est encore en cours. Toutefois, les contraintes de préséance de l'activité *5* sont respectées et les ressources nécessaires pour son exécution sont disponibles. L'activité *5* est donc ordonnancée à la date *4* et l'activité *4* à la date *7*.

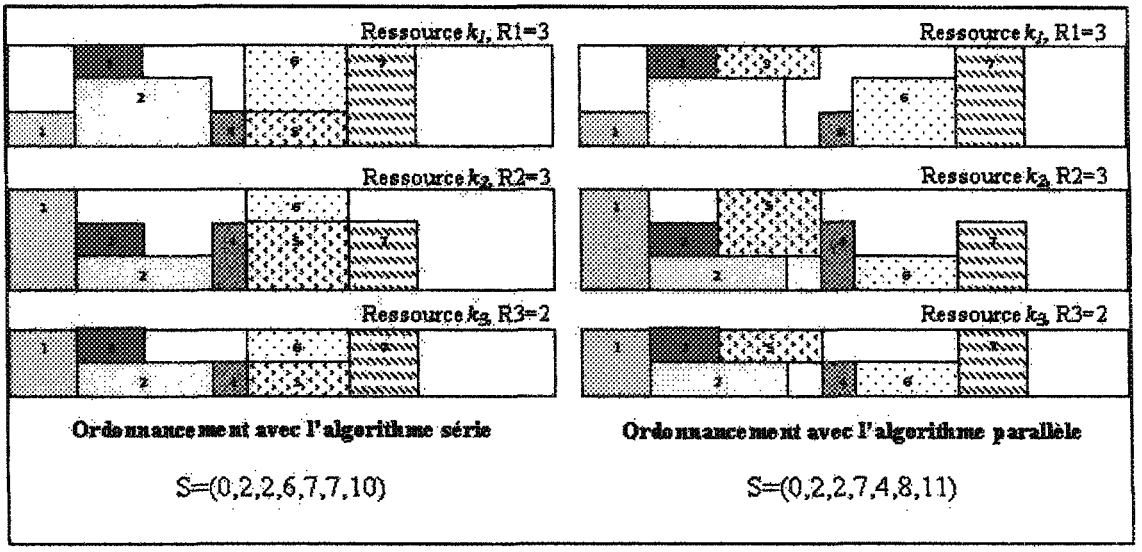


Figure 2.2 : Exemple d'ordonnancement avec l'algorithme série et l'algorithme parallèle

Concernant les heuristiques basées sur les règles de priorité, il existe des heuristiques simples, dites à passe unique, et des heuristiques à passes multiples. Une heuristique simple permet de déterminer l'ordre de sélection des activités par une règle de priorité unique. Par exemple, l'ordre croissant de la marge totale (délai entre le début au plus tôt et le début au plus tard de l'activité) permet d'ordonnancer les activités les plus urgentes d'abord. Les heuristiques à passes multiples consistent à lancer successivement l'algorithme série ou parallèle avec différentes séquences d'activités, en conservant à tout moment la meilleure solution obtenue. Différentes règles de priorité peuvent, par exemple, êtres utilisées pour construire les séquences. Il est aussi possible de modifier aléatoirement la séquence d'activités obtenue par une règle de priorité donnée. Avec cette dernière méthode, Kolisch et Hartmann [2000] montrent, par expérimentation sur des ensembles d'instances de test classiques du RCPSP, que les meilleurs résultats sont obtenus par la règle LFT (plus petite date de fin au plus tard) avec l'algorithme série pour les instances de

petite taille (*30* activités), et les règles LFT et WCS (plus petite marge restante si l'activité n'est pas sélectionnée) avec l'algorithme parallèle pour les plus grandes instances (*60* et *120* activités). Une autre méthode à passes multiples appelée ordonnancement avant/arrière (*forward-backward improvement*), consiste à exécuter alternativement l'algorithme série ou l'algorithme parallèle dans l'ordre normal puis, à l'envers en inversant la liste des activités et les arcs du graphe de préséance, et en ordonnançant les activités à partir de la fin du projet.

Une métaheuristique est un processus de génération itérative qui permet de guider une heuristique subordonnée par la combinaison de divers concepts tels que l'exploitation et l'exploration de l'espace de recherche, ainsi que des stratégies d'apprentissage qui sont utilisées pour structurer efficacement l'information afin de déterminer des solutions d'excellente qualité [Osman et Laporte 1996]. Les métaheuristiques sont en général non-déterministes et ne donnent aucune garantie d'optimalité mais ils peuvent faire usage de l'expérience accumulée durant la recherche de l'optimum, pour mieux guider la suite de la procédure de recherche. On distingue les métaheuristiques à solution unique parmi lesquelles on retrouve de nombreux algorithmes de recherche locale, et les métaheuristiques à base de population.

Les techniques de recherche locale associent itérativement un voisinage à chaque solution s_i de l'ensemble des solutions du problème d'ordonnancement, la première solution étant généralement générée par un algorithme constructif simple. Divers critères d'arrêt peuvent être considérés, tel qu'un temps limite imposé ou un écart acceptable par

rapport à une borne inférieure. La méthode de recherche locale la plus simple est la méthode de descente qui génère, à chaque itération, la solution voisine s_{i+1} qui minimise la fonction 'objectif' dans le voisinage de la solution s_i. L'algorithme s'arrête dès qu'il n'y a pas d'amélioration de la fonction 'objectif' entre deux itérations successives. Le principal défaut de la méthode de descente est son arrêt au premier minimum local rencontré. Divers algorithmes ont été proposés pour remédier à cette situation.

La technique du recuit simulé (*Simulated Annealing*) [Kirkpatrick *et al.* 1983], par exemple, choisit au hasard la solution s_{i+1} dans le voisinage de s_i. Cette solution voisine s_{i+1} est automatiquement acceptée comme nouvelle solution courante si elle est meilleure que s_i. Dans le cas contraire, s_{i+1} n'est acceptée comme nouvelle solution courante qu'avec une certaine probabilité. Si s_{i+1} est refusée, une nouvelle solution est choisie au hasard dans le voisinage de s_i, et ainsi de suite. La procédure peut s'arrêter, par exemple, lorsque la solution s_i n'a plus été modifiée depuis un certain temps. Jeffcoat et Bulfin [1993] utilisent cette méthode pour résoudre le RCPSP. Bouleimen et Lecocq [2003] utilisent la même méthode pour résoudre le RCPSP multi-mode. Valls *et al.* [2005] utilisent la méthode d'ordonnancement avant/arrière pour améliorer l'efficacité du recuit simulé et prouvent que cette technique permet d'obtenir de meilleurs résultats.

La recherche avec tabous (*Tabu Search*) [Glover 1986] est similaire à la méthode de descente, en ce sens que la solution s_{i+1} choisie dans le voisinage de s_i est la meilleure possible. Cependant, l'algorithme ne s'arrête pas lorsque le premier minimum local est atteint, le principe sous-jacent étant qu'il se peut que l'on doive détériorer la solution

courante pour pouvoir atteindre un optimum global. Il n'est pas impossible que s_i aussi soit la meilleure solution dans le voisinage de s_{i+1}, et l'algorithme pourrait alors boucler autour des solutions s_i et s_{i+1}. Pour contourner cette difficulté, la recherche avec tabous considère une liste de critères dits tabous, i.e. qu'il est interdit de choisir une solution répondant aux critères de cette liste pendant un certain nombre d'itérations. De ce fait, lorsque la recherche avec tabous se déplace d'une solution s_i vers une solution voisine s_{i+1}, le critère introduisant la différence entre s_i et s_{i+1} est placé dans la liste taboue, empêchant ainsi la boucle de se former. Ceci ne bloque pas seulement la solution s_i mais toutes les solutions présentant ce critère, ce qui constitue un léger handicap pour la recherche avec tabous. Cependant, une condition prioritaire appelée *condition d'aspiration* oblige l'algorithme à sélectionner une solution si elle est meilleure que toutes les solutions parcourues jusque-là, même si cette solution présente des critères de la liste taboue. Plusieurs auteurs ont utilisé la recherche avec tabous pour résoudre le RCPSP [Baar *et al.* 1998; Nonobe et Ibaraki 2002]. Thomas et Salhi [1998] introduisent une méthode où ils opèrent directement sur les ordonnancements au lieu de manipuler des listes d'activités. Klein [2000] développe une méthode réactive de recherche avec tabous pour le RCPSP avec des contraintes de ressources variables dans le temps. Gagnon [2002] propose une nouvelle règle de priorité ainsi qu'une adaptation de la recherche avec tabous pour résoudre le RCPSP sous contraintes de disponibilités variables de ressources. L'auteur propose également différentes stratégies de recherche avec tabous pour résoudre la variante multi-mode du RCPSP [Gagnon *et al.* 2005]. Pan *et al.* [2008] combinent d'autres méthodes d'intelligence

artificielle avec la recherche avec tabous et trouvent que cette technique donnent de meilleurs résultats que la méthode traditionnelle.

Contrairement à la recherche locale qui tente d'améliorer itérativement une solution courante, les méthodes à base de population, quant à elles, travaillent sur une population de solutions en appliquant un processus cyclique composé d'une phase de coopération et d'une phase d'adaptation individuelle qui se succèdent à tour de rôle. Dans la phase de coopération, les solutions de la population courante sont comparées entre elles, puis combinées, dans le but de produire de nouvelles solutions qui héritent des bons aspects de chaque membre de la population. Dans la phase d'adaptation individuelle, chaque solution dans la population peut évoluer de manière indépendante. On peut utiliser les mêmes types de critère d'arrêt que dans la recherche locale, ou alors on peut décider d'arrêter une métaheuristique à base de population dès que les solutions dans la population sont jugées trop similaires [Baptiste *et al.* 2005].

L'algorithme de colonies de fourmis (*Ant Colony Optimization*) a été proposé par Dorigo [1992] dans sa thèse de doctorat pour la recherche de chemins dans le problème du voyageur de commerce. L'auteur s'inspire de l'observation de l'exploitation des ressources alimentaires chez les fourmis. Une fourmi (l'éclaireuse) parcourt au hasard l'environnement autour de la colonie. Si celle-ci découvre une source de nourriture, elle rentre directement au nid, en laissant sur son chemin une piste de phéromones. Ces phéromones sont attractives et les fourmis passant à proximité ont tendance à suivre cette piste de façon systématique. En revenant au nid, ces mêmes fourmis vont renforcer la

piste en y déposant d'autres phéromones. Si deux pistes sont possibles pour atteindre la même source de nourriture, celle étant la plus courte sera parcourue par plus de fourmis que la longue piste. La piste courte est donc de plus en plus renforcée, et donc de plus en plus attractive. Les phéromones étant volatiles, la piste la plus longue finit par disparaître. Cette méthode a été utilisée dans la littérature pour la recherche de l'ordonnancement de moindre durée pour le problème du RCPSP [Merkle *et al.* 2002] ainsi que pour le GRCPSP [Shipeng *et al.* 2003]. Les auteurs concluent que l'algorithme de colonies de fourmis est parmi ceux qui performent le mieux pour la résolution des problèmes d'optimisation.

L'algorithme génétique (*Genetic Algorithm*) [Holland 1975], dénommé AG dans la suite de ce document, est un exemple particulier de méthode évolutive pour lequel l'adaptation individuelle est réalisée à l'aide d'un opérateur dit de mutation, alors que l'échange d'information est gouverné par un opérateur de reproduction et un opérateur de combinaison (*crossover*). Les AG ont attiré beaucoup de chercheurs du monde du RCPSP. Hartmann [1998] développe un AG qui utilise l'algorithme série avec la règle de priorité LFT pour l'ordonnancement des activités. L'auteur améliore ses résultats en réalisant un algorithme qui adapte l'algorithme série ou parallèle selon le problème traité [Hartmann 2002]. Alcaraz et Maroto [2001] combinent l'ordonnancement avant-arrière à l'algorithme série et obtiennent de meilleurs résultats avec les AG. Les auteurs associent plus tard l'algorithme parallèle et obtiennent de bien meilleurs résultats qu'ils améliorent un peu plus avec la recherche locale [Alcaraz et Maroto 2006]. Valls *et al.* [2008] combinent aussi la recherche locale aux AG pour en améliorer l'efficacité. Les travaux dans le domaine

continuent, les auteurs ne tarissent pas d'idées nouvelles sur le sujet [Chen et Weng 2009; Lova *et al.* 2009; Mendes *et al.* 2009].

L'optimisation par essaims particulaires (*Particle Swarm Optimization*) [Kennedy et Eberhart 1995], désigné par OEP dans la suite de ce document, s'inspire des comportements collectifs de déplacement chez certains groupes d'animaux tels que les oiseaux et les poissons. Que ce soit à la recherche de nourriture ou pour éviter leurs prédateurs, ces animaux se déplacent de façon relativement complexe mais cohérente alors que chaque individu n'a accès qu'à des informations limitées telles que la position et la vitesse de ses plus proches voisins [Dréo *et al.* 2003]. Chaque individu utilise donc l'information dont il dispose par rapport à sa propre histoire et à celle de ses plus proches voisins, afin de déterminer la direction qu'il doit prendre ainsi que sa vitesse de déplacement. L'OEP est similaire aux métaheuristiques précédentes dans le sens où elle part d'une population d'individus appelés particules, qu'elle essaie d'améliorer à chaque itération. La particule fait donc le choix à chacune des itérations de demeurer à sa position actuelle, d'aller vers sa meilleure position jamais connue ou vers celle de ses voisines appelées informatrices. L'OEP a été conçue initialement pour les problèmes d'optimisation continue. Cependant, plusieurs auteurs l'ont récemment adapté pour résoudre des problèmes discrets comme le RCPSP [Zhang *et al.* 2005]. Shan *et al.* [2007] hybrident cette méthode avec la technique d'optimisation par colonie de fourmis pour résoudre la version multi-mode du RCPSP. Les auteurs concluent, en général, que cette approche obtient de bonnes performances et qu'elle semble avoir un bel avenir dans le domaine de l'ordonnancement.

2.4 Conclusion

Dans ce chapitre, nous avons décrit le RCPSP ainsi que les méthodes les plus utilisées dans la littérature pour le résoudre. Ces méthodes sont nombreuses et comprennent aussi bien des méthodes exactes, pour la résolution des problèmes de petite taille, que des méthodes approchées. Parmi les méthodes approchées, les métaheuristiques se distinguent au sein des techniques qui performent le mieux pour la résolution du RCPSP.

Le chapitre qui suit présente un peu plus en détails les métaheuristiques utilisées dans le cadre de ce mémoire, notamment les AG et l'OEP.

CHAPITRE 3

MÉTAHEURISTIQUES : LES ALGORITHMES GÉNÉTIQUES ET L'OPTIMISATION PAR ESSAIMS PARTICULAIRES

3.1 Introduction

Les métaheuristiques ont pour but de déterminer en un temps raisonnable, une solution approchée à un problème d'optimisation lorsqu'il n'existe pas de méthode exacte pour le résoudre efficacement. Ces méthodes sont des techniques générales adaptables à chaque problème particulier et ont démontré leur efficacité dans plusieurs domaines [Baptiste *et al.* 2005]. Elles comprennent deux groupes d'approches qui sont les méthodes à solution unique et les méthodes à base de population. Ces dernières se basent sur une population d'individus qui interagissent entre eux afin de s'améliorer ou de produire de nouveaux individus plus performants. Il existe plusieurs métaheuristiques à base de population. Les métaheuristiques présentées dans ce chapitre sont celles qui ont fait l'objet de notre étude : les algorithmes génétiques et l'optimisation par essaims particulaires.

3.2 Les algorithmes génétiques (AG)

Les AG appartiennent à la famille des algorithmes évolutionnaires. Ces algorithmes ont la particularité de s'inspirer de l'évolution des espèces dans leur cadre naturel, d'après la notion de sélection naturelle développée par Charles Darwin au XIXe siècle et les méthodes de combinaison de gènes introduites par Gregor Mendel au XXe siècle. Ces méthodes consistent à faire évoluer une population dans le but d'en améliorer les individus. À chaque itération, un ensemble d'individus est mis en avant, générant ainsi un ensemble de solutions pour un problème donné.

Les AG sont issus des travaux de John Holland et de ses collègues et étudiants de l'université de Michigan qui ont commencé à s'y intéresser depuis 1960. Ces travaux ont

vu leur aboutissement en 1975 avec la publication par John Holland d'un livre intitulé *Adaptation in natural and artificial systems* [Holland 1975]. Quelque temps après, Goldberg sortait un livre dans lequel il résumait les divers domaines d'applications des AG tels que la recherche, l'optimisation et l'apprentissage-machine [Goldberg 1989]. Cette publication marque le début d'un intérêt scientifique croissant pour cette nouvelle technique d'optimisation. Gen et Cheng [1997] développent alors un peu plus l'application de ces algorithmes aux problèmes d'optimisation. Dans la même période, une librairie C++ appelée GAlib a été élaborée pour servir de support à la programmation à base d'AG. Cette librairie contient une panoplie d'outils conçus pour résoudre des problèmes d'optimisation à l'aide des AG.

3.2.1 Analogie avec la génétique

La génétique est une branche de la biologie qui étudie la transmission des caractères héréditaires chez les êtres vivants. Les AG étant inspirés de cette science, il importe de replacer les divers termes utilisés dans ce document, dans leur contexte originel afin de mieux en saisir le sens.

Les êtres vivants sont constitués de cellules. Chaque cellule contient des chromosomes. Le nombre de chromosomes contenu dans une cellule est spécifique à chaque espèce. À l'intérieur de chaque chromosome, se trouve l'information génétique de chaque être, conservée sous forme d'une molécule d'acide désoxyribonucléique communément appelée ADN. L'élément de base de ces chromosomes est un gène, qui est une séquence d'ADN. Sur chaque chromosome, il existe une suite de gènes constituant une

chaîne qui code les diverses caractéristiques de l'organisme (par exemple, la couleur des cheveux, la forme du nez). Chez une espèce donnée chaque gène possède, sur un chromosome, un emplacement fixe appelée le locus. Les différentes formes qui peuvent être prises par un même gène sont appelées des allèles. L'ensemble des gènes d'un individu est son génotype et son apparence physique constitue son phénotype.

Par analogie à la génétique, les AG travaillent sur une population d'individus et non sur un individu isolé. Ici, un individu est une solution potentielle au problème posé. Une population est donc un ensemble de solutions, plus ou moins performantes, pour ce problème. Chaque individu est représenté par un chromosome unique constituant son génotype. Une population constitue donc un ensemble de chromosomes. Un chromosome est composé d'une chaîne de symboles qui sont les gènes. Le phénotype est la représentation naturelle d'une solution au problème traité, obtenue après le décodage du génotype. La population évolue durant une succession d'itérations, appelées générations, jusqu'à ce qu'un critère d'arrêt, spécifié à l'avance, soit vérifié.

Selon Darwin [1859], l'évolution des êtres vivants dans un milieu donné repose sur la compétition. Les individus les mieux adaptés survivent, se reproduisent et transmettent à leurs descendants les caractères qui ont favorisé leur survie. Quant aux autres, ils finissent par disparaître. On finit donc au fil des générations par obtenir des individus de plus en plus forts et mieux adaptés. Il est rare, mais cependant possible, qu'il y ait au cours de la reproduction ou par un effet de la nature, une modification de la structure génétique d'un individu. Ce phénomène appelé dérive génétique ou mutation, se produit aléatoirement au

sein d'une population. Les AG traduisent ces trois caractéristiques par des opérateurs de sélection, de croisement et de mutation qu'ils utilisent pour générer des populations 'enfants' et former de nouvelles générations.

3.2.2 Le principe d'un algorithme génétique

Le schéma de la Figure 3.1 illustre la structure générale d'un AG. L'algorithme commence par générer une population initiale qui se compose d'un nombre déterminé d'individus. La meilleure solution peut ne pas se trouver dans cette population initiale. À chaque génération, une succession d'opérations de sélection, de croisement, de mutation, d'évaluation et de remplacement est appliquée aux individus de la population, afin de produire la génération suivante. Lorsque le critère d'arrêt est atteint, la meilleure solution trouvée est sélectionnée et l'algorithme s'arrête.

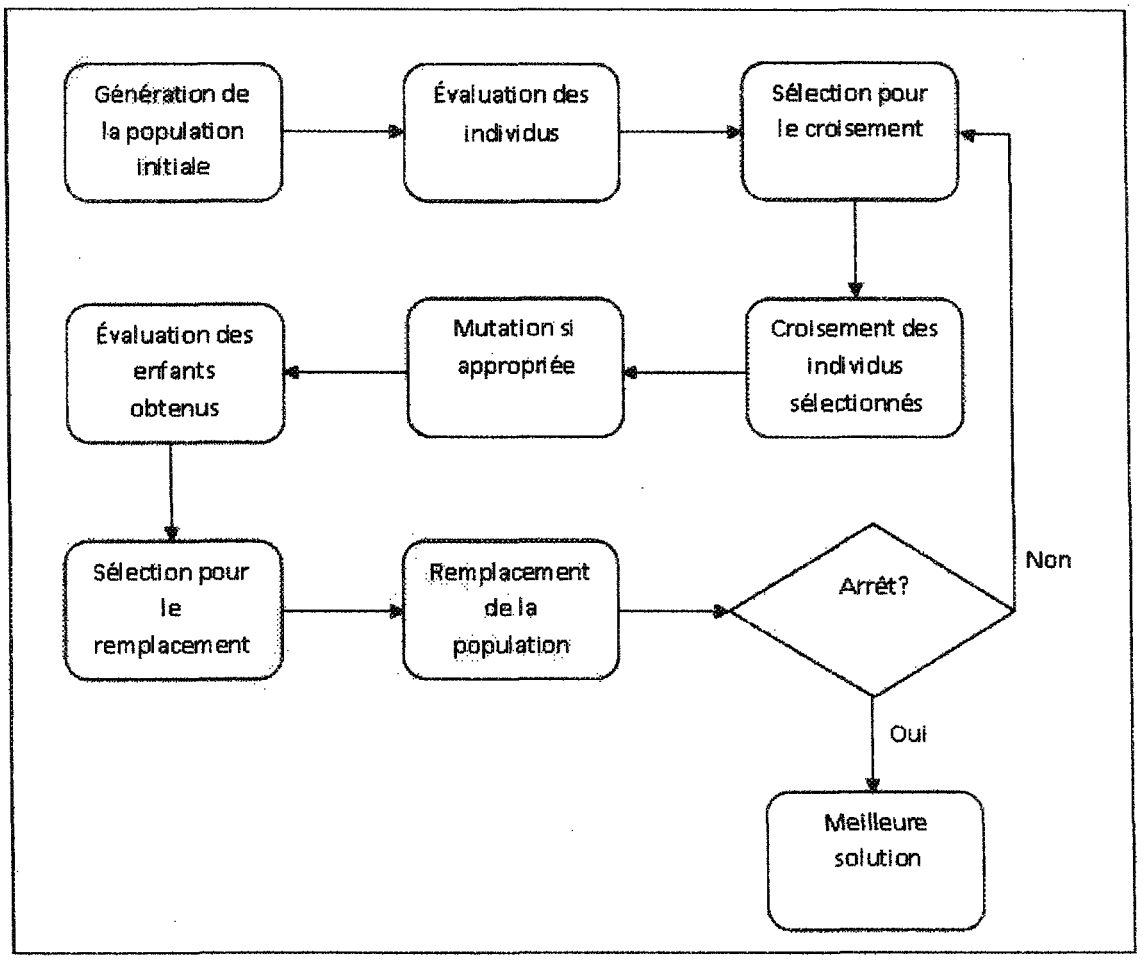


Figure 3.1 : Structure générale d'un algorithme génétique [Dréo *et al.* 2003]

L'objectif des AG est de faire évoluer une population afin de trouver le meilleur individu. En termes mathématiques, le but des AG est de déterminer l'extrémum d'une fonction f d'un espace de recherche E vers l'ensemble des réels :

$$f: E \rightarrow \mathbb{R}$$

Chaque élément de E est noté x_n. Une population P est donc un ensemble de N individus de l'espace E telle que $P= (x_1, x_2, \ldots, x_n, \ldots, x_N)$. La fonction f appelée fonction d'adaptation, fonction d'évaluation ou encore fonction *fitness*, sert comme son nom

l'indique, à mesurer l'adaptation des individus à leur environnement. Elle associe donc une valeur à chaque individu de la population, permettant ainsi aux solutions de pouvoir être comparées entre elles. Dans la suite du document, le terme 'fonction d'adaptation' est utilisé pour désigner la fonction *f*.

La fonction d'adaptation est propre à chaque type de problème. Elle devrait tenir compte de la représentation choisie et de la nature des opérateurs afin de pouvoir donner des indications non trompeuses sur la progression vers l'optimum. Cependant, dans le cas des problèmes industriels, l'évaluation de la fonction d'adaptation consomme de loin la plus grande part de la puissance de calcul durant une optimisation évolutionnaire [Dréo *et al.* 2003]. Il faudrait donc veiller à la simplifier autant que possible afin de réduire la puissance de calcul nécessaire.

3.2.3 La représentation des individus (encodage)

Généralement, les chromosomes sont représentés sous forme d'une chaîne de bits. Ce type de codage a pour intérêt de permettre de créer des opérateurs simples de croisement et de mutation. Cependant, ce type de codage ne convient pas pour modéliser tous les problèmes. C'est le cas, par exemple, du problème du voyageur de commerce (PVC). Ce problème concerne un commerçant qui doit parcourir n villes afin de livrer ses produits. Son objectif est de trouver l'ordre dans lequel il doit parcourir ces villes et retourner à son point de départ, afin d'avoir le trajet le plus court possible, sachant qu'il ne peut passer par une ville qu'une et une seule fois. Gen et Cheng [1997] proposent deux formes de représentation pour ce type de problème : la représentation par clés aléatoires et la

représentation par permutation. Ces deux représentations sont aussi celles qui sont retrouvées le plus souvent dans la littérature du RCPSP.

La représentation par clés aléatoires consiste à encoder la solution avec des nombres aléatoires générés dans l'intervalle [0, 1[. Chaque emplacement représente un numéro de ville et les nombres (ou clés) aléatoires représentent l'ordre de passage des villes. En ordonnant les clés de façon croissante, on retrouve l'ordre de passage de chaque ville. Pour un problème à cinq villes, on a, par exemple, le chromosome [0,24 0,14 0,78 0,18 0,32] qui représente l'ordre de passage suivant : [2 – 4 – 1 – 5 – 3]. Le voyageur part donc de la ville *2* et se rend à la ville *4* suivie respectivement des villes *1, 5* et *3*. Il retourne ensuite à la ville *2*.

La représentation par permutation est, selon les auteurs, la représentation la plus naturelle du PVC [Gen et Cheng 1997]. Elle consiste à énumérer les villes sous forme d'une liste, dans l'ordre dans lequel elles sont visitées. Un chromosome constitue donc une permutation des *N* villes parcourues. Dans le cas d'un problème à 5 villes, on peut avoir, par exemple, le chromosome [3 2 5 1 4] qui consiste à parcourir dans cet ordre, les villes *3, 2, 5, 1, 4* et à retourner à la ville *3*.

3.2.4 La génération de la population initiale

La population initiale est très importante car elle affecte non seulement la qualité de la solution, mais également le nombre de générations au bout duquel on obtient de bonnes solutions [Sait et Youssef 1999]. Le niveau de diversité présente au sein de la population

initiale représente également une caractéristique importante de l'algorithme. La génération de la population initiale concerne aussi bien la taille de la population que la manière dont elle est générée.

Habituellement, la population initiale est générée de manière aléatoire ou à l'aide d'une méthode de construction progressive. Cependant, on peut parfois partir de quelques solutions trouvées à l'aide de certaines heuristiques. Cette méthode connue sous le nom de *seeding* est, selon Davis [1991], plus efficace.

En ce qui concerne la taille de la population, elle agit non seulement sur le taux de convergence et la qualité des solutions obtenues, mais aussi sur la performance de l'algorithme. Une taille trop petite ne permet pas une bonne exploration de l'espace de recherche et peut conduire à une convergence prématurée de la population vers un optimum local. Par ailleurs, une taille trop grande peut abaisser exagérément le taux de convergence et altérer les performances de l'algorithme. Se basant sur ses propres expériences, Alander [1992] propose une valeur comprise entre l et $2l$ (l étant la longueur d'un chromosome) pour la taille de la population.

3.2.5 La sélection

À chaque génération de l'AG, deux sélections sont effectuées : la première pour la reproduction, et la seconde pour le remplacement. La sélection pour la reproduction consiste à choisir et à apparier les individus qui sont ensuite croisés pour donner de nouvelles solutions; il est possible de sélectionner plusieurs fois le même individu. Quant à

la sélection pour le remplacement, elle consiste à choisir les chromosomes qui vont constituer la génération suivante; chaque chromosome ne peut être sélectionné qu'au plus une fois. La capacité d'un chromosome à être sélectionné dépend à la fois de sa performance pour la fonction d'adaptation et de la méthode de sélection utilisée. Les chromosomes ayant les meilleures valeurs de fonction d'adaptation ont souvent plus de chance d'être reproduits que les autres et sont fréquemment choisis pour remplacer les moins bons. Il se pose cependant un problème lors de la sélection : le dilemme 'exploration vs exploitation'. Il s'agit alors de choisir entre utiliser les méthodes stochastiques afin de favoriser une diversification des solutions dans le système (exploration), versus effectuer une sélection davantage élitiste afin d'obtenir les meilleurs individus et améliorer la population (exploitation).

3.2.5.1 La sélection pour la reproduction

La sélection pour la reproduction peut s'effectuer de différentes façons. On distingue, par exemple, la sélection déterministe, la sélection proportionnelle et la sélection par tournoi [Dréo *et al.* 2003].

3.2.5.1.1 La sélection déterministe

La sélection déterministe consiste à ranger les individus dans l'ordre croissant (décroissant pour un problème de maximisation) de leur fonction d'adaptation et à sélectionner les meilleurs. Cette méthode est aussi appelée méthode d'élitisme car elle favorise les meilleurs individus. La sélection déterministe a une variance très faible et se base uniquement sur la valeur retournée par la fonction d'adaptation. Le fait de sélectionner

uniquement les meilleurs individus favorise une convergence prématurée de la population, ce qui est à éviter car l'algorithme est susceptible de s'arrêter sur un optimum local [Dréo *et al.* 2003].

3.2.5.1.2 La sélection proportionnelle

Deux techniques principales se sont imposées pour ce type de sélection : le tirage à roulette (Roulette Wheel Selection) introduite par Holland [1975] et la sélection universelle stochastique (*Stochastic Universal Sampling*) proposée par Baker [1985].

Le tirage à roulette

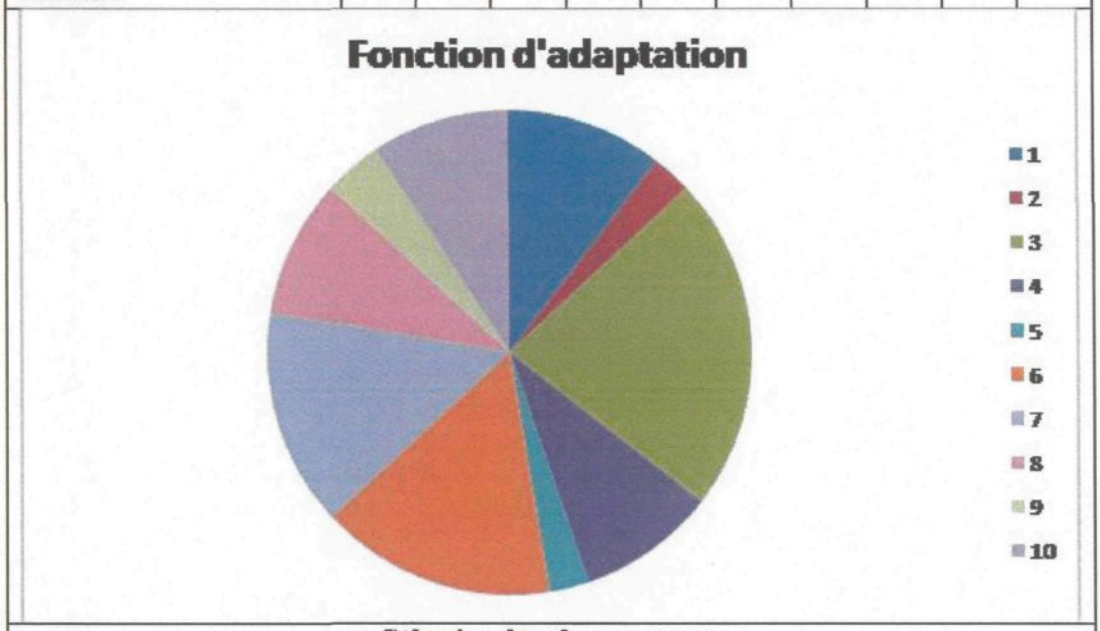
Le tirage à roulette est une méthode stochastique qui exploite la métaphore d'une roulette de casino. Elle consiste à associer à chaque individu un segment (ou case de la roue) dont la longueur est proportionnelle à sa performance. La roue étant lancée, l'individu sélectionné est celui sur lequel la roue s'est arrêtée. Cette méthode favorise les meilleurs individus, mais tous les individus conservent néanmoins des chances d'être sélectionnés.

Soit F la somme totale des fonctions d'adaptation des individus de la population. Le tirage à roulette consiste à sélectionner aléatoirement un nombre entre 0 et F. Pour chaque individu de la population, on fait la somme cumulée de sa fonction d'adaptation avec ceux des individus qui le précèdent. L'individu choisi est le premier pour lequel cette somme cumulée est supérieure ou égale au nombre aléatoire généré. La Figure 3.2 illustre un exemple de sélection par la méthode de la roulette tiré du livre de Davis [1991]. Cette figure est constituée en trois parties. La première partie est un tableau de trois lignes : dans la première se trouvent dix numéros de chromosomes; la deuxième contient leur fonction

d'adaptation individuelle et la troisième, les fonctions d'adaptation cumulées. La deuxième partie montre une roue divisée en dix tranches proportionnelles à la fonction d'adaptation de chaque individu : c'est la roue de la sélection. Dans la troisième partie se trouve un tableau constitué de deux lignes : la première contient des nombres générés aléatoirement et la seconde, le numéro des chromosomes correspondants. Le problème ci-dessous est un problème de maximisation. Plus la fonction d'adaptation d'un chromosome est grande, plus ce chromosome a donc de chance d'être sélectionné. On remarque que le meilleur individu (numéro *3*) a été sélectionné à trois reprises dans ce cas précis.

La méthode du tirage à roulette a une forte variance. Toutefois, il peut arriver que sur n sélections successives destinées à désigner les parents de la génération suivante, la majorité des chromosomes sélectionnés soient de mauvaise qualité selon leurs valeurs de la fonction d'adaptation. Cette occurrence peut réduire la performance de l'algorithme génétique, mais la probabilité que cela advienne reste très faible [Davis 1991]. Réciproquement, on peut aussi se retrouver face à une domination écrasante par un chromosome «localement supérieur». Ceci entraine une perte importante de la diversité de la population conduisant parfois à une convergence prématurée vers cet optimum local.

Chromosome	1	2	3	4	5	6	7	8	9	10
Fonction d'adaptation	8	2	17	7	2	12	11	7	3	7
Fonction d'adaptation cumulée	8	10	27	34	36	48	59	66	69	76



Sélection des chromosomes										
Nombre aléatoire	23	49	76	13	1	27	57	48	29	6
Chromosome sélectionné	3	7	10	3	1	3	7	6	4	1

Figure 3.2 : Exemple de sélection par la méthode du tirage à roulette [Davis 1991]

La sélection universelle stochastique

La sélection universelle stochastique consiste à choisir les individus qui participent au croisement en se basant sur la valeur attendue e_n d'un individu x_n. Cette valeur est calculée comme indiqué par l'Équation 3.1, où N est le nombre d'individus au sein de la population et f_n est la fonction d'adaptation de l'individu x_n.

$$e_n = \frac{f_n}{\sum_{j=1}^{N} f_j} * N \qquad \textit{(Équation 3.1)}$$

Cette méthode consiste à faire la sélection de façon proportionnelle à la fonction d'adaptation, mais avec une petite portion de génération aléatoire qui permet à chaque individu d'avoir une chance d'être sélectionné. Le principe est simple et est illustré par l'algorithme de la Figure 3.3. Au départ, un nombre est généré aléatoirement dans l'intervalle [0, 1[et conservé dans *Variable*. Pour chaque individu x_n de la population, *Cumul* est calculé en faisant la somme des valeurs attendues e_n des individus qui le précèdent et de la sienne. Tant que *Cumul* est supérieur à la valeur contenue dans *Variable*, l'individu courant x_n est sélectionné et *Variable* est incrémenté de *1*.

```
Début
Initialiser Cumul à 0
Initialiser aléatoirement Variable
Pour n allant de 1 à N Faire | N étant la taille de la population
    Cumul = Cumul + e_n
    Tant que Cumul > Variable Faire
        Sélectionner le chromosome x_n
        Variable = Variable + 1
    Fin Tant que
Fin Pour
Fin
```

Figure 3.3 : Algorithme de la sélection universelle stochastique [Gen et Cheng 1997]

La Figure 3.4 illustre un exemple de sélection universelle stochastique, se basant sur l'exemple de la Figure 3.2. La première partie de la figure est constituée d'un tableau de trois lignes contenant les chromosomes, leur fonction d'adaptation et leur valeur attendue e_n. La seconde partie est aussi un tableau de trois lignes qui montre le choix des chromosomes d'après le cumul des e_n et la valeur courante de *Variable* mentionnée plus

haut. La valeur aléatoire générée au départ est *0,5*. Il faut noter que seules les étapes auxquelles un chromosome est sélectionné sont explicitées dans cette figure.

Chromosome x_n	1	2	3	4	5	6	7	8	9	10
Fonction d'adaptation f_n	8	2	17	7	2	12	11	7	3	7
Valeur attendue e_n	1,05	0,26	2,24	0,92	0,26	1,58	1,45	0,92	0,39	0,92
Sélection des individus										
Cumul des e_n (*Cumul*)	1,05	3,55	3,55	3,55	4,73	6,31	7,76	7,76	8,68	9,99
Variable	0,5	1,5	2,5	3,5	4,5	5,5	6,5	7,5	8,5	9,5
Chromosome sélectionné	1	3	3	3	5	6	7	7	8	10

Figure 3.4 : Exemple de la sélection universelle stochastique

L'exemple présenté montre que plus forte est la fonction d'adaptation d'un chromosome, plus ce chromosome a de chances d'être sélectionné. Néanmoins, le chromosome *5* qui a une très faible fonction d'adaptation, a pu obtenir une chance d'être choisi de faire partie des individus croisés pour produire la nouvelle génération. Cette méthode, à l'instar du tirage à roulette, maintient davantage la diversité au sein de la population.

Deux autres méthodes de sélection stochastique basées sur la valeur attendue ont été présentées par Sait et Youssef [1999]. Il s'agit des méthodes de sélection stochastique avec et sans remplacement.

La **sélection stochastique avec remplacement** consiste à ranger les individus par ordre décroissant de leur valeur attendue e_n. Les meilleurs individus sont ensuite sélectionnés et une probabilité de sélection leur est attribuée en fonction de e_n. La Figure 3.5 illustre le principe de cette méthode en utilisant trois tableaux. Le premier tableau contient, dans sa première ligne les numéros des chromosomes, et leur fonction

d'adaptation dans la seconde. Dans le deuxième tableau, les chromosomes sont ordonnés suivant l'ordre décroissant de leur valeur attendue e_n. Enfin, dans le troisième tableau, les chromosomes sont sélectionnés à chaque étape et leur probabilité de sélection est spécifiée. Ainsi le chromosome *3* ayant une valeur attendue e_n de *2,24* est sélectionné deux fois avec une probabilité de *1,00* et une troisième fois avec une probabilité de *0,24*. Il en est de même pour les chromosomes suivants.

Les individus sont donc choisis proportionnellement à leur valeur attendue, et par conséquent, à leur fonction d'adaptation. Cette méthode est peu utilisée du fait de sa faible variance : seuls les individus ayant les meilleures fonctions d'adaptation ont la chance d'être sélectionnés, ce qui entraine une convergence très prématurée de la population.

Chromosome	1	2	3	4	5	6	7	8	9	10
Fonction d'adaptation	8	2	17	7	2	12	11	7	3	7
Rangement										
Valeur attendue	2,24	1,58	1,45	1,05	0,92	0,92	0,92	0,39	0,26	0,26
Ordre	3	6	7	1	4	8	10	9	2	5
Sélection des chromosomes										
Rang	1	2	3	4	5	6	7	8	9	10
Chromosome choisi	3	3	3	6	6	7	7	1	1	4
Probabilité	1,00	1,00	0,24	1,00	0,58	1,00	0,45	1,00	0,05	0,92

Figure 3.5 : Exemple de sélection par la sélection stochastique avec remplacement

La **sélection stochastique sans remplacement** part du même principe. La différence ici est que chaque chromosome peut avoir la chance d'être sélectionné. Certains chromosomes sont d'abord sélectionnés par la méthode précédente en utilisant uniquement la partie entière de leur valeur attendue. Les chromosomes restants sont ensuite sélectionnés par la méthode du tirage à roulette dans laquelle la probabilité de sélection de chaque

individu est proportionnelle à la partie décimale de sa valeur attendue. Cette méthode est illustrée à la Figure 3.6. Cette dernière est constituée de cinq parties. La première partie contient les numéros des chromosomes et leur fonction d'adaptation. Dans la deuxième partie, les chromosomes sont ordonnés suivant l'ordre décroissant de leurs valeurs attendues. Dans la troisième partie, les premiers individus sont sélectionnés, ceux dont la valeur attendue est supérieure ou égale à *1*. La quatrième partie de la figure montre le cumul des parties décimales des valeurs attendues de chaque chromosome. Enfin, dans la dernière partie, les individus restants sont sélectionnés en utilisant la méthode du tirage à roulette. Ainsi le chromosome *2* qui a une très faible valeur attendue, a pu être sélectionné pour aller au croisement.

Cette méthode associe le tirage à roulette à la sélection stochastique, permettant de ce fait à des individus plus faibles d'êtres sélectionnés tout en privilégiant les plus performants.

3.2.5.1.3 La sélection par tournoi

La sélection par tournoi [Brindle 1981] a pour principe la comparaison des individus d'un sous-groupe de q chromosomes ($q \geq 2$) de la population. Il s'agit de la *sélection q-tournoi*. Le chromosome ayant la plus forte valeur de fonction d'adaptation est sélectionné. Le tournoi est refait à chaque fois, avec ou sans remise, jusqu'à ce que le nombre d'individus nécessaires pour le croisement soit atteint. Le tournoi peut être déterministe ou stochastique. Dans le cas d'un tournoi stochastique, le meilleur individu du sous-groupe considéré est sélectionné avec une probabilité comprise entre 0,5 et 1. La

variance de cette méthode est forte, mais le choix de la taille q du sous-groupe permet de faire varier la pression sélective, i.e. les chances de sélectionner les individus ayant les meilleures valeurs de la fonction d'adaptation. Plus q est élevé, plus forte est la pression sélective et inversement. La méthode de sélection par tournoi est très utilisée du fait de sa simplicité de mise en œuvre.

Chromosomes	1	2	3	4	5	6	7	8	9	10
Fonction d'adaptation f_n	8	2	17	7	2	12	11	7	3	7
Rangement										
Valeur attendue e_n	2,24	1,58	1,45	1,05	0,92	0,92	0,92	0,39	0,26	0,26
Ordre	3	6	7	1	4	8	10	9	2	5
Sélection des chromosomes										
Rang	1	2	3	4	5	6	7	8	9	10
Chromosomes sélectionnés	3	3	6	7	1					
Calcul des cumuls des parties décimales										
Chromosomes	1	2	3	4	5	6	7	8	9	10
Parties décimales	5	26	24	92	26	58	45	92	39	92
Valeurs cumulées	5	31	55	147	173	231	276	368	407	499
Sélection des chromosomes restants										
Rang	1	2	3	4	5	6	7	8	9	10
Nombre aléatoire						305	24	240	133	49
Chromosomes sélectionnés						8	2	7	4	3

Figure 3.6 : Exemple de sélection par la sélection stochastique sans remplacement

3.2.5.2 La sélection pour le remplacement

Il existe aussi plusieurs méthodes de sélection pour le remplacement en vue de former la génération suivante. Dréo *et al.* [2003] présentent quelques-unes de ces méthodes à savoir le remplacement générationnel, le remplacement stationnaire, les stratégies d'évolution et la méthode d'élitisme.

3.2.5.2.1 Le remplacement générationnel

Ce type de remplacement est le plus simple. Considérant une population de taille *N*, les *N* enfants générés par le croisement et la mutation remplacent purement et simplement les parents de la génération courante afin de constituer la prochaine génération.

3.2.5.2.2 Le remplacement stationnaire (Steady-State)

Dans ce schéma, un petit nombre de descendants (un ou deux) sont engendrés à chaque génération. Ceux-ci sont réinsérés dans la population en remplacement d'un nombre inférieur ou égal de parents. Ces derniers peuvent être sélectionnés par un tournoi inversé (le gagnant étant le moins performant), de manière stochastique (uniforme ou non) ou déterministe (les parents les moins performants sont remplacés).

3.2.5.2.3 La méthode d'élitisme

Cette méthode consiste à conserver dans la génération suivante, au moins le meilleur individu de la génération courante. Le même individu peut donc se retrouver dans plusieurs générations consécutives. Ceci favorise l'exploitation des meilleures solutions au détriment de l'exploration de l'espace de recherche. Il pourrait en résulter une convergence prématurée de l'algorithme vers cette solution.

3.2.5.2.4 Les stratégies d'évolution

Deux schémas sont regroupés sous ces appellations : les schémas *(μ, λ)-ES* et *(μ+λ)-ES*. À partir d'une population de taille μ, λ enfants sont générés par application des opérateurs génétiques, λ étant supérieur à μ. L'étape de remplacement est totalement déterministe. Dans le schéma *(μ, λ)-ES*, les μ meilleurs enfants deviennent les parents de la

génération suivante. Dans le schéma *(μ + λ)-ES*, les meilleurs des (μ + λ) parents et enfants deviennent les parents de la génération suivante. Ce deuxième schéma est aussi considéré comme une méthode élitiste.

3.2.6 Le croisement

Le croisement permet, par la manipulation de la structure des chromosomes, l'enrichissement de la population. Il consiste en un échange de gènes entre deux ou plusieurs chromosomes afin d'en former de nouveaux. Classiquement, les croisements impliquent deux parents qui génèrent un ou deux enfants. Selon Davis [1991], c'est le croisement (encore appelé *recombinaison* ou *crossover*) qui fait la force des AG. Il s'agit de sélectionner deux individus parmi les parents potentiels, aléatoirement ou à l'aide d'une des méthodes de sélection pour la reproduction. Ces derniers sont croisés suivant une probabilité p_{crois} appelée *probabilité de croisement* afin d'obtenir de nouveaux individus appelés 'enfants'. Parfois, les 'bons' gènes d'un parent se substituent aux 'mauvais' gènes de l'autre pour former un meilleur descendant. La probabilité de croisement est plus ou moins élevée dans les AG. Elle varie souvent entre *0,7* et *1* [Yang 2008].

Il existe plusieurs méthodes de croisement en fonction de la méthode de représentation des solutions. Pour la représentation par permutation, les méthodes suivantes peuvent être citées : le *croisement à un ou plusieurs points*, le *croisement partiellement tracé (PMX)*, le *croisement par ordre (OX)*, le *croisement par cycle (CX)* et le *croisement par recombinaison d'arêtes (ERX)*.

Le *croisement à un point* [Holland 1962] consiste à choisir aléatoirement, sur les chromosomes parents, un point de croisement à partir duquel ceux-ci échangent leurs gènes. Il faut noter que le croisement s'effectue au niveau des gènes, ce qui permet de couper les chromosomes à n'importe quel niveau. Il existe aussi des croisements multipoints. Le plus courant, le *croisement à deux points*, consiste à choisir aléatoirement deux points sur les chromosomes 'Parents' et à échanger les gènes compris entre ces deux points. Dans le cas de la représentation de solutions par permutation des gènes, ces méthodes de croisement entraînent à coup sûr la formation de solutions non viables (répétition de certains gènes et suppression d'autres).

Reeves [1995] définit une technique générale de croisement (à un ou plusieurs points) pour la représentation des solutions par permutation. La Figure 3.7 présente un exemple pour le croisement à deux points où les chromosomes sont coupés entre le troisième et le sixième emplacement. Les gènes du premier chromosome sont recopiés dans le chromosome 'enfant' jusqu'au premier point de croisement. Ensuite, les gènes du second chromosome qui ne font pas encore partie du chromosome 'enfant' sont recopiés dans l'ordre à partir du début jusqu'au second point de croisement. Les gènes restants sont enfin recopiés dans le chromosome 'enfant' selon leur ordre dans le premier parent. La procédure est la même pour les autres types de croisement à points basés sur la représentation par permutation. Dans le cas du RCPSP, cette technique conserve aussi les relations de préséance au sein du chromosome [Hartmann 1998]. Dans l'exemple de la Figure 3.7, toutes les relations de préséance communes aux deux chromosomes 'parents' sont en effet maintenues dans les chromosomes 'enfants'.

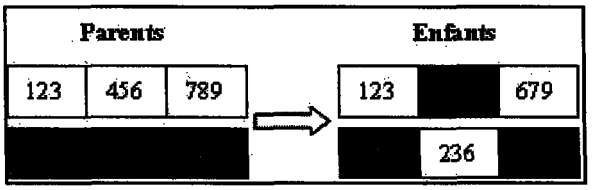


Figure 3.7 : Illustration du croisement à deux points

La première étape du *PMX (Partially Mapped Crossover)* [Goldberg et Lingle 1985] est la sélection de deux points de coupure sur chaque chromosome 'parent'. Les gènes compris entre ces deux points de coupure sont ensuite échangés. Les gènes dupliqués dans les chromosomes 'enfants' obtenus sont ensuite remplacés par leurs correspondants qui ont été supprimés par le croisement. La Figure 3.8 montre un exemple illustrant le *PMX* avec les deux parents précédents. À l'Étape 1, les gènes situés entre les points de coupure sont échangés. À l'Étape 2, les gènes dupliqués qui se situent à l'extérieur des deux points de coupure sont remplacés. Ainsi, dans le premier enfant, les gènes *8* et *2* à l'extérieur des points de coupure sont remplacés par les gènes *4* et *5* respectivement. Cet opérateur de croisement essaye de préserver le plus possible la position absolue des gènes. Le nombre de gènes qui n'héritent pas leur emplacement de l'un des deux parents, après le croisement, est au plus égal au double de la taille de la fenêtre de croisement [Potvin 1996]. Les relations de préséance pour le RCPSP ne sont pas systématiquement maintenues par cet opérateur. L'exemple montre en effet que le gène *5* précède le gène *8* dans les deux parents, ce qui n'est pas le cas dans le second chromosome 'enfant'.

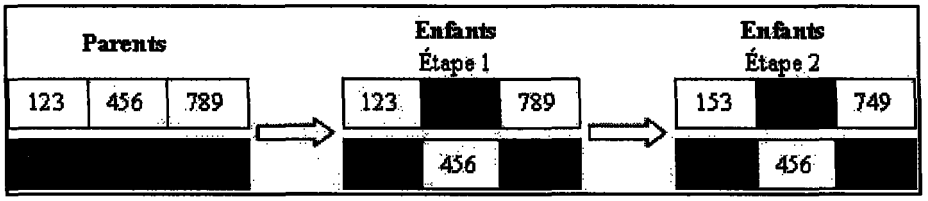


Figure 3.8 : Illustration du croisement partiellement tracé *(PMX)*

L'*OX (Order Crossover)* [Goldberg 1989; Oliver *et al.* 1987] ressemble au *PMX* à la différence près que l'*OX* tente plutôt de maintenir la position relative des gènes alors que le *PMX* essaie de maintenir leurs positions absolues. Dans un premier temps, les gènes entre les points de coupure du premier parent sont recopiés dans le chromosome 'enfant'. Le second parent est ensuite parcouru de la gauche vers la droite, à partir du second point de coupure, et les gènes manquants dans le chromosome 'enfant' y sont recopiés dans l'ordre du second parent. L'exemple précédent est reproduit à la Figure 3.9 en utilisant, cette fois-ci, l'opérateur *OX* pour effectuer le croisement. À la première étape, les gènes compris entre les emplacements *3* et *6* sont échangés. À la seconde étape, les places vides sont remplies, à partir du second point de coupure, avec les gènes de l'autre parent qui n'apparaissent pas encore dans le chromosome 'enfant'. L'*OX* ne maintient pas non plus les relations de préséance entre les activités du RCPSP. Dans l'exemple de la Figure 3.9, plusieurs relations de préséance communes aux deux chromosomes 'parents' sont en effet brisées dans les chromosomes 'enfants'.

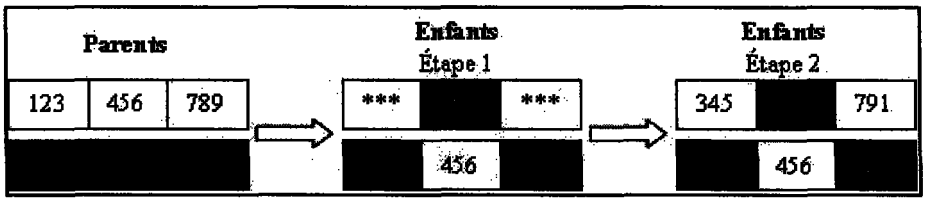


Figure 3.9 : Illustration du croisement par ordre *(OX)*

Le *CX (Cycle Crossover)* [Oliver *et al.* 1987], comme son nom l'indique, utilise des cycles pour réaliser l'opération de croisement. Chaque gène du chromosome 'enfant' est situé au même emplacement dans l'un ou l'autre des deux parents. Le principe de cet opérateur est le suivant : on part du premier gène de l'un des parents (P_1) que l'on place au premier emplacement dans le chromosome 'enfant'. Le premier gène du second parent (P_2) est ensuite recherché dans P_1. Ce gène, qui se trouve à un emplacement a dans le parent P_1, est alors positionné au même emplacement a dans le chromosome 'enfant'. Le *a-ième* gène du parent P_2 est recherché dans le parent P_1, et ainsi de suite jusqu'à former un cycle. À la fin du premier cycle, un autre cycle est réalisé en commençant, cette fois-ci, par les gènes du parent P_2. L'alternance est ainsi effectué à la fin de chaque cycle, jusqu'à l'obtention de tous les gènes du chromosome 'enfant'. À la Figure 3.10, l'exemple précédent est repris illustrant le fonctionnement du *CX*. Dans cet exemple, quatre cycles de gènes ont été obtenus pour chaque enfant à savoir pour le premier enfant, les cycles '1', '4253784', '6' et '9'. Les gènes situés au même emplacement dans les deux parents forment un cycle unitaire contenant un seul gène. Cet exemple montre un cas particulier du *CX* où tous les cycles formés sont unitaires, sauf un, ce qui fait que les enfants obtenus sont en tous points semblables aux parents.

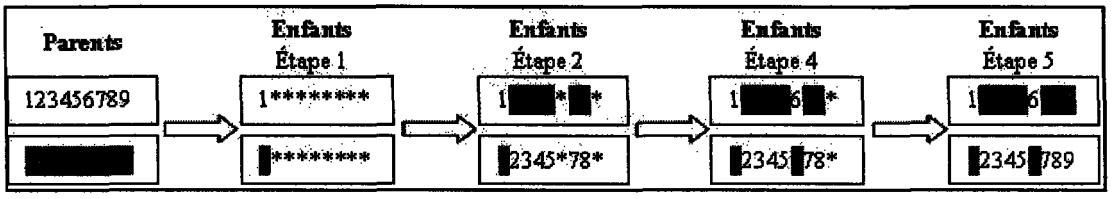


Figure 3.10 : Illustration du croisement par cycle *(CX)*

L'*ERX (Edge Recombination Crossover)* a été introduit par Whitley *et al.* [1989]. Cet opérateur tente de préserver le plus d'arêtes possibles provenant des parents de façon à minimiser l'introduction d'arêtes additionnelles. Pour cela, une carte d'arêtes est construite, contenant toutes les connexions entre les différents gènes pour les deux parents. La présence d'une d'arête entre deux gènes indique que ces gènes sont voisins dans l'un ou l'autre des deux parents. Pour construire un enfant, un gène courant est choisi au départ entre les gènes des parents. La carte d'arêtes est ensuite utilisée pour choisir le prochain gène courant. Parmi les gènes liés au gène courant dans l'un des parents, celui ayant le moins de liens disponibles dans la carte d'arêtes est sélectionné. En cas d'égalité entre deux gènes ou plus, l'un d'entre eux est choisi aléatoirement. Lorsqu'un gène est choisi, la carte d'arêtes est remise à jour. La Figure 3.11 illustre la carte d'arêtes initiale pour l'exemple précédent, où P_1=*123456789* et P_2=*145826379*.

Le gène *1* possède des arrêtes vers : 2, 4, 9

Le gène *2* possède des arrêtes vers : 1, 3, 6, 8

Le gène *3* possède des arrêtes vers : 2, 4, 6, 7

Le gène *4* possède des arrêtes vers : 1, 3, 5

Le gène *5* possède des arrêtes vers : 4, 6, 8

Le gène *6* possède des arrêtes vers : 2, 3, 5, 7

Le gène *7* possède des arrêtes vers : 3, 6, 8, 9

Le gène *8* possède des arrêtes vers : 2, 5, 7, 9

Le gène *9* possède des arrêtes vers : 1, 7, 8

Figure 3.11 : Carte d'arêtes initiale

Les opérations successives effectuées avec l'opérateur *ERX*, à partir de la carte de la Figure 3.11, sont illustrées à la Figure 3.12. Le gène *1* est sélectionné au début de l'exemple. Les gènes suivants sont ensuite sélectionnés l'un après l'autre, en respectant les étapes décrites ci-dessus. Le chromosome 'enfant' obtenu à la fin du croisement est : *145897623*. Il faut noter que les relations de préséance du RCPSP ne sont pas préservées par cet opérateur de croisement. Le gène *3*, par exemple, précède les gènes *7* et *9* dans les deux chromosomes 'parents', ce qui n'est pas le cas dans le chromosome enfant obtenu dans cet exemple.

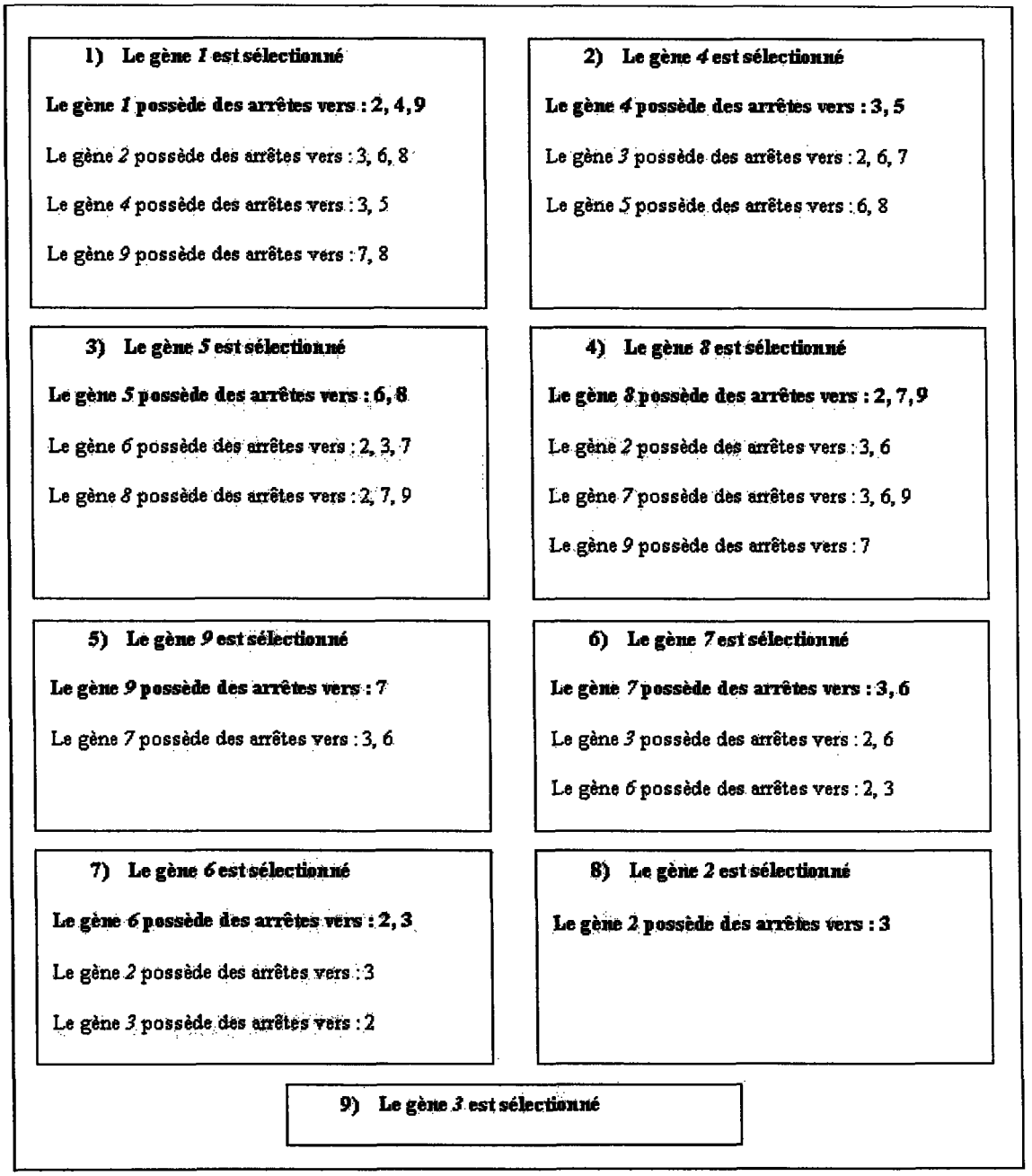


Figure 3.12 : Illustration du croisement par recombinaison d'arêtes *(ERX)*

Parmi les méthodes de croisement présentées ci-dessus, la technique de croisement à points de Reeves [1995] semble la plus adaptée pour résoudre le RCPSP car elle

maintient non seulement l'intégrité de la solution, mais aussi, comme vu précédemment à la page 47 et à la Figure 3.7, les relations de préséance qui constituent une particularité importante du problème.

3.2.7 La mutation

La mutation permet de favoriser la diversification dans la population afin d'avoir des individus de tout genre et de pouvoir choisir les meilleurs. Dans le but d'empêcher une convergence trop rapide et d'obtenir cette variété dans la population, on applique donc aux individus de chaque génération une mutation selon une probabilité p_{mut} donnée. Cette probabilité est appelée le *taux de mutation.* Contrairement au croisement, la mutation est une opération qui implique un seul chromosome. Le taux de mutation est généralement faible dans le cas des AG [Dréo *et al.* 2003]. Selon les auteurs, un taux de mutation élevé équivaudrait à une marche aléatoire dans l'espace de recherche; le rôle de l'opérateur de croisement serait dérisoire et on assisterait à une convergence trop lente de l'algorithme. Ils proposent un taux de mutation compris entre *0,01* et *0,1*. Yang [2008] abonde dans le même sens en proposant un taux de mutation compris entre *0,001* et *0,05*.

Il existe plusieurs méthodes de mutation. Les méthodes les plus courantes pour la représentation par permutation sont la mutation par insertion, la mutation par échange et la mutation par inversion.

La mutation par insertion [Boctor 1996] consiste à choisir aléatoirement un gène et à l'insérer dans le même chromosome, à un emplacement choisi aléatoirement. La Figure

3.13 montre un exemple de mutation par insertion où le troisième gène est inséré à l'emplacement *7*.

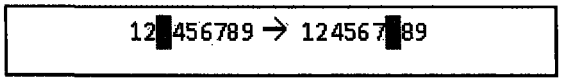


Figure 3.13 : Illustration de la mutation par insertion

La mutation par échange [Reeves 1993] consiste à interchanger deux gènes sur un chromosome. Cette méthode est illustrée par l'exemple de la Figure 3.14. Les gènes situés aux emplacements 4 et 7 sont interchangés.

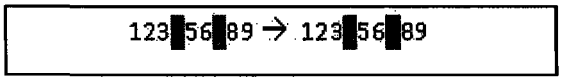


Figure 3.14 : Illustration de la mutation par échange

Sait et Youssef [1999] proposent une technique d'inversion pour la représentation par permutation. Il consiste à sélectionner sur un chromosome, deux points entre lesquels la chaîne de gènes est inversée. L'exemple de la Figure 3.15 illustre bien cette technique. Dans cet exemple, les gènes de l'emplacement 4 à l'emplacement 8 sont inversés.

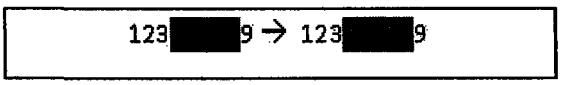


Figure 3.15 : Illustration de la technique de l'inversion [Sait et Youssef 1999]

La liste des opérateurs présentés dans cette section n'est pas exhaustive. Il existe beaucoup d'autres opérateurs de sélection, de croisement et de mutation. L'AG est une méthode relativement ancienne qui a été beaucoup utilisée pour résoudre divers problèmes d'optimisation. Les opérateurs ont été adaptés selon la structure de chaque problème. La

section qui suit présente une méthode plus récente, qui se fait de plus en plus connaître dans le monde de l'optimisation, l'OEP.

3.3 L'optimisation par essaims particulaires (OEP)

L'OEP est une métaheuristique récente qui a été mise au point par le sociologue James Kennedy et l'ingénieur électrique Russell C. Eberhart en 1995 [Kennedy et Eberhart 1995]. Les auteurs s'inspirent des observations effectuées lors de simulations informatiques des comportements sociaux des êtres vivants tels que le déplacement collectif d'un banc de poissons ou d'un groupe d'oiseaux par Reynolds [1987] et Heppner et Grenander [1990].

Contrairement à la plupart des métaheuristiques, l'OEP est utilisée à l'origine pour déterminer l'optimum de fonctions continues non linéaires. Cependant, plusieurs auteurs l'ont récemment adaptée pour résoudre des problèmes discrets tels que le PVC [Tasgetiren *et al.* 2007b], l'ordonnancement de véhicules [Chen *et al.* 2006], le problème d'atelier à cheminement unique [Liao *et al.* 2007], le problème d'atelier à cheminements multiples [Sha et Hsu 2006], le problème de machine unique [Anghinolfi et Paolucci 2009], ainsi que le RCPSP [Zhang *et al.* 2005].

L'OEP présente quelques similarités avec les AG en ce sens qu'elle fonctionne sur la base d'une population de solutions qui interagissent entre elles afin de trouver la meilleure solution possible à un problème d'optimisation. Cependant, contrairement aux AG où l'évolution est basée sur la compétition et l'élimination des individus les moins performants, l'OEP se base sur la coopération entre les individus afin de faire évoluer

chacun. Aucune solution n'est éliminée, chacune utilisant les connaissances que ses voisines possèdent du milieu afin de se déplacer efficacement et de s'améliorer.

3.3.1 Fonctionnement général de l'OEP

L'OEP repose sur un ensemble de solutions disposées aléatoirement au début de l'algorithme. Chaque solution est appelée *particule* et se situe à une *position* donnée dans l'espace de recherche. Chaque particule dispose d'une mémoire dans laquelle est conservée sa meilleure position visitée. L'OEP fonctionne aussi de manière itérative. À chaque itération de l'algorithme, les particules se déplacent dans l'espace de recherche selon une certaine *vitesse*. Le calcul de cette vitesse dépend de plusieurs facteurs tels que la vitesse actuelle pondérée par un coefficient d'inertie w, ainsi que l'écart par rapport à leur meilleure position connue et à celle de leurs voisines pondérées respectivement par des coefficients de confiance c_1 et c_2. Les particules voisines d'une autre particule sont appelées ses *informatrices*.

La structure de base d'un algorithme d'OEP est illustrée par le schéma de la Figure 3.16. Comme dans le cas des AG, une population initiale est d'abord générée et évaluée. Les principes de génération de cette population et d'évaluation des solutions demeurent les mêmes pour les deux méthodes. À chaque itération de l'algorithme, la nouvelle vitesse et la nouvelle position de chaque particule sont calculées. Les particules sont ensuite évaluées et leurs mémoires, i.e. leurs meilleures positions connues, sont mises à jour. La diversité au sein de la population permet à l'algorithme de ne pas rester bloqué dans un optimum local.

Au fil des itérations, la population doit normalement converger vers l'optimum global de la fonction 'objectif'.

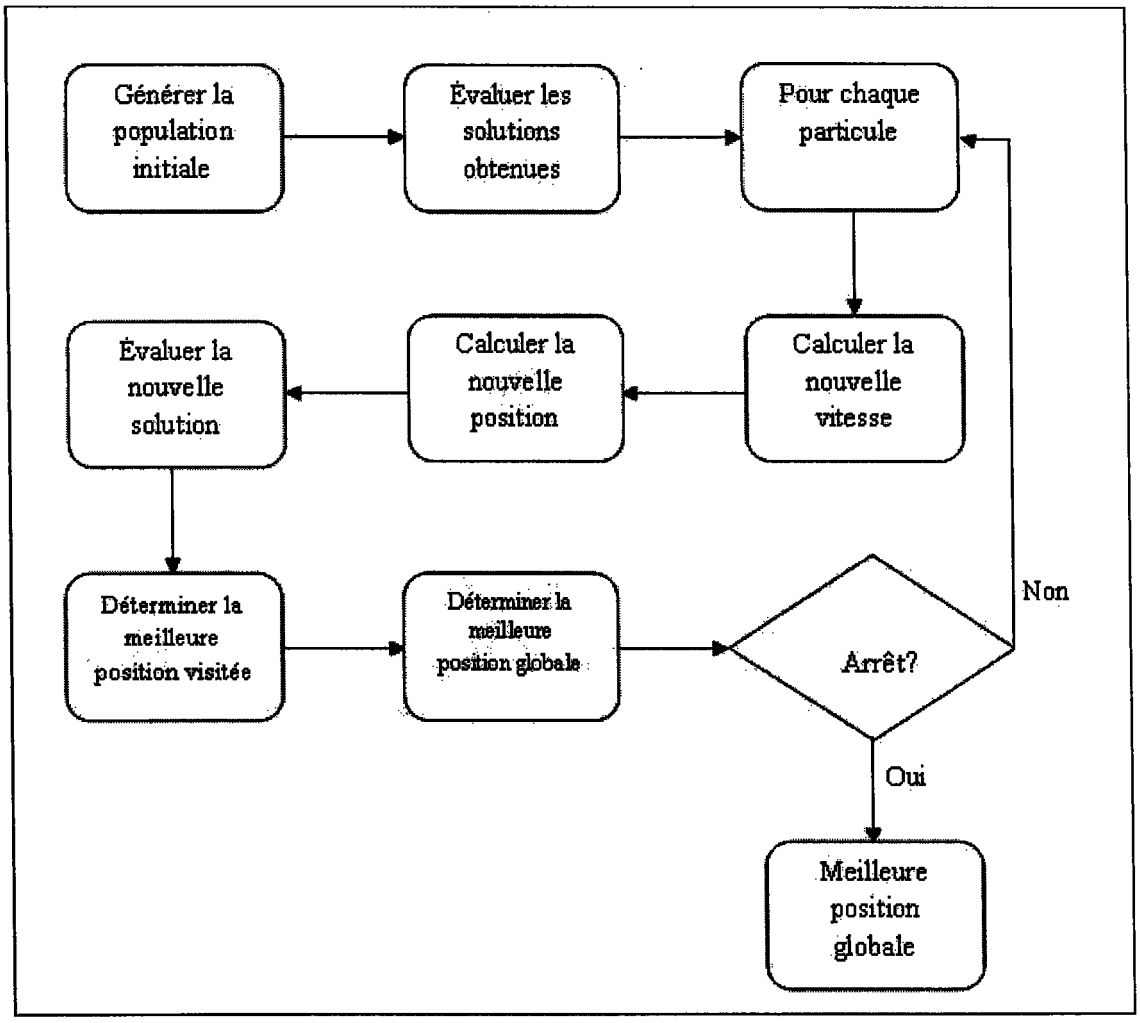


Figure 3.16 : Principe de l'algorithme d'optimisation par essaims particulaires

3.3.2 L'OEP pour l'optimisation continue

Contrairement à l'optimisation combinatoire, les variables à optimiser en optimisation continue évoluent dans un domaine de possibilités infinies. La fonction

'objectif' est donc une fonction réelle $f(x)$ où x est un vecteur de variables dans un espace continu de dimension I, I étant le nombre de variables du problème étudié.

L'OEP ayant été conçue originellement pour traiter des problèmes d'optimisation continue, nous commençons par décrire la méthode de calcul de la vitesse V_n et de la position X_n d'une particule π_n dans ce domaine. Ces deux éléments sont donnés pour une itération t donnée par les Équations 3.2 et 3.3 où p_n est la meilleure position visitée par la particule π_n et l_n est la meilleure position visitée par ses informatrices. w représente le facteur d'inertie, c_1 et c_2 sont les coefficients de confiance de l'algorithme qui seront définis aux Sections 3.3.2.3 et 3.3.2.4 et r_1 et r_2 sont des nombres réels choisis aléatoirement dans l'intervalle $[0,1]$.

$$V_n(t) = w.V_n(t-1) + c_1.r_1.(p_n - X_n(t-1)) + c_2.r_2.(l_n - X_n(t-1)) \qquad \textit{(Équation 3.2)}$$

$$X_n(t) = X_n(t-1) + V_n(t) \qquad \textit{(Équation 3.3)}$$

Ce modèle est appelé modèle *lbest* (*local best model*) par les auteurs de la méthode [Kennedy 2006]. Lorsque la notion de voisinage d'une particule, définie à la Section 3.3.2.2, est élargie à toute la population, la meilleure position visitée par les informatrices est la meilleure position globale g et on obtient le modèle *gbest* (*global best model*) dont la formule de calcul de la vitesse est donnée par l'Équation 3.4.

$$V_n(t) = w.V_n(t-1) + c_1.r_1.(p_n - X_n(t-1)) + c_2.r_2.(g - X_n(t-1)) \qquad \textit{(Équation 3.4)}$$

La configuration de l'OEP nécessite le réglage de divers paramètres. Ces paramètres ont un impact direct sur les performances de l'algorithme. Ils sont présentés dans les sous-sections qui suivent.

3.3.2.1 La taille de la population

Il n'existe pas de règle prédéfinie pour déterminer la taille de la population. Cela dépend de l'espace de recherche, des capacités de la machine utilisée et du temps de calcul alloué. Clerc [2005] conseille néanmoins une taille de *20* à *40* particules.

3.3.2.2 La topologie et le choix du voisinage

Le voisinage d'une particule peut être global ou local. Dans un voisinage global, chaque particule est reliée à toutes les autres particules. La meilleure informatrice est alors la meilleure solution globale. Lorsque le voisinage est local, on tire aléatoirement à chaque itération et pour chaque particule, k voisines qu'elle doit informer, où k est un nombre prédéfini généralement entre *3* et *5* [Clerc 2005]. On peut aussi décider de former un certain nombre de partitions et d'associer chaque particule à une partition donnée [Parsopoulos et Vrahatis 2007].

3.3.2.3 Le facteur d'inertie

Le facteur d'inertie w représente la confiance que la particule a en sa vitesse actuelle. Il permet de définir la capacité d'exploration de chaque particule en vue d'améliorer la convergence de la méthode. Une grande valeur de w est synonyme d'une grande amplitude de mouvement et donc d'exploration globale de l'espace de recherche.

Au contraire, une faible valeur de w est synonyme de faible amplitude de mouvement et donc d'exploitation locale. Fixer ce facteur revient donc à trouver un compromis entre l'exploitation et l'exploration de l'espace de recherche. Bien que Shi et Eberhart [1998] conseillent une valeur entre *0,8* et *1,2*, on rencontre généralement dans la littérature des valeurs de w se situant entre *0,5* et *1*. La détermination de la meilleure valeur de ce paramètre pour chaque algorithme se fait à travers des expérimentations numériques. Il est également possible de diminuer le facteur d'inertie au cours du temps, un peu à la manière de la température dans un algorithme de recuit simulé [Kirkpatrick *et al.* 1983]. De bons résultats ont été trouvés pour une valeur décroissant linéairement de 0.9 à 0.4 [Shi et Eberhart 1999].

3.3.2.4 Les coefficients de confiance

Les coefficients de confiance sont des constantes qui pondèrent les tendances de la particule à vouloir suivre son instinct de conservation (c_1) et à aller vers sa meilleure position connue, ou son panurgisme (c_2) en se rapprochant plus de la meilleure position de ses informatrices. Ces valeurs sont définies de façon empirique. On choisit généralement ces deux constantes de sorte que leur somme soit inférieure à *4*. Clerc [2005] conseille une valeur autour de *1.5* pour chacun des coefficients.

3.3.2.5 Le coefficient de constriction

Pour éviter l'explosion du système, i.e. que les particules se déplacent trop vite et au hasard dans l'espace de recherche, un mécanisme de confinement d'intervalle est souvent

mis en place. La pratique la plus courante est de définir une vitesse maximale V_{max} à ne pas dépasser.

Clerc [1999] et Clerc et Kennedy [2002] définissent un coefficient de constriction χ qui permet d'améliorer la convergence de l'algorithme sans avoir besoin de confiner la vitesse des particules. Le calcul de ce paramètre se fait à partir de l'Équation 3.5.

$$\chi = \frac{2}{\left|2-\rho-\sqrt{\rho^2-4\rho}\right|} \quad où \quad \rho=c'_1+c'_2 \ et \ \rho>4 \qquad (Équation\ 3.5)$$

L'Équation 3.6 traduit le calcul de la vitesse V_n lorsque le coefficient de constriction est utilisé. On obtient alors une relation entre le facteur d'inertie w, qui devient équivalent au coefficient de constriction χ et demeure inférieur à 1, et les coefficients de confiance c_1 et c_2 qui sont respectivement équivalents aux produits $\chi.c'_1$ et $\chi.c'_2$.

$$V_n(t) = \chi.(V_n(t-1) + c'_1.r_1.(p_n - X_n(t-1)) + c'_2.r_2.(l_n - X_n(t-1))) \qquad (Équation\ 3.6)$$

Ceci donne d'après les auteurs, une garantie de convergence à l'algorithme. Eberhart et Shi [2000] confirment cette affirmation par des expérimentations numériques. Les auteurs comparent les résultats obtenus en utilisant le confinement V_{max} et ceux obtenus en utilisant uniquement le coefficient de constriction et constatent que ce dernier offre un meilleur taux de convergence.

Ce modèle d'optimisation continue a été transformé par divers auteurs afin de l'adapter à la résolution des problèmes d'optimisation combinatoire, dont le RCPSP.

3.3.3 L'OEP pour l'optimisation combinatoire

Différentes approches discrètes ont été proposées pour l'OEP. Ces approches diffèrent de l'approche continue par la manière dont une particule est associée à une solution discrète, de même que par le modèle de vitesse utilisé.

Anghinolfi et Paolucci [2009] proposent une classification des diverses approches de la littérature concernant l'OEP pour l'optimisation discrète. Selon les auteurs, il y a trois manières d'associer une particule à une solution discrète qui sont la représentation binaire, la représentation par valeurs réelles et la représentation par permutation. De même, trois modèles de vitesse ont été utilisés, à savoir le modèle de valeurs réelles, le modèle stochastique et le modèle basé sur une liste de déplacements.

Kennedy et Eberhart [1997] proposent une première version de l'OEP pour l'optimisation discrète avec des variables binaires. Chaque solution est alors représentée par un vecteur binaire de dimension I, I étant la taille du problème. La vitesse représente la probabilité pour chaque composante du vecteur de prendre la valeur *'1'*. Liao *et al.* [2007] présentent une variation de ce modèle pour traiter le problème d'ordonnancement d'atelier à cheminement unique. Dans ce cas, la particule est représentée par une matrice carrée d'ordre I où chaque élément x_{ij} prend la valeur *'1'* lorsque le travail i est programmé à la position j et *'0'* dans le cas contraire.

La représentation par valeurs réelles, semblable à la représentation par clés aléatoires dans les AG, a été utilisée avec le modèle de vitesse à valeurs réelles par

Tasgetiren *et al.* [2007a] pour résoudre le problème d'ordonnancement d'atelier à cheminement unique, par Sha et Hsu [2006] pour le problème d'ordonnancement d'atelier à cheminements multiples, par Zhang *et al.* [2005] pour le RCPSP, et d'autres encore. Ce modèle ressemble beaucoup au modèle continu mais les valeurs associées aux dimensions de la particule représentent, dans ce cas, la priorité associée à chaque activité. Parsopoulos et Vrahatis [2005] définissent un modèle unifié avec un facteur d'unification $u \in [0, 1]$ utilisé pour calculer la vitesse comme une somme convexe des vitesses obtenues avec les modèles *gbest* et *lbest*.

Le modèle de l'OEP discrète basé sur la représentation par permutation a été introduit par Hu *et al.* [2003] pour résoudre le problème des n reines. Chaque position d'une particule est représentée par une permutation des n reines. Le modèle de vitesse utilisé est stochastique et chaque composante de la vitesse représente la probabilité de déplacer la reine correspondante dans la position de la particule. Les auteurs appliquent aussi un opérateur de mutation à la particule afin d'éviter le piège de l'optimum local. Ce même modèle a été utilisé par Lian *et al.* [2006] pour résoudre les problèmes d'ordonnancement d'atelier à cheminements multiples et à cheminement unique [Lian *et al.* 2008]. Les auteurs associent la particule à une séquence d'activités et empruntent les concepts de croisement et de mutation aux AG afin de définir leur vitesse. Cette approche a été prouvée meilleure que les techniques standards des AG pour les deux problèmes. Récemment, Anghinolfi et Paolucci [2009] ont utilisé le modèle par permutation pour résoudre le problème de minimisation du retard total sur une machine unique. Les auteurs conçoivent une technique particulière où la vitesse représente la distance et le sens de

déplacement d'un travail dans une séquence. Zhang *et al.* [2005] utilisent également ce modèle pour résoudre le RCPSP. Les auteurs utilisent un concept basé sur la méthode de croisement *PMX* pour définir la vitesse des particules. Ils concluent en comparant leurs deux formulations, par valeurs réelles et par permutation, et déduisent que cette dernière donne de meilleurs résultats pour le RCPSP.

Bien qu'elle n'obtienne pas toujours les meilleures performances, l'OEP discrète est décrite comme une méthode ayant beaucoup de potentiel dans le domaine de l'optimisation combinatoire, les résultats obtenus à ce jour étant assez prometteurs [Anghinolfi et Paolucci 2009; Lian *et al.* 2008; Liao *et al.* 2007].

3.4 Objectifs de la recherche

Le RCPSP est un problème d'ordonnancement théorique qui possède plusieurs applications dans le monde réel, aussi bien en industrie que dans le domaine organisationnel. Par ses nombreuses applications et leur importance, le RCPSP a attiré l'attention d'une forte communauté de chercheurs. Plusieurs méthodes ont été proposées pour résoudre ce problème, aussi bien des méthodes exactes que des méthodes approchées. Les métaheuristiques, et plus précisément les AG, se sont distinguées parmi les meilleures méthodes de résolution du RCPSP. L'OEP, quant à elle, est une méthode initialement conçue pour l'optimisation continue et qui récemment, a fait l'objet de quelques adaptations pour résoudre les problèmes d'optimisation discrets et a donné de bons résultats. L'objectif général de ce travail de recherche est de contribuer à l'amélioration des outils de décision en ordonnancement de projet, afin de favoriser l'avancement de la

recherche scientifique dans ce domaine. À cet égard, deux objectifs spécifiques sont proposés.

Le premier objectif spécifique de ce travail de recherche consiste à valider le potentiel de l'OEP pour la résolution du RCPSP. En effet, cette métaheuristique a déjà démontré de bons résultats sur des problèmes présentant des caractéristiques similaires au RCPSP.

Le second objectif spécifique consiste à proposer un schéma d'hybridation entre l'OEP et un AG afin de combiner les forces des deux métaheuristiques. Par sa structure, l'OEP peut en effet être assimilée à une méthode de croisement en AG. Sa combinaison avec une autre méthode de croisement permet ainsi de réaliser un bon compromis entre l'exploitation et l'exploration de l'espace de recherche.

La démarche utilisée dans ce travail consiste à emprunter une formulation efficace de l'OEP et à l'adapter au RCPSP. À cet effet, la formulation proposée par Anghinolfi et Paolucci [2009] qui a donné de bons résultats sur le problème de minimisation du retard total pondéré sur une machine unique est utilisée. Étant donné que ce problème n'est pas soumis à des contraintes de préséance comme le RCPSP, la gestion de ces contraintes a été intégrée. Il est ensuite procédé à la reproduction d'un AG existant et performant pour résoudre le RCPSP afin de servir de base à l'hybridation. Dans ce sens, l'algorithme de Alcaraz et Maroto [2006], qui offre les meilleurs résultats de la littérature sans aucune procédure de recherche locale, a été reproduit. Deux schémas d'hybridation ont, par la suite, été testés. Le premier consiste à remplacer la méthode de croisement initiale de l'AG

reproduit par une nouvelle méthode de croisement inspirée de l'algorithme d'OEP. Le second schéma d'hybridation consiste à utiliser les deux méthodes de croisement dans l'algorithme dans des proportions déterminées. Différentes configurations ont été testées pour les algorithmes conçus; les plus avantageuses ont été retenues à chaque étape. L'utilisation de benchmarks classiques de la littérature a permis de valider l'efficacité des différentes approches proposées.

3.5 Conclusion

Dans ce chapitre, nous avons présenté les bases de l'AG et de l'OEP. Cette présentation n'est absolument pas exhaustive, mais elle montre les grandes lignes de ces deux méthodes. Les paramètres et les opérateurs utilisés par ces algorithmes ont été définis.

L'OEP est une métaheuristique qui s'est fait récemment connaître dans le domaine de l'optimisation. La méthode était conçue initialement pour l'optimisation continue, contrairement à la plupart des autres métaheuristiques. Plusieurs chercheurs se sont intéressés depuis peu à l'adaptation de cette méthode pour résoudre des problèmes d'optimisation discrète et les résultats obtenus sont prometteurs. Il existe d'ailleurs un travail dans ce sens dans la littérature du RCPSP. Les deux chapitres qui suivent décrivent, de manière détaillée, les travaux réalisés dans le cadre de cette recherche. Le Chapitre 4 présente la conception d'un algorithme d'OEP la résolution du RCPSP ainsi que la reproduction d'un AG de la littérature. Dans le Chapitre 5, une hybridation de ces deux méthodes est effectuée.

CHAPITRE 4

ALGORITHME D'OPTIMISATION PAR ESSAIMS PARTICULAIRES ET ALGORITHME GÉNÉTIQUE POUR LE RCPSP

4.1 Introduction

Dans les chapitres précédents, l'état de l'art du RCPSP a été présenté. Les méthodes utilisées dans la littérature pour résoudre ce problème ont aussi été décrites. Un accent particulier a été mis sur deux métaheuristiques, notamment l'AG et l'OEP.

Dans ce chapitre, nous présentons, dans un premier temps, la conception d'un algorithme d'OEP discrète pour résoudre le RCPSP. Cet algorithme est inspiré de celui de Anghinolfi et Paolucci [2009] pour résoudre le problème d'ordonnancement sur une machine unique avec minimisation du retard total. L'algorithme a été adapté et reconfiguré afin de satisfaire aux contraintes incluses dans le RCPSP. Les résultats obtenus ont été comparés à ceux de Zhang *et al.* [2005]. Ces auteurs sont les seuls qui, à notre connaissance, ont tenté de résoudre le RCPSP à l'aide de l'OEP.

Il a été procédé, dans un second temps, à la reproduction de l'AG proposé par Alcaraz et Maroto [2006]. Cet AG utilise une codification 'intelligente' pour résoudre le RCPSP. La version de cet algorithme n'incluant pas la recherche locale est utilisée, afin de pouvoir évaluer la performance réelle de certaines contributions proposées dans ce mémoire.

4.2 Conception d'un algorithme d'OEP discrète pour le RCPSP

Dans la littérature sur le RCPSP, Zhang *et al.* [2005] ont déjà abordé ce problème avec l'OEP discrète. Les auteurs ont proposé deux concepts différents pour mettre en œuvre une particule. Ils ont utilisé le principe du croisement *PMX* pour définir la vitesse avec deux formes de représentation de solution : la représentation par valeurs réelles et la

représentation par permutation. Les auteurs concluent que la représentation par permutation donne de meilleurs résultats.

Anghinolfi et Paolucci [2009] ont proposé un algorithme d'OEP pour résoudre le problème de minimisation du retard total pondéré sur une machine unique. De bonnes performances ont été obtenues par cet algorithme en rivalisant avec les meilleurs résultats de la littérature pour ce problème. De plus, le problème d'ordonnancement sur une machine unique présente certaines similarités avec le RCPSP, en ce sens que c'est aussi un problème de permutation. Il y a cependant une différence notable du fait que le RCPSP est soumis à des contraintes de préséance et de ressources qui doivent être prises en compte dans l'ordonnancement. Nous empruntons donc à Anghinolfi et Paolucci [2009] leurs concepts de vitesse et de mise-à-jour de la position d'une particule, pour réaliser un algorithme d'OEP pour résoudre le RCPSP.

Les résultats dans la littérature du RCPSP sont basés sur le nombre de solutions évaluées. Étant donné le nombre réduit d'évaluations de solutions généralement utilisées pour la comparaison des divers algorithmes (*1000* à *5000*), aucune procédure d'intensification n'est utilisée afin de laisser à l'algorithme d'OEP la possibilité d'évoluer. De plus, l'objectif recherché est d'évaluer l'efficacité de l'approche proposée sans d'autres mécanismes d'intensification. L'algorithme d'OEP proposé est décrit à la section qui suit.

4.2.1 Description de l'algorithme

L'algorithme d'OEP proposé suit la procédure générale de l'OEP précédemment décrite à la Figure 3.16. Le pseudo-code de l'algorithme est présenté à la Figure 4.1. Une présentation détaillée des diverses étapes de cet algorithme est effectuée par la suite.

```
Générer la population initiale.
Evaluer la population initiale.
Conserver la meilleure solution dans la mémoire globale.
Tant que nbreGenerations>=Gencourante
        Mettre à jour les vitesses des particules.
        Mettre à jour les positions des particules.
        Réparer la particule pour respecter les préséances.
        Évaluer les particules et mettre à jour leurs mémoires individuelles.
        Si une meilleure solution globale est trouvée, remplacer la mémoire globale.
        Incrémenter la génération courante.
Fin Tant que
Retourner la meilleure solution trouvée
```

Figure 4.1 : Pseudo-code de l'algorithme d'OEP pour résoudre le RCPSP

4.2.1.1 La codification de la solution

La représentation par permutation est utilisée par l'algorithme d'OEP pour codifier une solution. La génération d'un ordonnancement à partir d'une séquence d'activités donnée se fait à l'aide de l'algorithme série présenté à la Section 2.3.2. Cet algorithme prend en compte la gestion des contraintes de ressources lors de la génération de l'ordonnancement.

4.2.1.2 La génération de la population initiale

Les particules de la population initiale sont générées par une construction progressive probabiliste. Le choix de la prochaine activité dans une particule se fait de façon proportionnelle, à l'aide de la règle de priorité LFT et en respectant les contraintes de préséance. Les vitesses des particules sont initialisées aléatoirement et la méthode adoptée est expliquée à la Section 4.2.1.3, après la définition du concept de la vitesse d'une particule.

4.2.1.3 La vitesse d'une particule

La vitesse d'une particule est un ensemble de déplacements à appliquer aux activités dans la position de la particule afin d'obtenir une autre particule donnée. Ce déplacement est positif lorsqu'il s'effectue vers la droite et négatif lorsque l'activité est décalée vers la gauche. La Figure 4.2 montre le calcul de la différence entre les positions de deux particules π_1 et π_2, i.e. l'ensemble des déplacements *(j,d)* à appliquer à la particule π_1 pour obtenir la particule π_2, *d* étant la distance entre les deux particules pour l'activité *j*. Le déplacement *(1,2)*, par exemple, traduit que l'activité *1* doit être déplacée de *2* emplacements vers la droite dans la particule π_1 pour se retrouver au même emplacement dans la particule π_2 tandis que le déplacement *(3,-1)* traduit que l'activité *3* doit être déplacée d'un emplacement vers la gauche. Les activités situées au même emplacement dans les deux particules, telle que l'activité *4*, ont une distance nulle.

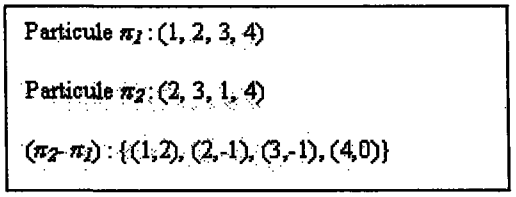


Figure 4.2 : Calcul de la liste de déplacements d'une particule π_1 à une particule π_2

L'Équation 4.1 rappelle la formule servant à calculer la vitesse V_n d'une particule π_n pour le modèle *gbest* à l'itération *t*. Dans cette équation, *w* représente le facteur d'inertie, c_1 et c_2 les coefficients de connaissance, p_n la meilleure position de la particule, *g* la meilleure solution globale de la population et $X_n(t)$ la position de la particule à l'itération *t*. r_1 et r_2 sont des nombres choisis aléatoirement dans l'intervalle *[0, 1]*.

$$V_n(t) = w.V_n(t-1) + c_1.r_1.(p_n - X_n(t-1)) + c_2.r_2.(g - X_n(t-1)) \qquad \textit{(Équation 4.1)}$$

Les termes $(p_n - X_n(t-1))$ et $(g - X_n(t-1))$ de ces équations désignent respectivement la liste de déplacements à appliquer à la particule par rapport à sa meilleure position connue et la liste de déplacements par rapport à la meilleure solution globale, calculées comme illustré à la Figure 4.2.

Soient $C_1=c_1.r_1$ et $C_2=c_2.r_2$. Lors de la mise à jour de la vitesse, on est ainsi amené à multiplier une liste de déplacements par un nombre réel positif w, C_1 ou C_2. Pour ce faire, la distance d associée à chaque activité de la liste est multipliée par ce nombre réel positif. La distance étant un nombre entier, un opérateur de multiplication noté $\otimes$ est défini pour effectuer cette opération tout en conservant l'intégrité de la liste de déplacements. Cet opérateur est conçu comme une fonction qui renvoie le plancher ou le plafond de ce produit, i.e. son arrondi à l'entier inférieur ou à l'entier supérieur le plus proche. Par exemple, si $C_1=0,7$ et $d=2$, le produit $C_1.d$ donne $1,4$; la fonction $C_1 \otimes d$ renvoie alors aléatoirement 1 ou 2.

L'opération d'addition entre deux listes de déplacements revient à faire la somme des distances pour chaque activité. Ainsi, l'addition de deux déplacements (j,d) et (j,d') donne le déplacement $(j,d+d')$. Cet opérateur est noté $\oplus$. L'Équation 4.1 devient alors l'Équation 4.2.

$$V_n(t) = w \otimes V_n(t-1) \oplus C_1 \otimes (p_n - X_n(t-1)) \oplus C_2 \otimes (g - X_n(t-1)) \qquad \textit{(Équation 4.2)}$$

Cette équation montre que le calcul de la vitesse se fait par l'addition ⊕ de trois opérandes respectivement liées à l'inertie (iv_n), à la confiance en soi (pv_n) et à la meilleure solution globale (gv_n). Les Équations 4.3 à 4.5 précisent le calcul de ces trois opérandes.

$$iv_n(t) = w \otimes V_n(t-1) \qquad \textit{(Équation 4.3)}$$

$$pv_n(t) = C_1 \otimes (p_n - X_n(t-1)) \qquad \textit{(Équation 4.4)}$$

$$gv_n(t) = C_2 \otimes (g - X_n(t-1)) \qquad \textit{(Équation 4.5)}$$

La méthode utilisée pour l'initialisation de la vitesse des particules est celle décrite par Anghinolfi et Paolucci [2009] pour leur algorithme. Elle consiste à choisir aléatoirement un entier k entre $I/4$ et $I/2$, I étant le nombre d'activités du projet. Ce nombre k représente le nombre de déplacements à affecter à la particule. Une activité est ensuite choisie au hasard parmi les I activités et elle se voit attribuer une distance aléatoire d choisie dans l'intervalle $[-I/3, I/3]$. Cette opération est répétée k fois, en choisissant à chaque fois une activité différente de celles sélectionnées précédemment. À la fin, une distance nulle est affectée aux activités non sélectionnées.

4.2.1.4 La mise à jour de la position d'une particule

Après avoir mis à jour la vitesse d'une particule, sa nouvelle position est calculée. Pour ce faire, une opération appelée *pseudo-insertion* consiste à déplacer chaque activité suivant sa nouvelle distance d. La Figure 4.3 illustre une pseudo-insertion sur une particule $\pi_1(t-1)=(1, 2, 3, 4)$ avec une vitesse $V_1(t)=\{(1,0), (2,2), (3,-2), (4,0)\}$. Le résultat généré par cette opération est une *pseudo-séquence* qu'il faut corriger afin d'obtenir une nouvelle particule $\pi_1'(t)$.

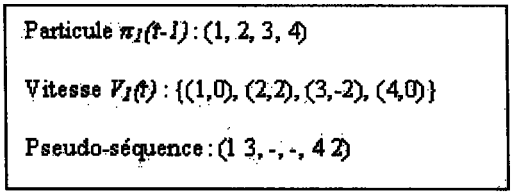


Figure 4.3 : Exemple de pseudo-insertion sur une particule π_1

La procédure de correction d'une pseudo-séquence considère un emplacement à la fois dans la pseudo-séquence, en commençant par le début. Si l'emplacement contient un seul élément, on passe au suivant. Si l'emplacement contient plusieurs éléments, ces derniers sont extraits un à un et insérés à l'emplacement suivant dans la pseudo-séquence, jusqu'à ce qu'il n'en reste qu'un seul. Dans le cas où l'emplacement considéré ne contient aucun élément, la procédure parcours le reste de la pseudo-séquence jusqu'à trouver le premier emplacement non vide, duquel une activité est extraite puis insérée à l'emplacement vide. L'extraction d'une activité à partir d'un emplacement donné de la pseudo-séquence se fait suivant le principe d'une liste FIFO (*First In First Out*). À la fin de la procédure, on obtient une nouvelle particule $\pi_1'(t)$. Les différentes étapes de correction de la pseudo-séquence obtenue à la Figure 4.3 sont illustrées à la Figure 4.4. À l'étape 1, l'activité *1* est extraite du premier emplacement et réinséré à l'emplacement suivant. À l'étape 2, le troisième emplacement, qui était vide, est comblé en extrayant l'activité *4* du quatrième emplacement. On obtient alors la particule $\pi_1'(t)=(3, 1, 4, 2)$. Cette particule peut ne pas respecter les contraintes de préséance de l'instance de RCPSP traitée. La Section 4.2.1.5 décrit la procédure de réparation appliquée à la particule afin de la rendre valide.

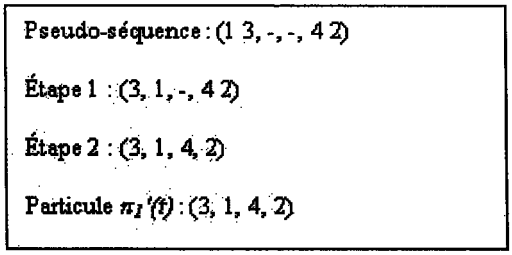


Figure 4.4 : Exemple de procédure de correction d'une pseudo-séquence.

4.2.1.5 La procédure de réparation d'une particule

L'algorithme d'Anghinolfi et Paolucci [2009] a été conçu pour un problème qui n'est pas soumis à des contraintes de préséance. La séquence $\pi_I'(t)$ obtenue à la section précédente peut donc ne pas respecter les contraintes de préséance liées au RCPSP. Une procédure de réparation lui est alors appliquée afin de la rendre valide et de former ainsi la nouvelle particule $\pi_I(t)$. Cette procédure se comporte comme un algorithme de construction progressive qui détermine l'ensemble des activités éligibles à chaque emplacement i de la particule $\pi_I(t)$, suivant les contraintes de préséance. La séquence $\pi_I'(t)$ est ensuite parcourue depuis le début. La première activité éligible rencontrée est sélectionnée et ordonnancée à l'emplacement i dans $\pi_I(t)$. Le pseudo-code de cette procédure est présenté à la Figure 4.5.

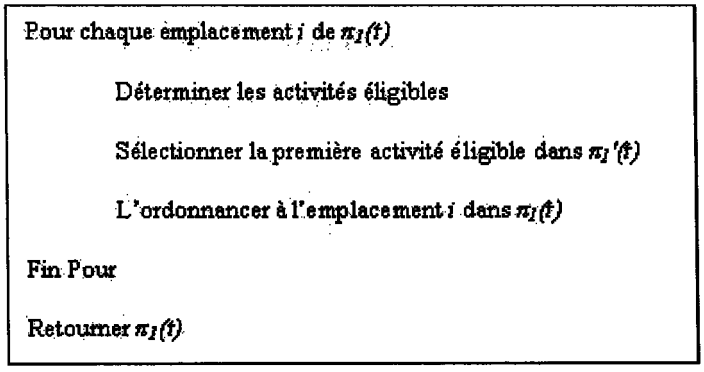


Figure 4.5 : Pseudo-code de la procédure de réparation des préséances d'une particule.

4.2.1.6 Les alternatives de mise à jour de la position d'une particule

L'équation de mise à jour de la position d'une particule est rappelée par l'Équation 4.6. Les procédures de pseudo-insertion, de correction de la pseudo-séquence et de réparation des préséances dans la particule sont nécessaires pour mettre à jour la position d'une particule. Ces procédures constituent donc l'opération d'addition de la vitesse $V_n(t)$ d'une particule π_n à sa position initiale $X_n(t-1)$ dans le but de calculer sa nouvelle position $X_n(t)$ à l'itération t.

$$X_n(t) = X_n(t-1) + V_n(t) \qquad \textit{(Équation 4.6)}$$

Anghinolfi et Paolucci [2009] proposent deux méthodes pour la mise à jour de la position de la particule. La première, appelée *UP1*, consiste à calculer la nouvelle position selon l'Équation 4.6. L'opération d'addition + est alors effectuée une seule fois.

La seconde alternative de mise à jour, *UP2*, consiste à ajouter séparément chacune des opérandes du calcul de la vitesse à la position initiale. Cette procédure est traduite par les Équations 4.7, 4.8 et 4.9. $iv_n(t)$ est l'opérande de la vitesse $V_n(t)$ liée au facteur d'inertie, $pv_n(t)$ l'opérande liée à la confiance en soi et $gv_n(t)$ l'opérande liée à la meilleure solution globale. Le calcul de ces trois opérandes a été présenté dans la Section 4.2.1.3 par les Équations 4.3, 4.4 et 4.5. L'opération d'addition + est donc effectuée trois fois pour la procédure *UP2*, ce qui implique une consommation plus élevée des ressources du système par rapport à la procédure *UP1*.

$$is_1 = X_n(t-1) + iv_n(t) \qquad \textit{(Équation 4.7)}$$

$$is_2 = is_1 + pv_n(t) \qquad \textit{(Équation 4.8)}$$

$X_n(t) = is_2 + gv_n(t)$ *(Équation 4.9)*

La Figure 4.6 montre un exemple illustrant la différence entre les deux méthodes UP1 et UP2 utilisées pour mettre à jour la position d'une particule. Dans cet exemple, la vitesse $V_n(t)$ de la particule est d'abord calculée en faisant la somme des trois opérandes $iv_n(t)$, $pv_n(t)$ et $gv_n(t)$. La méthode UP1 utilise cette vitesse pour mettre à jour la position $X_n(t)$ de la particule en appliquant les procédures de pseudo-insertion, de correction de la pseudo-séquence et de réparation des préséances comme décrites précédemment aux Sections 4.2.1.4 et 4.2.1.5. La méthode UP2, quant à elle, utilise séparément les trois opérandes $iv_n(t)$, $pv_n(t)$ et $gv_n(t)$ et applique à chaque fois les trois procédures nécessaires pour la mise à jour de la particule. Cet exemple produit un résultat final identique pour les deux procédures. Ces résultats pourraient néanmoins être différents, suivant les informations de départ. Il faut noter que lorsque la distance associée à une activité est trop grande (ou trop petite pour un déplacement négatif) pour qu'elle reste au sein de la particule après le déplacement, on assigne à cette activité la distance maximale (ou minimale) qu'elle peut parcourir.

$X_n(t-1)=(1,2,3,4)$

$iv_n(t)= \{(1,1),(2,0),(3,-1),(4,1)\}$

$pv_n(t)=\{(1,0),(2,2),(3,-1),(4,0)\}$

$gv_n(t)= \{(1,-2),(2,3),(3,2),(4,-2)\}$

Calcul de $V_n(t)$:

$V_n(t)= \{(1,-1),(2,5),(3,0),(4,-1)\}$

Contraintes de préséance : *2 précède 3*

- **Méthode UP1 :**

$V_n(t)= \{(1,-1),(2,5),(3,0),(4,-1)\}$ $X_n(t-1)=(1,2,3,4)$

Pseudo-insertion : pseudo-séquence=(1,-,3 4,2)

Correction de la pseudo-séquence : (1,3,4,2)

Réparation des préséances : $X_n(t)=(1,4,2,3)$

- **Méthode UP2 :**

$iv_n(t)= \{(1,1),(2,0),(3,-1),(4,1)\}$ $X_n(t-1)=(1,2,3,4)$

Pseudo-insertion : pseudo-séquence=(-,2 1 3,-,4)

Correction de la pseudo-séquence : (2,3,1,4)

Réparation des préséances : $is_1=(2,3,1,4)$

$pv_n(t)= \{(1,0),(2,2),(3,-1),(4,0)\}$ $is_1=(2,3,1,4)$

Pseudo-insertion : pseudo-séquence=(3,-,1 2,4)

Correction de la pseudo-séquence : (3,1,2,4)

Réparation des préséances : $is_2=(1,2,3,4)$

$gv_n(t)= \{(1,-2),(2,3),(3,2),(4,-2)\}$ $is_2=(1,2,3,4)$

Pseudo-insertion : pseudo-séquence=(1,4,-,2 3)

Correction de la pseudo-séquence : (1,4,2,3)

Réparation des préséances : $X_n(t)=(1,4,2,3)$

Figure 4.6 : Exemple d'application des méthodes de mise à jour UP1 et UP2

Après la mise à jour de leur position, les particules sont évaluées et leur meilleure position individuelle ainsi que la meilleure position globale de la population sont mises à jour.

4.2.2 Essais numériques et résultats obtenus

L'implémentation informatique de l'algorithme présenté dans cette section a été faite en langage C++ selon une approche objet, en utilisant la version 2005 du logiciel Microsoft Visual C++. Toutes les expérimentations ont été effectuées sur un PC Pentium IV 3.2GHz avec 1Go de RAM et utilisant le système d'exploitation Windows XP. Pour des fins de comparaison, l'algorithme a été testé sur trois ensembles d'instances de *30*, *60* et *120* activités générées par ProGen, le générateur d'instances mis au point par Kolisch et *al.* [1995]. Ces ensembles d'instances sont respectivement nommés *J30*, *J60* et *J120*. Ils sont disponibles en téléchargement sur le site internet de PSPLIB à l'adresse http://129.187.106.231/psplib/. Les ensembles *J30* et *J60* comportent chacune *480* instances tandis que l'ensemble *J120* est constitué de *600* instances.

Différentes configurations de l'algorithme d'OEP sont présentées dans cette section. Ces algorithmes sont nommés suivant le modèle OEP($\alpha,\beta,\gamma,\delta$), en fonction des paramètres testés. Dans ce modèle, l'élément 'α' représente l'alternative utilisée pour mettre à jour la position des particules, soit *UP1* ou *UP2*. L'élément 'β' représente le jeu de paramètres w, c_1 et c_2 qui sont les paramètres de calibration de l'OEP, i.e. le facteur d'inertie et les facteurs de connaissance. Deux jeux de paramètres ont été utilisés, à savoir le jeu de paramètres retenu par Anghinolfi et Paolucci [2009] désigné par *Ang* dans le nom de

l'algorithme et un jeu de paramètres calculés à partir du coefficient de constriction χ discuté à la Section 3.3.2.5. Ce dernier jeu de paramètres est désigné par le symbole χ dans le nom de l'algorithme. L'élément 'γ' désigne le type de liste utilisé par la procédure de correction de la pseudo-séquence, soit une liste FIFO (*First In First Out*), une liste LIFO (*Last In First Out*), ou encore une liste mixte avec une procédure qui sélectionne au hasard, à chaque utilisation, entre les listes FIFO et LIFO. L'élément 'δ' représente le nombre de solutions évaluées par l'algorithme. Enfin, le symbole '-' est employé à la place d'un paramètre lorsque celui-ci varie pour le même algorithme. Ainsi, un algorithme d'OEP qui utilise l'alternative *UP1* avec les paramètres d'Anghinolfi et Paolucci [2009] et une liste FIFO pour la procédure de correction des pseudo-séquences et qui effectue *5000* évaluations de solutions, est nommé OEP(*UP1,Ang,FIFO,5000*). Lorsque deux ou plusieurs algorithmes sont comparés, l'élément qui les distingue est mis en gras dans leur nom.

La première série d'expérimentations vise à comparer les deux alternatives de mise à jour de la position des particules, soit *UP1* et *UP2*, en utilisant les meilleurs paramètres retenus par Anghinolfi et Paolucci [2009]. Ces paramètres sont $w=0,5$, $c_1=1,5$, et $c_2=2$. Les deux algorithmes sont donc nommés OEP(***UP1***,*Ang,FIFO,*-) et OEP(***UP2***,*Ang,FIFO,*-). Les ensembles d'instances *J30*, *J60*, et *J120* ont été résolus en utilisant comme critère d'arrêt le nombre d'évaluations de solutions. Le Tableau 4.1 présente les déviations moyennes obtenues par l'algorithme OEP(***UP1***,*Ang,FIFO,*-) pour chaque ensemble d'instances avec *1000* et *5000* évaluations de solutions. Cinq exécutions ont été réalisées dans chaque cas et la moyenne de ces cinq exécutions a été calculée. Le Tableau 4.2 présente les résultats de l'algorithme OEP(***UP2***,*Ang,FIFO,*-) sous la même forme. Dans la

première ligne de chaque tableau, les noms des algorithmes utilisés sont précisés. Les ensembles d'instances résolus sont spécifiés dans la deuxième ligne. De la troisième à la septième ligne sont présentées les déviations moyennes obtenues par rapport à la solution optimale pour chaque exécution de l'algorithme sur les ensembles considérés. La huitième ligne du tableau présente les résultats moyens des cinq exécutions. L'Équation 4.10 donne la formule de calcul de la déviation moyenne pour l'ensemble d'instances *J30*.

$$Déviation\ moyenne = \frac{\sum_{z=1}^{Nombre\ d'instances} \frac{Resultat_z - Optimal_z}{Optimal_z}}{Nombre\ d'instances} \qquad (Équation\ 4.10)$$

Pour les ensembles *J60* et *J120*, les solutions optimales n'étant pas connues à ce jour, la déviation moyenne est calculée par rapport à la longueur du chemin critique nommée *CPLB* dans l'Équation 4.11.

$$Déviation\ moyenne = \frac{\sum_{z=1}^{Nombre\ d'instances} \frac{Resultat_z - CPLB_z}{CPLB_z}}{Nombre\ d'instances} \qquad (Équation\ 4.11)$$

Tableau 4.1 : Résultats obtenus avec l'alternative *UP1*

	OEP(***UP1***,*Ang*,*FIFO*,*1000*)			OEP(***UP1***,*Ang*,*FIFO*,*5000*)		
Instances	J30	J60	J120	J30	J60	J120
1	0,673%	13,67%	42,66%	0,384%	**13,04%**	**41,18%**
2	0,650%	13,67%	42,63%	**0,352%**	13,06%	41,33%
3	0,645%	**13,66%**	42,64%	0,400%	13,08%	41,30%
4	0,643%	13,75%	**42,61%**	0,367%	13,06%	41,33%
5	**0,622%**	13,71%	42,69%	0,385%	13,05%	41,20%
Moyenne	**0,647%**	**13,69%**	**42,65%**	**0,378%**	**13,06%**	**41,27%**

Tableau 4.2 : Résultats obtenus avec l'alternative *UP2*

	OEP(***UP2***,*Ang*,*FIFO*,*1000*)			OEP(***UP2***,*Ang*,*FIFO*,*5000*)		
Instances	J30	J60	J120	J30	J60	J120
1	0,643%	**13,76%**	42,99%	0,385%	13,19%	**41,39%**
2	0,655%	13,81%	**42,91%**	**0,355%**	13,16%	41,48%
3	0,676%	13,77%	42,95%	0,411%	13,21%	41,55%
4	0,690%	13,80%	42,97%	0,378%	13,17%	41,49%
5	**0,626%**	13,81%	42,94%	0,362%	**13,15%**	41,56%
Moyenne	**0,658%**	**13,79%**	**42,95%**	**0,378%**	**13,18%**	**41,49%**

Les meilleurs résultats obtenus dans chaque cas sont mis en gras dans les deux tableaux précédents. Ces tableaux montrent de meilleurs résultats pour l'alternative *UP1* que pour *UP2* aussi bien au niveau des meilleurs résultats obtenus qu'au niveau de la moyenne à l'exception de l'égalité des résultats moyens dans le cas de l'ensemble d'instances *J30* pour *5000* évaluations de solutions.

Les résultats ci-dessus sont traduits par les graphes de la Figure 4.7. Ces graphes superposent les résultats des algorithmes OEP(***UP1***,*Ang*,*FIFO*,-) et OEP(***UP2***,*Ang*,*FIFO*,-) pour chaque colonne des Tableaux 4.1 et 4.2 avec en abscisses les numéros des exécutions et les déviations moyennes en ordonnées. Ils montrent que la courbe de l'algorithme OEP(***UP1***,*Ang*,*FIFO*,-) reste toujours au-dessous de celle de l'algorithme OEP(***UP2***,*Ang*,*FIFO*,-), à part pour les ensembles d'instances *J30* où la courbe de OEP(***UP1***,*Ang*,*FIFO*,-) est au-dessus de celle de OEP(***UP2***,*Ang*,*FIFO*,-) une fois sur cinq. La tendance générale est donc légèrement en faveur de l'algorithme OEP(***UP1***,*Ang*,*FIFO*,-) tant au niveau des résultats individuels que des moyennes observées. L'alternative *UP1* donne de meilleurs résultats que *UP2* pour le RCPSP, contrairement au problème de machine unique pour lequel les auteurs affirment que d'après leurs expérimentations, *UP2*

surpasse toujours *UP1* [Anghinolfi et Paolucci 2009]. La procédure *UP1* est retenue pour la suite des expérimentations. Nous sommes toutefois conscients qu'un test statistique devrait être appliqué pour tirer une conclusion définitive pour la comparaison des deux alternatives. De même, un plus grand nombre d'expérimentations devrait être réalisé pour une meilleure comparaison.

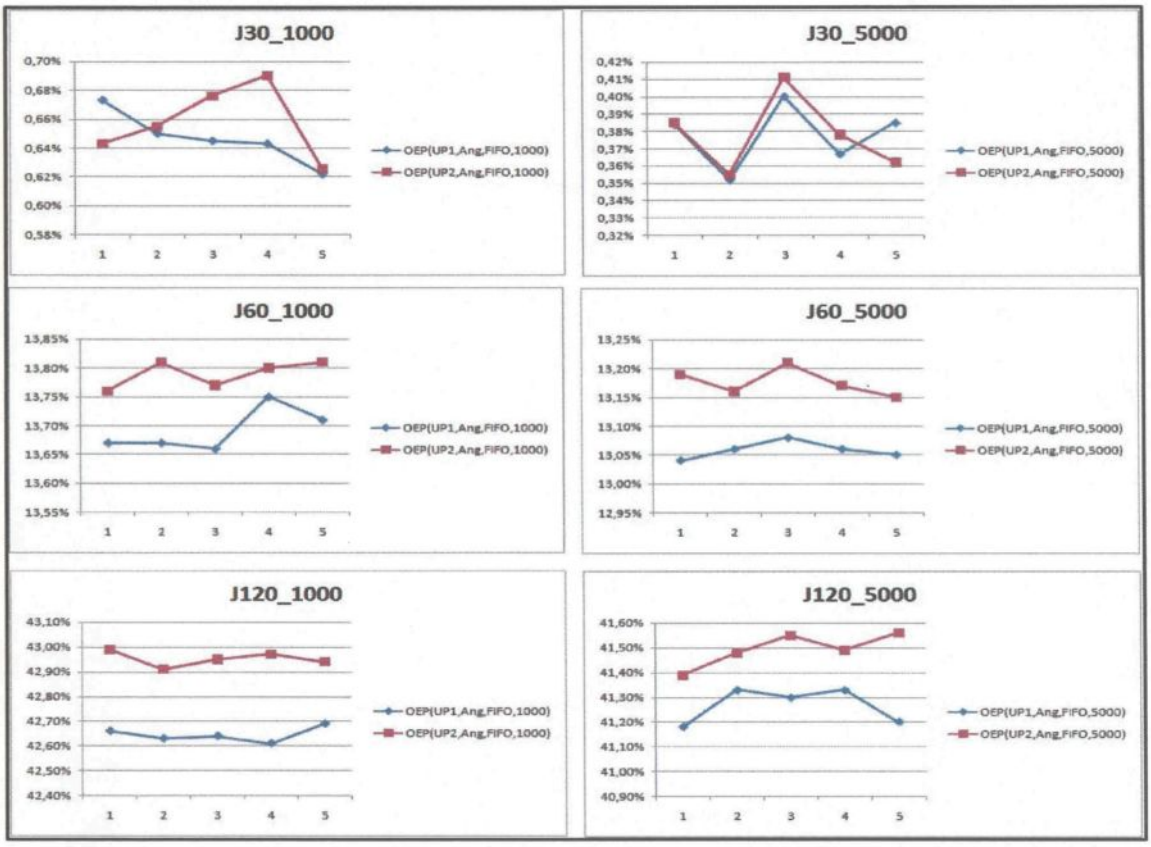


Figure 4.7 : Graphes de comparaison des résultats obtenus par les alternatives de mise à jour *UP1* et *UP2*

Afin d'étudier le comportement de l'algorithme par rapport aux paramètres de configuration de l'OEP à savoir le facteur d'inertie w et les facteurs de connaissance c_1 et c_2, ces paramètres ont été modifiés suivant le facteur de constriction χ introduit à la Section

3.3.2.5 par l'Équation 3.5 et qui garantit la convergence d'un algorithme d'OEP. Pour obtenir $\rho>4$ dans l'équation, les coefficients c'_1 et c'_2 ont été choisis égaux à *2,05*. Cela donne un facteur de constriction $\chi=0,73$. Ainsi à l'Équation 3.5, $\chi.\ c'_1= \chi.\ c'_2=1,50$. Par analogie à l'Équation 4.1 où $\chi=w$, $\chi.c'_1= c_1$ et $\chi.c'_2=c_2$, le poids d'inertie w a été défini à *0,7* et les facteurs de connaissance c_1 et c_2 à *1,5*. L'algorithme avec les nouveaux paramètres est nommé OEP(*UP1,χ,FIFO,-*). Les résultats obtenus sont présentés dans le Tableau 4.3 qui est structuré de la même manière que les tableaux précédents, avec une ligne supplémentaire qui rappelle les résultats obtenus avec l'algorithme retenu OEP(*UP1,**Ang**,FIFO,-*).

Tableau 4.3 : Résultats obtenus par l'algorithme avec le facteur de constriction χ

	OEP(*UP1,χ,FIFO,1000*)			OEP(*UP1,χ,FIFO,5000*)		
Instances	J30	J60	J120	J30	J60	J120
1	**0,618%**	13,73%	**42,75%**	0,395%	13,16%	41,38%
2	0,664%	13,86%	42,86%	0,390%	13,14%	41,48%
3	0,659%	13,70%	42,86%	0,398%	13,11%	41,43%
4	0,650%	**13,68%**	42,78%	0,393%	13,13%	**41,37%**
5	0,652%	13,74%	42,77%	**0,381%**	**13,09%**	41,38%
Moyenne	**0,649%**	**13,74%**	**42,80%**	**0,391%**	**13,13%**	**41,41%**
OEP(*UP1,**Ang**,FIFO,-*)	**0,647%**	**13,69%**	**42,65%**	**0,378%**	**13,06%**	**41,27%**

Ce tableau montre au niveau des moyennes calculées, que l'algorithme OEP(*UP1,**Ang**,FIFO,-*) donne de meilleurs résultats que l'algorithme OEP(*UP1,χ,FIFO,-*), quelque soit l'ensemble d'instances considéré. Une comparaison des déviations minimales obtenues par l'algorithme OEP(*UP1,χ,FIFO,-*) avec ceux de l'algorithme OEP(*UP1,**Ang**,FIFO,-*) dans le Tableau 4.1 donne le même schéma, sauf dans le cas où

1000 solutions sont évaluées pour l'ensemble d'instances *J30*. Il est d'ailleurs à noter que dans ce cas particulier, les résultats obtenus sont assez proches les uns des autres.

Les graphes de la Figure 4.8 superposent les résultats des algorithmes OEP(*UP1,**Ang**,FIFO,-*) et OEP(*UP1,**χ**,FIFO,-*) pour chaque colonne des Tableaux 4.1 et 4.3.

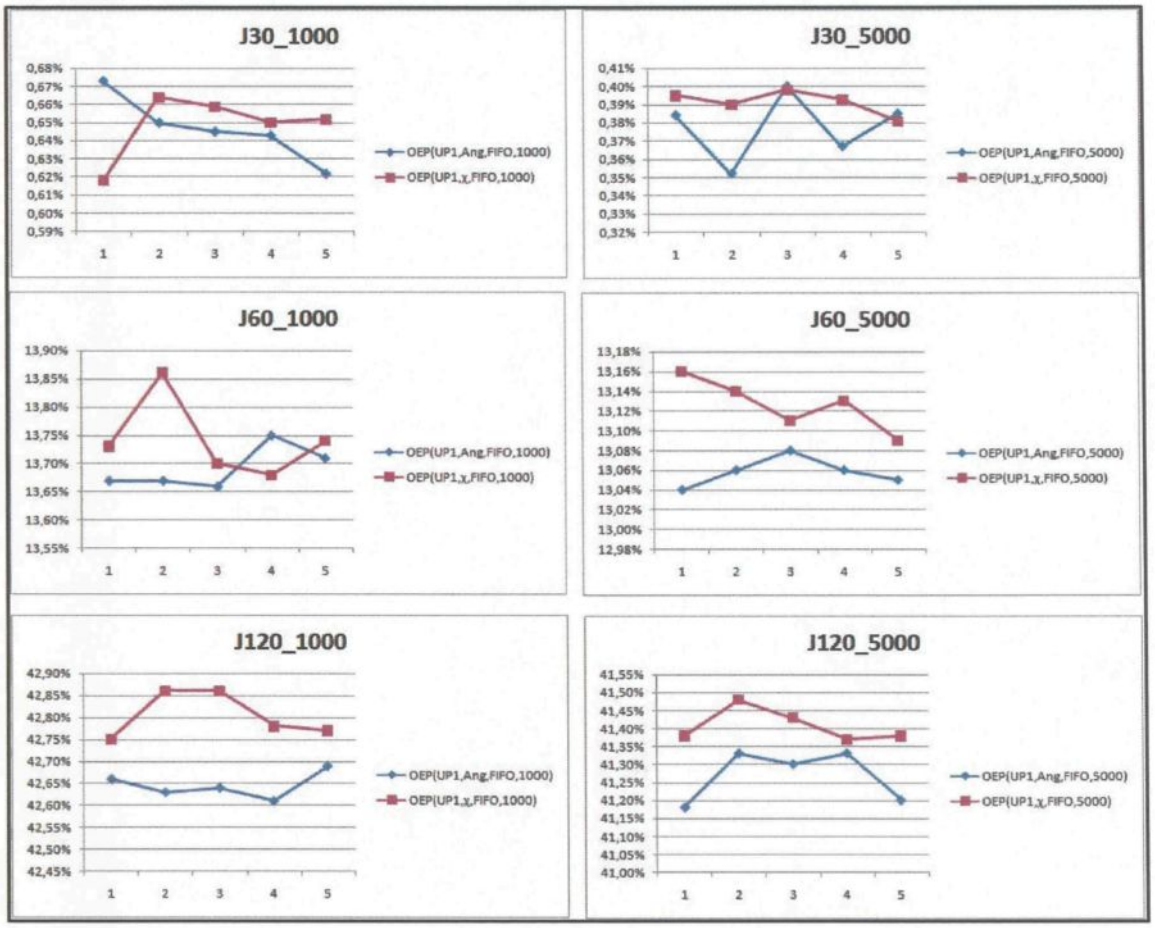


Figure 4.8 : Graphes de comparaison des résultats des algorithmes OEP par rapport à l'utilisation du facteur de constriction χ

À la lumière de ces résultats, les tendances générales sont en faveur de l'algorithme OEP(*UP1,**Ang**,FIFO,-*). Cela pourrait s'expliquer par un fort taux de convergence qui limite l'exploration de l'espace de recherche par l'algorithme OEP(*UP1,χ,FIFO,-*). Nous sommes toutefois conscients qu'un test statistique serait nécessaire pour établir la comparaison stricte de ces résultats. L'étude des deux algorithmes en fonction du nombre d'évaluations de solutions pourrait donner une meilleure compréhension de cette expérience. Le Tableau 4.4 présente les moyennes obtenues pour cinq exécutions par chacun des algorithmes OEP(*UP1,**Ang**,FIFO,-*) et OEP(*UP1,χ,FIFO,-*) sur les ensembles d'instances *J30* pour *1000*, *2000*, *3000*, *4000* et *5000* évaluations de solutions.

Tableau 4.4 : Étude comparée des algorithmes OEP par rapport à l'utilisation du facteur de constriction χ, sur l'ensemble d'instances *J30*

Nombre d'évaluations de solutions	OEP(*UP1,**Ang**,FIFO,-*)	OEP(*UP1,χ,FIFO,-*)
1000	0,647%	0,649%
2000	0,504%	0,534%
3000	0,443%	0,425%
4000	0,419%	0,422%
5000	0,378%	0,391%

Ce tableau montre que l'algorithme OEP(*UP1,**Ang**,FIFO,-*) donne quatre fois sur cinq de meilleurs résultats que l'algorithme OEP(*UP1,χ,FIFO,-*). Une représentation visuelle de ces résultats est donnée par le graphe de la Figure 4.9. Les deux courbes du graphe représentant les résultats des algorithmes OEP(*UP1,**Ang**,FIFO,-*) et OEP(*UP1,χ,FIFO,-*) partent sensiblement du même point à *1000* évaluations de solutions. On observe sur la courbe de l'algorithme OEP(*UP1,χ,FIFO,-*), une pente abrupte qui tend à se stabiliser après *3000* évaluations de solutions. L'algorithme tendrait donc vers la

convergence. Quant à la courbe de l'algorithme OEP(*UP1,**Ang**,FIFO,-*), elle descend progressivement jusqu'à *5000* évaluations de solutions. L'espace de recherche est encore riche et l'algorithme aurait possiblement besoin de plus d'itérations pour converger. Ces résultats confirment la théorie de la convergence liée à l'utilisation de facteur de constriction χ. Bien que les écarts observés entre les deux algorithmes ne soient pas très importants, l'algorithme OEP(*UP1,**Ang**,FIFO,-*) est conservé pour la suite car c'est avec ces paramètres les meilleurs résultats sont obtenus.

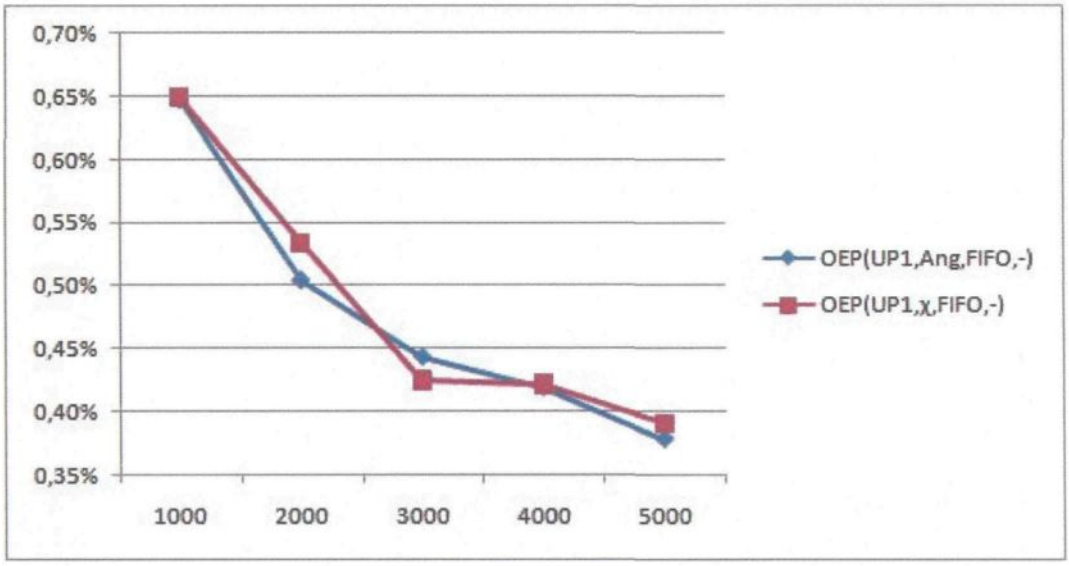


Figure 4.9 : Graphe de comparaison des résultats des algorithmes OEP par rapport à l'utilisation du facteur de constriction χ, sur l'ensemble *J30* en fonction du nombre de solutions évaluées

La mise à jour d'une particule implique le déplacement de certaines activités de la particule suivant leur vitesse. Le calcul de cette vitesse dépend de la distance entre les activités semblables de deux particules différentes. Les activités situées au même emplacement dans ces deux particules ont une distance nulle. Ces activités ne devraient pas

être déplacées si l'on prend en compte seulement ce facteur dans le calcul de la vitesse. Toutefois, l'utilisation d'une liste FIFO pour la correction des pseudo-séquences favorise le déplacement des activités qui avaient été maintenues à leur emplacement initial à cause de leur vitesse nulle. Pour favoriser le maintien de ces activités à leur position initiale, la liste FIFO (*First In First Out*) est remplacée par une liste LIFO (*Last In First Out*). Ceci apporte un peu plus d'intensification dans l'algorithme où la diversification est assez forte. L'algorithme ainsi modifié est appelé OEP(*UP1,Ang,**LIFO**,-*) et le Tableau 4.5 présente les résultats obtenus par cet algorithme. Ce tableau conserve la même structure que le Tableau 4.3. La dernière ligne rappelle les résultats moyens de l'algorithme retenu jusqu'à maintenant, à savoir OEP(*UP1,Ang,**FIFO**,-*).

Tableau 4.5 : Résultats obtenus par l'algorithme OEP utilisant une liste LIFO

	OEP(*UP1,Ang,**LIFO**,1000*)			OEP(*UP1,Ang,**LIFO**,5000*)		
Instances	J30	J60	J120	J30	J60	J120
1	0,644%	13,76%	42,89%	0,408%	13,10%	**41,37%**
2	0,660%	13,72%	42,92%	0,402%	**13,09%**	41,43%
3	**0,602%**	13,71%	42,83%	**0,352%**	13,14%	41,42%
4	0,609%	**13,69%**	**42,80%**	0,361%	13,13%	41,49%
5	0,635%	13,78%	42,92%	0,357%	13,13%	41,42%
Moyenne	**0,630%**	**13,73%**	**42,87%**	**0,376%**	**13,12%**	**41,43%**
OEP(*UP1,Ang,**FIFO**,-*)	**0,647%**	**13,69%**	**42,65%**	**0,378%**	**13,06%**	**41,27%**

Les résultats de ce tableau montrent une meilleure moyenne pour l'algorithme utilisant la liste LIFO sur l'ensemble d'instances *J30* à *1000* évaluations de solutions. La déviation minimale (*0,602*) observée dans ce cas est également meilleure que celle obtenue par l'algorithme utilisant la liste FIFO (*0,622*). À *5000* évaluations de solutions, avec le

même ensemble d'instances, les résultats obtenus sont sensiblement équivalents pour les deux algorithmes. Quant aux ensembles d'instances *J60* et *J120*, la comparaison entre les déviations observées montre de meilleurs résultats pour l'algorithme OEP(*UP1,Ang,**FIFO**,-*) que pour l'algorithme OEP(*UP1,Ang,**LIFO**,-*), aussi bien au niveau des résultats moyens qu'au niveau des déviations minimales obtenues.

Les graphes de la Figure 4.10 comparent les résultats des algorithmes OEP(*UP1,Ang,**FIFO**,-*) et OEP(*UP1,Ang,**LIFO**,-*) pour chaque colonne des Tableaux 4.1 et 4.5.

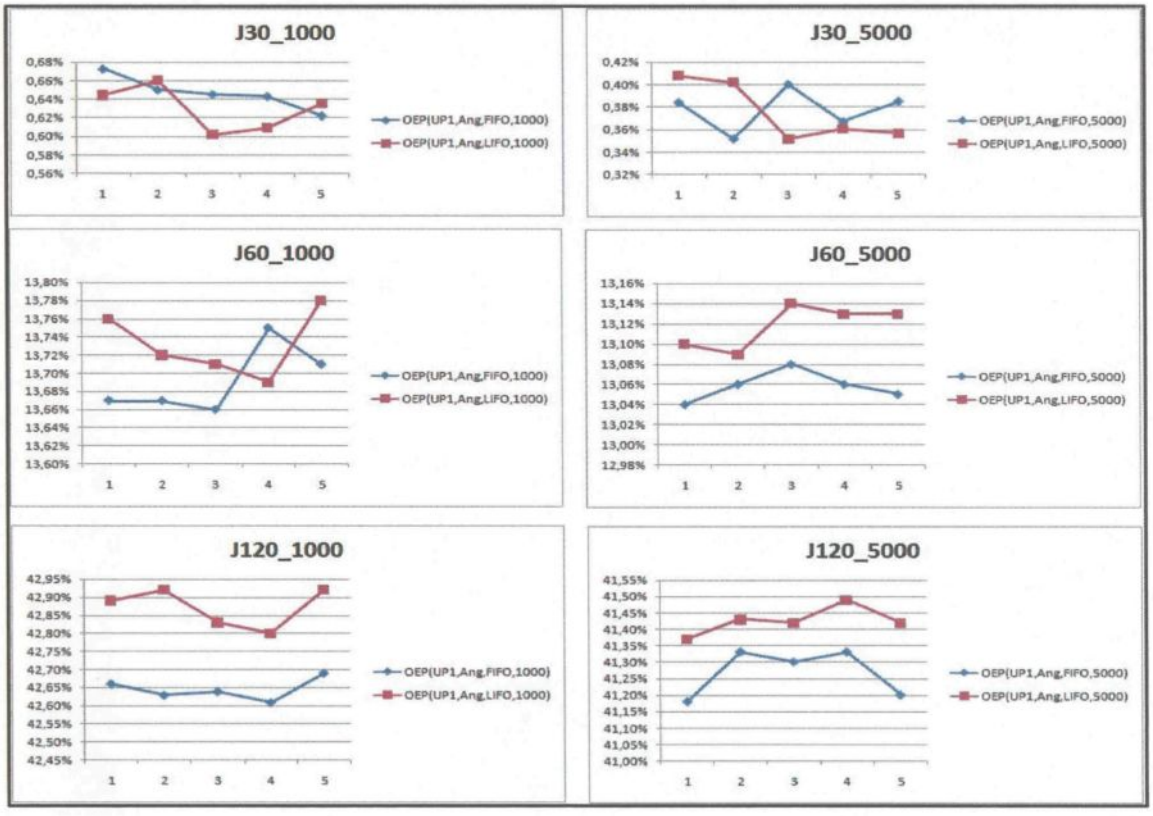


Figure 4.10 : Graphes de comparaison des résultats des algorithmes OEP par rapport au type de liste utilisé

On note une amélioration des résultats pour l'ensemble d'instances *J30* tandis que ceux des ensembles *J60* et *J120* se dégradent. L'intensification semble donc bénéfique seulement pour les instances de petite taille.

Afin d'aller chercher quelques gains sur les instances de petite taille sans trop dégrader les autres, un algorithme mixte utilisant aléatoirement soit une liste FIFO, soit une liste LIFO, a été mis au point. Les résultats obtenus par l'algorithme mixte, nommé OEP(*UP1,Ang,**Mixte**,-*), sont consignés dans le Tableau 4.6 qui a la même structure que le Tableau 4.1 avec deux lignes supplémentaires qui rappellent les résultats moyens obtenus par les algorithmes OEP(*UP1,Ang,**FIFO**,-*) et OEP(*UP1,Ang,**LIFO**,-*).

Tableau 4.6 : Résultats obtenus par l'algorithme OEP utilisant une liste mixte

	OEP(*UP1,Ang,**Mixte**,1000*)			OEP(*UP1,Ang,**Mixte**,5000*)		
Instances	J30	J60	J120	J30	J60	J120
1	**0,588%**	13,68%	42,85%	0,380%	13,17%	**41,30%**
2	0,630%	13,67%	42,82%	0,377%	13,12%	41,38%
3	0,667%	**13,65%**	**42,70%**	0,391%	**13,03%**	41,39%
4	0,659%	13,71%	42,71%	0,381%	13,08%	41,38%
5	0,632%	13,78%	42,84%	**0,369%**	13,06%	41,35%
Moyenne	**0,635%**	**13,70%**	**42,78%**	**0,380%**	**13,09%**	**41,36%**
OEP(*UP1,Ang,**FIFO**,-*)	**0,647%**	**13,69%**	**42,65%**	**0,378%**	**13,06%**	**41,27%**
OEP(*UP1,Ang,**LIFO**,-*)	**0,630%**	**13,73%**	**42,87%**	**0,376%**	**13,12%**	**41,43%**

Ce tableau montre des résultats assez semblables pour les trois algorithmes. Pour l'algorithme OEP(*UP1,Ang,**Mixte**,-*), les résultats moyens obtenus sont intermédiaires entre les résultats obtenus par les algorithmes OEP(*UP1,Ang,**FIFO**,-*) et OEP(*UP1,Ang,**LIFO**,-*). Une exception est faite pour le cas de l'ensemble d'instances *J30* à *5000* évaluations de solutions où on observe une légère dégradation de la moyenne obtenue. De meilleures

déviations minimales que celles obtenues jusque-là sont observées dans trois colonnes du tableau, à savoir pour les ensembles d'instances *J30* et *J60* à *1000* évaluations de solutions et pour l'ensemble *J60* à *5000* évaluations de solutions.

Les graphes de la Figure 4.11 superposent les résultats des trois algorithmes OEP(*UP1,Ang,**FIFO**,-*), OEP(*UP1,Ang,**LIFO**,-*) et OEP(*UP1,Ang,**Mixte**,-*) pour chaque colonne des Tableaux 4.1, 4.5 et 4.6. Ces graphes confirment les observations précédentes. L'algorithme OEP(*UP1,Ang,**Mixte**,-*) présente des résultats intermédiaires entre OEP(*UP1,Ang,**FIFO**,-*) et OEP(*UP1,Ang,**LIFO**,-*). Ces résultats sont prévisibles car, par rapport à l'algorithme OEP(*UP1,Ang,**LIFO**,-*), l'algorithme OEP(*UP1,Ang,**Mixte**,-*) perd en intensification et gagne en diversification, ce qui défavorise l'ensemble d'instances *J30*. Quant aux ensembles *J60* et *J120*, elles récupèrent un peu de la diversification qu'elles avaient perdue avec l'algorithme OEP(*UP1,Ang,**LIFO**,-*). En ce qui concerne l'ensemble d'instances *J60*, l'algorithme OEP(*UP1,Ang,**Mixte**,-*) a certes obtenu les meilleures déviations minimales observées jusque-là mais l'algorithme OEP(*UP1,Ang,**FIFO**,-*) montre une meilleure stabilité et de meilleures moyennes aussi bien pour *1000* que pour *5000* évaluations de solutions.

Bien que nous ne pouvons pas affirmer avec ces données que l'un des algorithmes surpasse l'autre, les algorithmes retenus pour la méthode d'optimisation OEP sur le RCPSP sont OEP(*UP1,Ang,LIFO,-*) pour l'ensemble d'instances *J30* et OEP(*UP1,Ang,FIFO,-*) pour les ensembles *J60* et *J120*. Une comparaison des résultats obtenus par rapport à ceux existants dans la littérature est effectuée à la prochaine section.

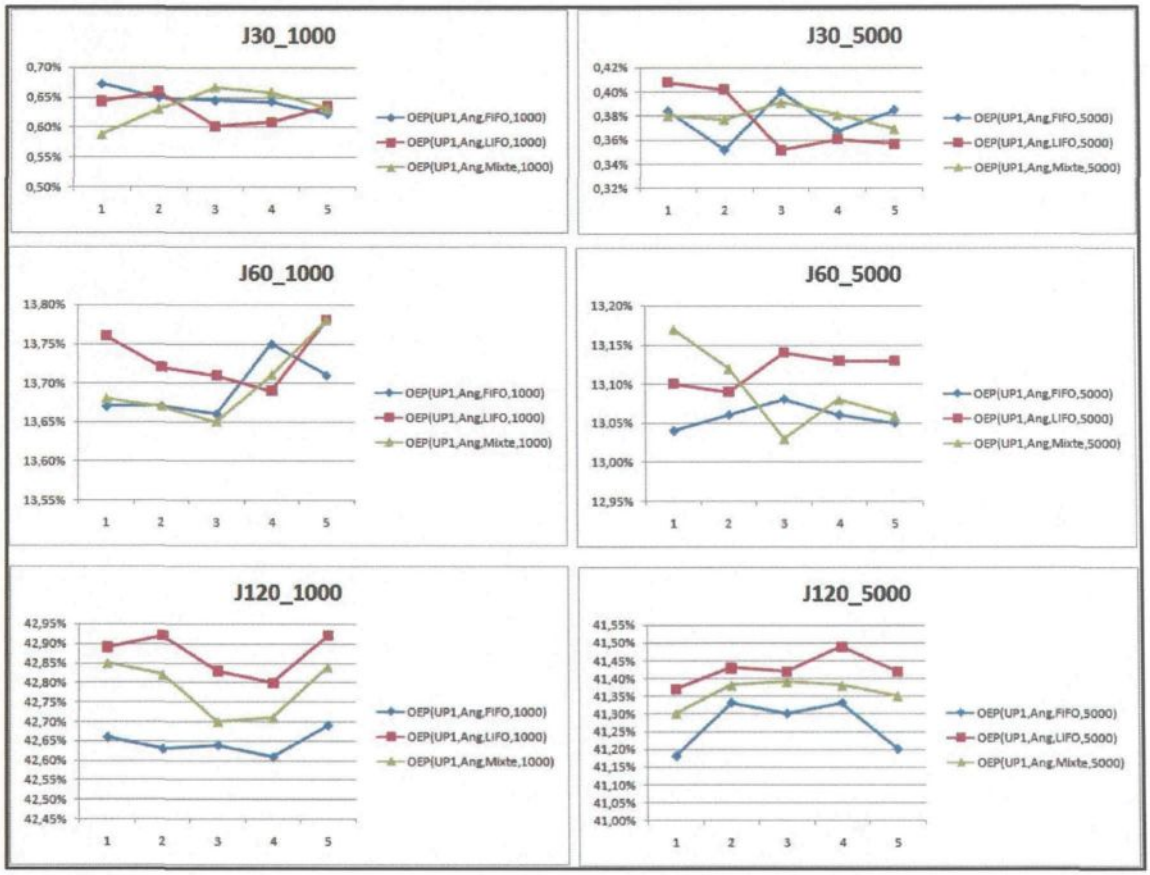


Figure 4.11 : Graphes de comparaison des résultats des algorithmes OEP utilisant les listes FIFO, LIFO et Mixte

4.2.3 Comparaison avec les algorithmes OEP proposés par Zhang *et al.* [2005]

Zhang *et al.* [2005] sont les seuls qui, à notre connaissance, ont résolu le RCPSP à l'aide de l'OEP. Les auteurs ont proposé deux algorithmes qui diffèrent l'un de l'autre par la représentation de la particule. Ils ont utilisé la représentation par valeurs réelles et la représentation par permutation, en adaptant la méthode de croisement *PMX* emprunté à l'AG pour définir les concepts de vitesse et de mise à jour d'une particule. Pour générer la population initiale, ils ont utilisé la règle de priorité LFT avec l'algorithme série pour

construire une solution à partir d'une séquence d'activités. Pour présenter leurs résultats, les auteurs ont effectué *1000*, *2000*, *3000*, *4000* et *5000* évaluations de solutions sur l'ensemble d'instances *J30* avec une population de *40* particules quelque soit le nombre de solutions évaluées. Ils n'ont pas résolu les ensembles *J60* et *J120*.

L'algorithme OEP(*UP1,Ang,LIFO,*-) utilise aussi la représentation de solutions par permutation. Afin d'assurer une base de comparaison valide, la génération de la population initiale ainsi que la construction d'une solution à partir d'une séquence d'activités est effectuée de la même manière que dans les algorithmes de Zhang *et al.* [2005]. La taille de la population a été définie à *40* particules quelque soit le nombre de solutions évaluées, en conformité avec les auteurs. La différence entre l'algorithme OEP(*UP1,Ang,LIFO,*-) et l'algorithme de Zhang *et al.* [2005] utilisant la représentation par permutation réside donc dans les concepts de vitesse et de mise à jour de la particule.

Les résultats obtenus par l'algorithme OEP(*UP1,Ang,LIFO,*-) pour l'ensemble *J30* sont présentés dans le Tableau 4.7. Ces résultats sont présentés pour cinq exécutions de l'algorithme effectuées pour chaque paramétrage du nombre de solutions évaluées. La première ligne du tableau spécifie le nombre de solutions évaluées. De la deuxième à la sixième ligne du tableau, on retrouve les déviations à l'optimum obtenues pour chaque exécution de l'algorithme. La dernière ligne du tableau contient les moyennes de ces cinq déviations.

Tableau 4.7 : Résultats obtenus par l'algorithme OEP(*UP1,Ang,LIFO,-*) sur l'ensemble d'instances *J30* par rapport au nombre de solutions évaluées

Nombre d'évaluations de solutions	1000	2000	3000	4000	5000
1	0,644%	0,516%	**0,426%**	0,415%	0,408%
2	0,660%	0,496%	0,468%	0,438%	0,402%
3	**0,602%**	0,488%	0,428%	0,378%	**0,352%**
4	0,609%	**0,462%**	0,440%	0,425%	0,361%
5	0,635%	0,511%	0,442%	**0,376%**	0,357%
Moyenne	**0,630%**	**0,495%**	**0,441%**	**0,406%**	**0,376%**

Ce tableau montre une certaine variabilité au niveau des résultats obtenus. Les résultats moyens obtenus sont utilisés pour réaliser la comparaison avec les résultats obtenus par les algorithmes de Zhang *et al.* [2005].

Le Tableau 4.8 présente donc la comparaison entre les résultats moyens obtenus par l'algorithme OEP(*UP1,Ang,LIFO,-*) et les résultats des deux algorithmes de Zhang *et al.* [2005]. Le nombre d'évaluations de solutions correspondant à chaque ligne du tableau est donné dans la première colonne. Dans la deuxième et la troisième colonne se trouvent les résultats de Zhang *et al.* [2005] auxquels l'algorithme OEP(*UP1,Ang,LIFO,-*) est comparé, respectivement pour la représentation de solutions par valeurs réelles et la représentation par permutation. La quatrième colonne présente les résultats moyens obtenus par l'algorithme OEP(*UP1,Ang,LIFO,-*).

Tableau 4.8 : Tableau comparatif de l'algorithme OEP(*UP1,Ang,LIFO,-*) avec ceux de Zhang *et al.* [2005]

Nombre d'évaluations de solutions	Zhang *et al.* Clés aléatoires	Zhang *et al.* Permutation	OEP(*UP1,Ang,LIFO,-*)
1000	0,92%	0,69%	0,630%
2000	0,83%	0,61%	0,495%
3000	0,75%	0,53%	0,441%
4000	0,67%	0,45%	0,406%
5000	0,61%	0,42%	0,376%

Ce tableau montre de meilleurs résultats pour l'algorithme OEP(*UP1,Ang,LIFO,-*) que pour les algorithmes de Zhang *et al.* [2005], quelque soit le nombre d'évaluations de solutions. Une comparaison visuelle de ces résultats est présentée par le graphe de la Figure 4.12.

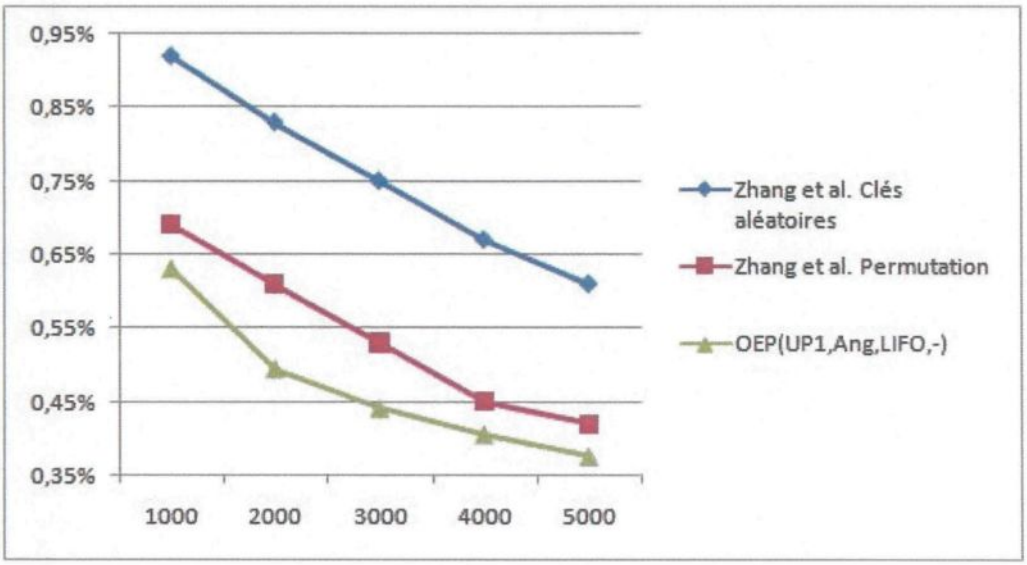


Figure 4.12 : Graphe de comparaison de l'algorithme OEP(*UP1,Ang,LIFO,-*) avec ceux de Zhang *et al.* [2005].

On remarque à l'aide de cette figure que l'algorithme OEP(*UP1,Ang,LIFO,-*) performe nettement mieux que les deux algorithmes de Zhang *et al.* [2005] sur l'ensemble d'instances *J30* et ce, quel que soit le nombre d'évaluations de solutions utilisé. On peut donc en déduire que les concepts de vitesse et de mise à jour de la particule utilisés par l'algorithme OEP(*UP1,Ang,LIFO,-*) sont plus performants pour résoudre les instances de petite taille du RCPSP que la méthode de croisement *PMX*. Aucune conclusion ne peut être tirée pour le moment, par rapport aux ensembles *J60* et *J120* car il n'existe aucun résultat disponible à ce sujet dans la littérature pour des algorithmes OEP.

4.3 Reproduction d'un AG avec croisement à deux points (AG^{2P})

Afin d'étudier et de comparer les performances de nos éventuelles propositions d'hybridation, un AG de base ne comportant pas d'autres procédures d'intensification est nécessaire. L'algorithme d'Alcaraz et Maroto [2006] fait partie des meilleurs AG de la littérature sur le RCPSP qui fournit des résultats sans la recherche locale. C'est la raison qui a motivé notre choix pour cet algorithme.

4.3.1 Description de l'algorithme

Comme il a été expliqué au chapitre précédent, les AG suivent, en général, un principe commun : une phase de génération de la population initiale suivie de phases de production de nouvelles générations successives grâce aux opérateurs génétiques de sélection, de croisement et de mutation. Chaque génération produit, en principe, un ordonnancement d'activités meilleur que les précédents. Plus l'algorithme évolue, plus les

solutions s'améliorent et on devrait obtenir à la fin une solution de bonne qualité, même si l'optimalité ne peut être garantie.

Pendant le processus d'évolution de l'algorithme proposé, deux populations existent en parallèle : la population courante de taille *N*, aussi appelée population *Parent*, et la nouvelle génération en voie de réalisation ou population *Enfant*, elle aussi de taille *N*. La Figure 4.13 présente le pseudo-code de l'algorithme AG^{2P}. Les différentes étapes de ce pseudo-code sont détaillées à la suite de la figure.

```
Générer la population initiale.
Evaluer la population initiale.
Conserver la meilleure solution dans la mémoire globale.
Tant que nbreGénérations>=Gencourante
        Pour i allant de 1 à N/2
                Sélectionner aléatoirement 2 parents P1 et P2 dans la population Parent.
                Si P_croi (probabilité de croisement)
                        Croiser P1 avec P2 -> E1.
                        Croiser P2 avec P1 -> E2.
                        Évaluer E1 et E2.
                        Mettre à jour la mémoire globale.
                        Placer E1 et E2 dans la population Enfant.
                Sinon
                        Placer P1 et P2 dans la population Enfant.
                Fin Si
        Fin Pour
        Muter chaque individu de la population Enfant (probabilité de mutation P_mut sur
        chaque gène de l'individu).
        Évaluer les solutions obtenues.
        Mettre à jour la mémoire globale.
        Remplacer la population Parent par la population Enfant.
        Incrémenter la génération courante.
Fin Tant que
Retourner la meilleure solution trouvée
```

Figure 4.13 : Pseudo code de l'algorithme AG^{2P}

4.3.1.1 La codification de la solution

La particularité de cet algorithme se trouve dans la codification de la solution. La représentation par permutation est utilisée, mais deux gènes supplémentaires sont ajoutés à

la fin de chaque solution. Ces deux gènes servent à déterminer la méthode utilisée pour générer un ordonnancement à partir d'une séquence d'activités donnée. Le premier gène, nommé *s/p* détermine s'il faut utiliser l'algorithme série ou l'algorithme parallèle discutés à la Section 2.3.2. Le second gène, quant à lui nommé *f/b*, détermine respectivement le choix d'un ordonnancement 'avant' ou 'arrière'. Un ordonnancement 'avant' consiste à placer les activités au plus tôt en partant de la première activité du projet, alors qu'un ordonnancement 'arrière' consiste à partir de la dernière activité et à les ordonnancer au plus tard. Chaque séquence d'activités pourrait donc produire quatre ordonnancements différents dépendamment des gènes *s/p* et *f/b* qui lui sont attribués. Les contraintes de ressources sont prises en compte lors de la génération d'un ordonnancement à partir d'une séquence d'activités, par l'algorithme série ou parallèle utilisé.

4.3.1.2 La génération de la population initiale

Les individus de la population initiale sont générés par une construction progressive probabiliste. Le choix du prochain gène dans une solution se fait en respectant les contraintes de préséance, de façon proportionnelle avec la règle de priorité LFT. Les deux gènes *s/p* et *f/b* sont initialisés aléatoirement.

4.3.1.3 Les opérateurs génétiques

Pour choisir les parents qui sont croisés, la sélection 2-tournoi avec remise est utilisée. Deux parents sont sélectionnés pour aller au croisement et produisent deux enfants. Cette opération est répétée *N/2* fois afin d'obtenir les *N* chromosomes de la population *Enfant*.

Pour la reproduction, le croisement à deux points de Reeves [1995] pour la représentation par permutation est utilisé. Cette méthode de croisement a été adaptée par Alcaraz et Maroto [2006] pour convenir à leur nouvelle codification. Selon cette nouvelle méthode de croisement, l'enfant hérite des gènes *s/p* et *f/b* du premier parent. Dans le cas où ce dernier est en ordonnancement 'arrière', le croisement se fait aussi en arrière, i.e. à partir du dernier gène du premier parent. Le croisement se fait selon une probabilité P_{crois}. Les parents qui ne sont pas croisés vont directement dans la population *Enfant*.

La Figure 4.14 montre un exemple du croisement étendu à deux points conçu par les auteurs. Les deux points de croisement, *2* et *6*, sont générés aléatoirement. Les quatre scénarios montrent quatre différentes configurations des gènes *s/p* et *f/b* pour le premier parent. Dans le premier scénario, le premier parent est en ordonnancement 'avant' et l'algorithme série est utilisé. Les gènes de ce parent sont copiés dans l'enfant jusqu'au premier point de coupure. Les gènes du second parent sont ensuite copiés dans l'enfant, à partir du début et sans répétition, jusqu'au second point de coupure. Enfin, les gènes restants sont recopiés dans l'enfant dans l'ordre du premier parent. L'enfant est, comme le premier parent, en ordonnancement 'avant' et utilise l'algorithme série pour produire un ordonnancement à partir de sa séquence d'activités. Les gènes *s/p* et *f/b* du second parent n'ont aucune influence sur l'enfant produit. Le deuxième scénario montre le premier parent en ordonnancement 'arrière' avec l'algorithme série. L'enfant hérite de ces deux gènes. Les gènes du premier parent sont ensuite recopiés dans l'enfant, cette fois-ci à partir de la fin jusqu'au second point de croisement, i.e. le point *6*. Les gènes du second parent sont ensuite copiés dans l'enfant, à partir de la fin et sans répétition, jusqu'au point *2*. Enfin, les

gènes restants sont recopiés dans l'enfant dans l'ordre du premier parent. Dans les scénarios 3 et 4, le premier parent utilise l'algorithme parallèle. On remarque que l'algorithme utilisé (série ou parallèle) n'influence pas la séquence d'activités produite par le croisement. Celle-ci dépend uniquement du type d'ordonnancement ('avant' ou 'arrière') utilisé.

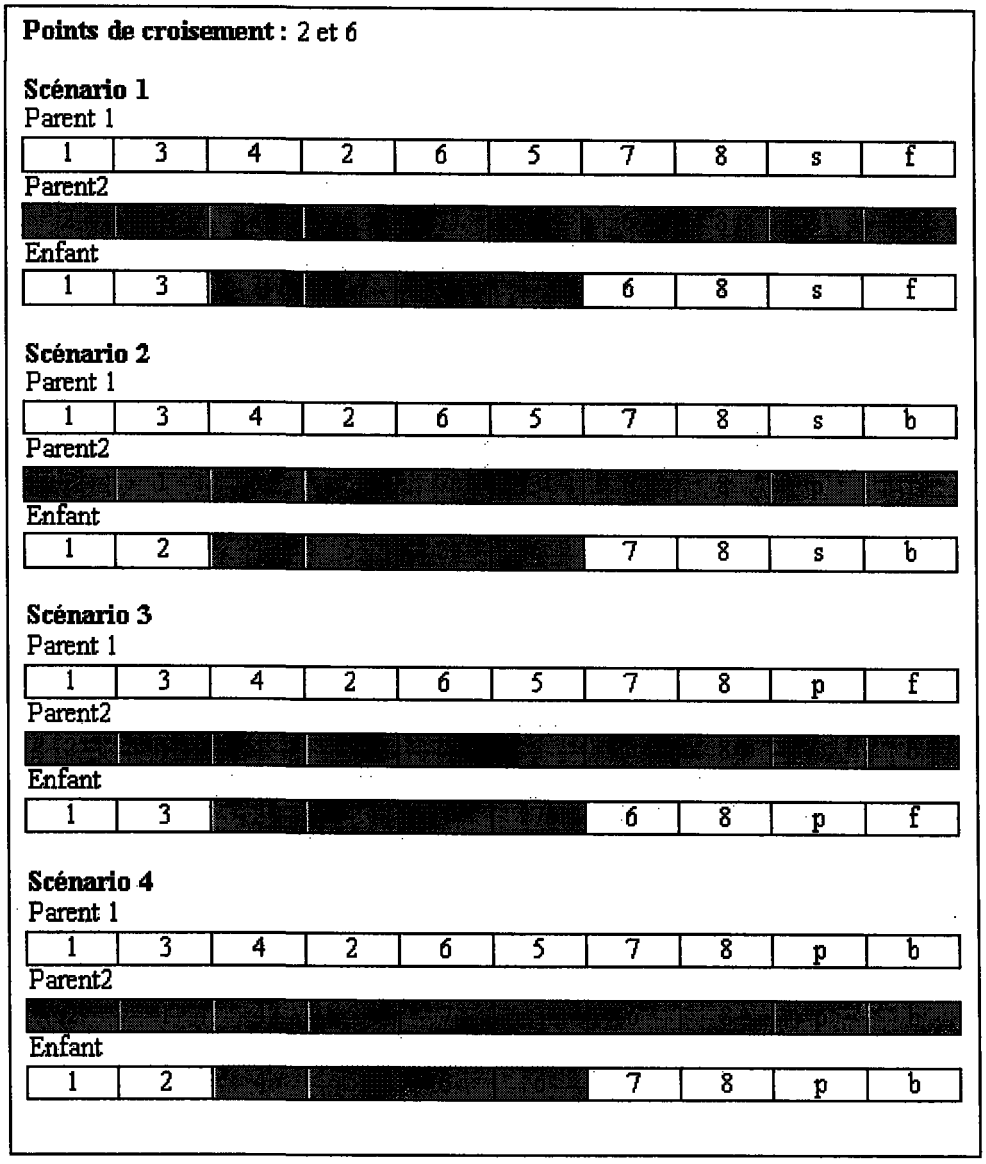


Figure 4.14 : Illustration du croisement étendu à deux points

La méthode de mutation utilisée est une extension de la mutation par insertion introduite par Boctor [1996]. Pour chaque activité de la séquence et selon une probabilité P_{mut} donnée, on choisit aléatoirement une nouvelle position et on l'y insère. Dans le but de maintenir le respect des contraintes de préséance, cette nouvelle position doit se situer entre le dernier prédécesseur et le premier successeur de l'activité concernée. Les deux derniers gènes, *s/p* et *f/b*, sont inversés selon cette même probabilité P_{mut}.

Afin de maintenir une bonne diversité dans la population, le remplacement se fait de façon générationnelle. À la fin de chaque génération, la population *Enfant* remplace la population *Parent* et devient la nouvelle génération courante.

4.3.2 Essais numériques et résultats obtenus

Les conditions expérimentales sont les mêmes que celles décrites dans la Section 4.2.2. L'implémentation informatique de l'algorithme a été faite en langage C++ selon une approche objet et la version 2005 du logiciel Microsoft Visual C++ a été utilisée. Toutes les expérimentations ont été effectuées sur un PC Pentium IV 3.2GHz avec 1Go de RAM et utilisant Windows XP. Ces expérimentations numériques ont été réalisées sur les ensembles d'instances *J30*, *J60* et *J120* en faisant *1000* et *5000* évaluations de solutions pour chaque instance, avec cinq exécutions pour chacune d'elles.

L'AG reproduit est nommé AG^{2P}. La probabilité de croisement est de *0,8*. La probabilité de mutation est de *0,05* pour l'ensemble *J30* et *0,01* pour les ensembles *J60* et *J120*. La taille de la population est de *50* individus lorsque *1000* solutions sont évaluées et de *100* individus lorsque *5000* solutions sont évaluées. Les résultats sont présentés au Tableau 4.9.

Ce tableau montre les déviations moyennes obtenues par exécution pour chaque ensemble d'instances. Les déviations moyennes sont calculées suivant les Équations 4.10 et 4.11. La moyenne des cinq exécutions de l'algorithme AG^{2P} ainsi que les résultats obtenus par l'algorithme d'origine d'Alcaraz et Maroto [2006], que nous désignons par l'acronyme AG^{AM} dans la suite de ce document, sont également consignés dans le Tableau 4.9. Ce tableau présente dans sa première colonne le numéro des exécutions. Dans les six autres colonnes, on retrouve respectivement, les résultats obtenus par l'algorithme AG^{2P} sur l'ensemble d'instances *J30* pour *1000* et *5000* évaluations de solutions, les résultats obtenus sur l'ensemble *J60* pour *1000* et *5000* évaluations et les résultats obtenus sur l'ensemble *J120* pour *1000* et *5000* évaluations de solutions.

Tableau 4.9 : Résultats obtenus par l'algorithme AG^{2P}

Exécutions	**J30_1000**	**J30_5000**	**J60_1000**	**J60_5000**	**J120_1000**	**J120_5000**
1	0,459%	0,235%	12,74%	12,07%	38,06%	36,05%
2	**0,425%**	0,213%	12,79%	12,02%	38,23%	36,04%
3	0,470%	0,241%	12,82%	12,02%	38,10%	**35,96%**
4	0,451%	**0,207%**	**12,69%**	12,02%	38,15%	36,03%
5	0,466%	0,254%	12,73%	**11,97%**	**38,03%**	36,03%
Moyenne	**0,454%**	**0,230%**	**12,75%**	**12,02%**	**38,11%**	**36,02%**
AG^{AM}	**0,334%**	**0,131%**	**12,41%**	**11,83%**	**37,17%**	**35,44%**

Ces résultats traduisent un écart entre les déviations moyennes obtenues par les deux algorithmes. Cet écart est beaucoup plus important au niveau de l'ensemble d'instances *J30* qu'au niveau des autres ensembles d'instances. En effet, l'écart relatif est de *36%* et *76%* pour l'ensemble *J30* et de *3%* et *2%* pour les ensembles *J60* et *J120*, respectivement pour *1000* et *5000* évaluations de solutions. Les résultats obtenus par

l'algorithme d'origine AG^{AM} sont aussi meilleurs que les déviations minimales obtenus par l'algorithme AG^{2P}.

L'algorithme AG^{2P} n'a donc pas réussi à reproduire les résultats de l'algorithme d'origine AG^{AM}. Les données de l'article, ainsi que celles du précédent article [Alcaraz et Maroto 2001] qui constitue la base de celui-ci, ont cependant été respectées au meilleur de notre connaissance. L'écart entre les algorithmes AG^{2P} et AG^{AM}, malgré la rigueur observée dans le travail effectué, peut laisser supposer que certaines informations ou détails sont manquants dans l'article ou que certains aspects ont pu être mal interprétés. Comme les algorithmes subséquents présentés au Chapitre 5 s'appuient sur l'algorithme AG^{2P}, les résultats obtenus à l'aide de ce dernier constituent la base de comparaison.

4.4 Conclusion

Dans ce chapitre, deux approches de résolution ont été présentées pour le RCPSP. La première approche est un algorithme d'OEP qui a été adapté au RCPSP. L'algorithme ayant été conçu pour résoudre un problème qui n'est pas soumis à des contraintes de préséance, la gestion de ces contraintes y a été introduite afin de construire des solutions valides pour le RCPSP. Cette approche innovatrice donne de meilleurs résultats que ceux de la littérature qui utilisent l'OEP. L'impact du facteur de constriction a été étudié sur l'algorithme, ainsi que les effets de l'intensification et de la diversification par rapport à la taille du problème traité. Les ensembles d'instances *J60* et *J120* ont été résolus pour la première fois dans la littérature à l'aide de l'OEP. Les expérimentations effectuées montrent que l'OEP tire un meilleur avantage de la diversité au sein de la population que de l'intensification. Cependant, pour un problème de petite taille, l'intensification peut aider à

aller chercher de meilleures solutions. Il reste néanmoins un pas important à franchir pour espérer rivaliser avec les meilleurs algorithmes de la littérature pour le RCPSP.

La seconde approche présentée est un AG qui a déjà donné de bons résultats dans la littérature et qui a été reproduit au meilleur de notre connaissance. Les résultats obtenus ne reflètent pas exactement ceux de l'algorithme d'origine, bien que les informations fournies par l'article aient été exploitées avec rigueur. Dans le Chapitre 5, nous procédons à l'hybridation de l'AG et de l'OEP selon différents schémas.

CHAPITRE 5

HYBRIDATION DE L'ALGORITHME GÉNÉTIQUE ET DE L'OPTIMISATION PAR ESSAIMS PARTICULAIRES

5.1 Introduction

Une manière d'améliorer les performances d'un algorithme ou de combler certaines de ses lacunes consiste à le combiner avec une autre méthode [Talbi, 2000]. Ce principe général, appelé hybridation, peut s'appliquer pour un grand nombre de méthodes. La tendance actuelle est l'émergence de ces méthodes hybrides, qui s'efforcent de tirer parti des avantages d'approches différentes en les combinant [Lova *et al.* 2009; Shan *et al.* 2007; Valls *et al.* 2008].

Dans ce chapitre, une nouvelle forme d'hybridation entre les AG et l'OEP est présentée. L'algorithme d'OEP conçu au chapitre précédent est exploité pour concevoir une méthode de croisement utilisée dans la reproduction de l'AG réalisée, elle-aussi, au chapitre précédent. En effet, le principe de l'OEP est semblable à celui d'un croisement dans le sens où l'algorithme combine plusieurs solutions pour en obtenir une nouvelle.

Le présent chapitre est subdivisé en deux parties. Dans la première partie, la nouvelle méthode de croisement de type OEP proposée est présentée ainsi que les résultats obtenus en la substituant à la méthode de croisement à deux points dans l'algorithme AG^{2P}. Comme dans le chapitre précédent, il a aussi été procédé par étapes en modifiant les paramètres de l'algorithme et en sélectionnant ceux qui offrent les meilleurs résultats pour la suite. Les paramètres concernés sont les paramètres de calibration de l'OEP à savoir le facteur d'inertie et les facteurs de connaissance, le type de liste utilisé par la procédure de correction de la pseudo-séquence, la taille de la population ainsi que le nombre de générations de l'algorithme. Dans la seconde partie du chapitre, la méthode de croisement

initiale à deux points est réintroduite en combinaison avec le croisement de type OEP. Des expérimentations numériques sont réalisées afin de déterminer le niveau de contribution de chacune de ces deux méthodes de croisement.

5.2 Conception d'un croisement de type OEP pour l'AG

L'algorithme d'OEP présenté au Chapitre 4, a démontré sa compétitivité pour la résolution de l'ensemble d'instances *J30* et les résultats obtenus sont assez prometteurs. Le principe de l'OEP présentant des similarités avec le croisement des AG, il semble judicieux de l'appliquer dans la structure d'évolution d'un AG. L'OEP pour la résolution des problèmes à variables discrètes est une méthode émergeante. Il n'existe, à notre connaissance, aucun auteur qui l'utilise pour traiter les ensembles d'instances *J60* et *J120*. Les expérimentations numériques ont donc aussi été effectuées sur ces ensembles afin de vérifier le comportement des algorithmes proposés.

5.2.1 Description du croisement de type OEP

Le croisement de type OEP est conçu suivant le principe de l'OEP qui combine les propriétés de trois particules afin d'en construire une nouvelle. Un vecteur de vitesse initialisé suivant la méthode décrite pour l'algorithme OEP du chapitre précédent est associé à chaque chromosome. La meilleure solution globale g de l'algorithme représente le troisième chromosome du croisement. Les deux autres chromosomes, nommés P_1 et P_2, sont sélectionnés comme dans l'algorithme AG^{2P}, à l'aide d'une sélection 2-tournoi avec remise.

Trois chromosomes parents P_1, P_2 et g s'associent donc pour produire un chromosome enfant E_1. La vitesse $V_1(t)$ du parent P_1 au temps t est d'abord mise à jour suivant l'Équation 4.1 de calcul de la vitesse d'une particule. Dans le croisement de type OEP, le parent P_1 désigne la position initiale de la particule et la meilleure position de la particule est remplacée par le parent P_2. La nouvelle formule de calcul de la vitesse ainsi obtenue à l'itération t est donnée par l'Équation 5.1. Comme dans l'OEP, w représente le facteur d'inertie, c_1 et c_2 sont les coefficients de connaissance, et r_1 et r_2 sont des nombres aléatoires tirés uniformément dans l'intervalle *[0, 1]*.

$$V_1(t)=w.V_1(t-1)+c_1.r_1.(P_2-P_1)+c_2.r_2.(g-P_1) \qquad \textit{(Équation 5.1)}$$

Cette vitesse $V_1(t)$ est ensuite utilisée avec le parent P_1 pour générer l'enfant E_1 suivant l'alternative de mise à jour *UP1* sélectionnée au Chapitre 4. Les opérateurs mathématiques utilisés sont ceux décrits pour l'algorithme OEP présenté au chapitre précédent. L'enfant E_1 hérite de la nouvelle vitesse du parent P_1. Le second chromosome enfant E_2 est produit en inversant l'ordre des parents P_1 et P_2. Après le croisement, les chromosomes 'enfants' obtenus sont mutés et l'algorithme continue tant que le critère d'arrêt n'est pas atteint.

5.2.2 Essais numériques et résultats obtenus

Les conditions expérimentales sont les mêmes que celles décrites au Chapitre 4. Les tests ont été réalisés sur les ensembles d'instances *J30*, *J60* et *J120* en faisant *1000* et *5000* évaluations de solutions. Les algorithmes présentés dans cette section sont nommés suivant le modèle $AG^{OEP}(a,b,c)$. L'élément '*a*' représente les paramètres liés à la calibration de

l'OEP. *Ang* est utilisé pour désigner les paramètres d'Anghinolfi et Paolucci [2009] ($w=0,5$, $c_1=1,5$, $c_2=2$), et χ est utilisé pour les paramètres liés au facteur de constriction ($w=0,7$, $c_1=1,5$, $c_2=1,5$). L'élément '*b*' désigne le type de liste utilisé par la procédure de correction de la pseudo-séquence. Comme au Chapitre 4, la liste utilisée est soit une liste FIFO, soit une liste LIFO, ou encore une liste mixte. L'élément '*c*' se présente sous la forme *N*x*G* où *N* désigne la taille de la population et *G* le nombre de générations. Le symbole '-' est utilisé pour désigner un paramètre qui varie pour le même algorithme. Ainsi, un AG qui utilise le croisement de type OEP avec les paramètres d'Anghinolfi et Paolucci [2009], une liste de type FIFO et une population de *50* individus pendant *20* générations, est nommé AG^{OEP}(*Ang,FIFO,50x20*). Lorsque deux configurations de l'algorithme sont comparées, le paramètre introduisant la différence entre ces deux configurations est spécifié en gras dans le nom de l'algorithme.

Les ensembles d'instances *J30*, *J60*, et *J120* ont été résolus en utilisant comme critère d'arrêt le nombre d'évaluations de solutions : *1000* et *5000* solutions ont été évaluées pour chaque instance, en effectuant cinq exécutions à chaque fois. Les paramètres liés à la calibration du croisement de type OEP sont ceux retenus au Chapitre 4 pour l'algorithme OEP, i.e. les paramètres d'Anghinolfi et Paolucci [2009]. La liste utilisée par la procédure de correction d'une pseudo-séquence est une liste de type FIFO. L'algorithme de départ est donc nommé AG^{OEP}(*Ang,FIFO,-*). L'algorithme AG^{OEP}(*Ang,FIFO,-*) diffère de l'algorithme AG^{2P} uniquement par le type de croisement utilisé. Dans le but d'obtenir la comparaison la plus équitable possible, les paramètres liés à la taille de la population et les probabilités de croisement et de mutation sont les mêmes que pour l'algorithme AG^{2P}. La

taille de la population est donc de *50* individus lorsque *1000* solutions sont évaluées et *100* individus lorsque *5000* solutions sont évaluées. La probabilité de croisement est de *0,80*. La probabilité de mutation est de *0,05* pour l'ensemble d'instances *J30* et *0,01* pour les ensembles *J60* et *J120*.

Les résultats de l'algorithme AG^{OEP}(*Ang, FIFO*, -) sont présentés au Tableau 5.1. Ce tableau présente dans sa première ligne les paramètres utilisés pour l'algorithme. Dans la deuxième ligne, les noms des ensembles résolus sont précisés. De la troisième à la septième ligne sont présentées les déviations moyennes obtenues pour chaque exécution de l'algorithme sur l'ensemble d'instances considéré. La huitième ligne du tableau présente la moyenne des cinq exécutions et la neuvième ligne rappelle les résultats obtenus avec l'algorithme AG^{2P}.

Tableau 5.1 : Résultats obtenus par l'algorithme AG^{OEP}(*Ang,FIFO,*-)

	AG^{OEP}(*Ang,FIFO,50x20*)			AG^{OEP}(*Ang,FIFO,100x50*)		
Instances	J30	J60	J120	J30	J60	J120
1	0,475%	13,07%	**39,79%**	0,265%	12,37%	37,98%
2	0,386%	13,12%	39,90%	0,257%	**12,35%**	37,94%
3	0,454%	**13,02%**	39,87%	0,253%	12,46%	**37,89%**
4	**0,355%**	13,02%	39,81%	**0,248%**	12,37%	37,89%
5	0,440%	13,04%	39,92%	0,256%	12,35%	37,89%
Moyenne	**0,422%**	**13,05%**	**39,86%**	**0,256%**	**12,38%**	**37,92%**
AG^{2P}	**0,454%**	**12,75%**	**38,11%**	**0,230%**	**12,02%**	**36,02%**

On observe à travers ces résultats de bonnes performances pour l'algorithme AG^{OEP}(*Ang,FIFO,*-) sur l'ensemble d'instances *J30* à *1000* évaluations de solutions, aussi bien au niveau de la moyenne que de la déviation minimale observée. Au niveau de l'ensemble *J30* à *5000* évaluations de solutions et pour les ensembles *J60* et *J120*, on

constate une dégradation des solutions par rapport à l'algorithme AG^{2P}. Ceci pourrait être dû à une réduction de l'intensification au sein de l'AG, provoquée par le nouveau croisement.

Une représentation visuelle de ces résultats est donnée par les graphes de la Figure 5.1. Ces graphes superposent les résultats obtenus par l'algorithme AG^{OEP}(*Ang,FIFO,-*) à ceux précédemment obtenus avec l'algorithme AG^{2P}. Les titres des graphes désignent l'ensemble d'instances considéré et le nombre d'évaluations de solutions effectuées.

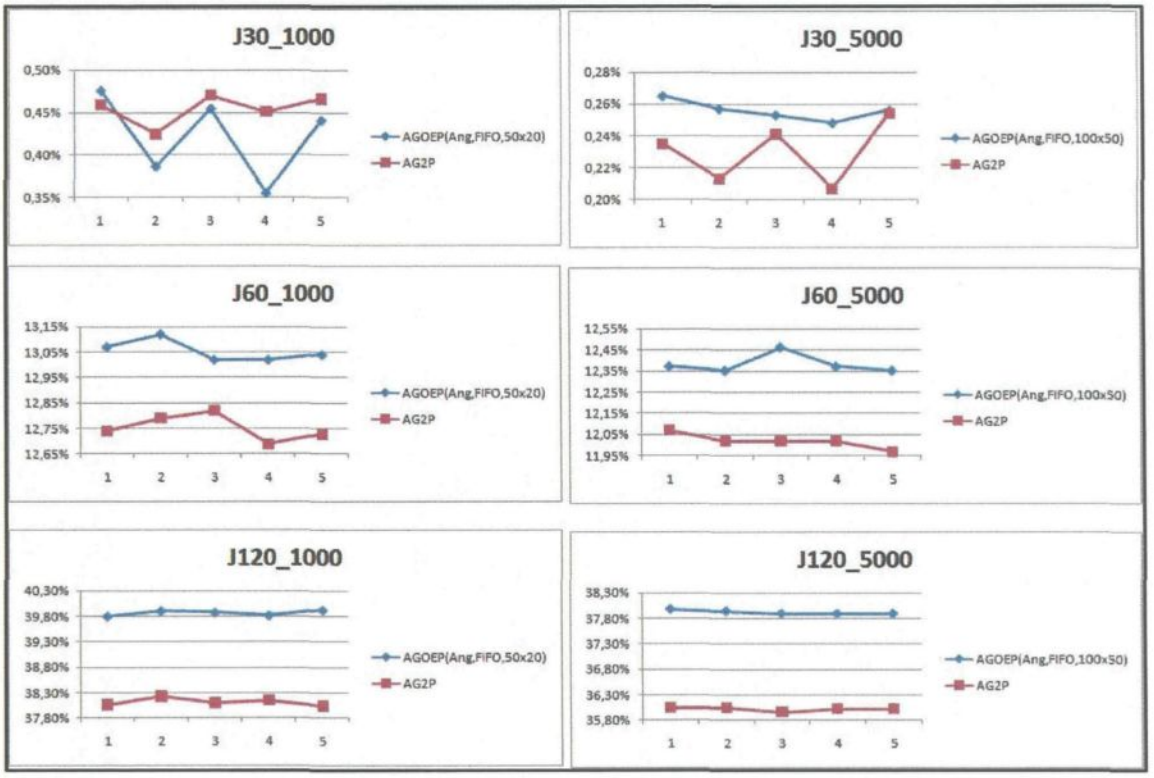


Figure 5.1 : Graphes de comparaison des résultats des algorithmes AG^{OEP}(*Ang,FIFO,-*) et AG^{2P}.

Ces graphes confirment les observations précédentes. En effet, la courbe de l'algorithme $AG^{OEP}(Ang,FIFO,-)$ se situe toujours au-dessus de celle de l'algorithme AG^{2P}, sauf dans le cas où *1000* solutions sont évaluées pour l'ensemble *J30*.

L'opérateur de croisement est celui qui contribue le plus à la convergence d'un AG en se basant sur l'exploitation des solutions. Afin d'améliorer les performances de l'algorithme, les paramètres de calibration de cet opérateur ont donc été modifiés suivant le facteur de constriction comme dans l'algorithme $OEP(UP1,\chi,FIFO,-)$ mentionné au Chapitre 4. Les paramètres $w=0,7$, $c_1=c_2=1,5$ ont donc été utilisés pour le nouvel algorithme ainsi nommé $AG^{OEP}(\chi,FIFO,-)$.

Les résultats obtenus par l'algorithme $AG^{OEP}(\chi,FIFO,-)$ sont présentés au Tableau 5.2. Ce tableau est structuré de la même manière que le tableau précédent. La dernière ligne du tableau présente les résultats de l'algorithme $AG^{OEP}(Ang,FIFO,-)$.

Tableau 5.2 : Résultats obtenus par l'algorithme $AG^{OEP}(\chi,FIFO,-)$

	$AG^{OEP}(\chi,FIFO,50x20)$			$AG^{OEP}(\chi,FIFO,100x50)$		
Instances	J30	J60	J120	J30	J60	J120
1	0,427%	13,03%	**39,66%**	0,259%	**12,34%**	**37,73%**
2	0,418%	13,06%	39,89%	**0,237%**	12,35%	37,84%
3	0,452%	**12,99%**	39,89%	0,243%	12,40%	37,84%
4	0,435%	13,10%	39,87%	0,244%	12,43%	37,77%
5	**0,415%**	13,08%	39,79%	0,246%	12,35%	37,85%
Moyenne	**0,429%**	**13,05%**	**39,82%**	**0,246%**	**12,37%**	**37,81%**
$AG^{OEP}(Ang,FIFO,-)$	**0,422%**	**13,05%**	**39,86%**	**0,256%**	**12,38%**	**37,92%**

On observe dans ce tableau de légères améliorations sur les résultats obtenus par rapport à l'algorithme AG^{OEP}(***Ang***,*FIFO*,-). Une baisse de performance est néanmoins observée pour *1000* évaluations de solutions sur l'ensemble *J30*.

Les graphes de la Figure 5.2 traduisent les résultats du Tableau 5.2. Une superposition de ces résultats est faite par rapport aux résultats de l'algorithme AG^{OEP}(***Ang***,*FIFO*,-).

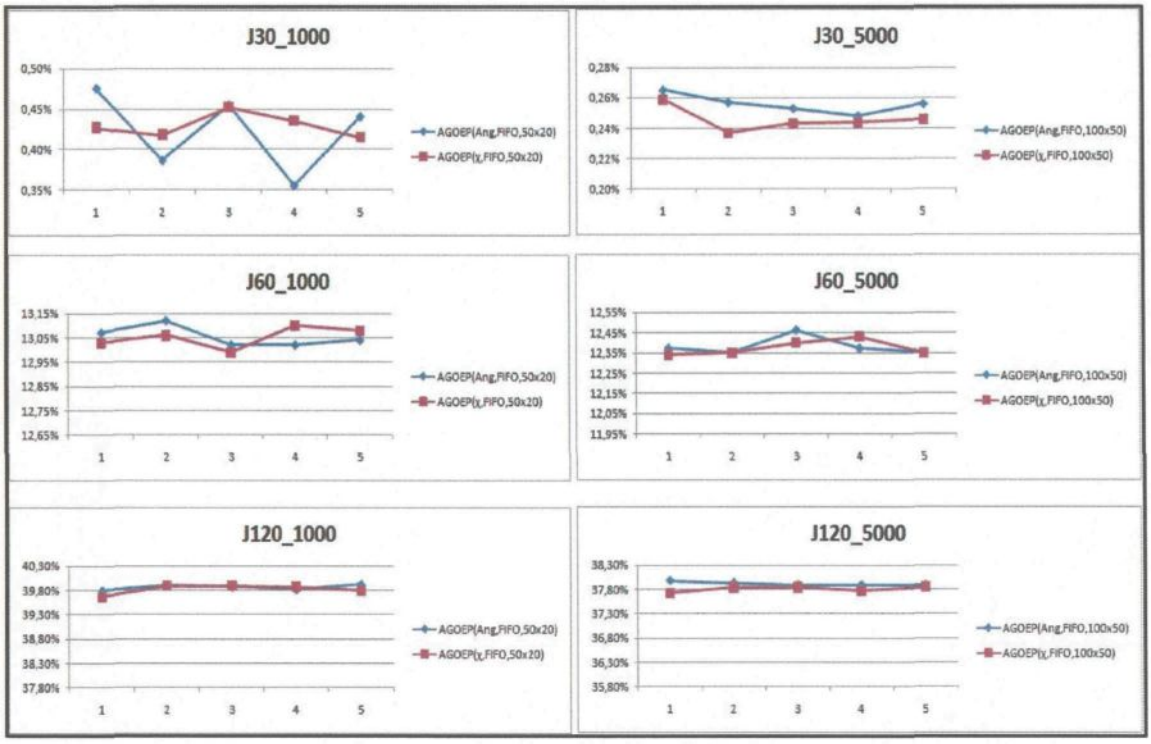


Figure 5.2 : Graphes de comparaison des résultats des algorithmes AG^{OEP} par rapport à l'utilisation du facteur de constriction

On note sur ces graphes une amélioration de la stabilité de l'algorithme surtout au niveau de l'ensemble d'instances *J30* à *1000* évaluations de solutions. Les minimums observés pour l'algorithme AG^{OEP}(***χ***,*FIFO*,-) sont aussi inférieurs pour la plupart à ceux de

l'algorithme $AG^{OEP}(\boldsymbol{Ang,FIFO,-})$. La modification a donc été retenue même si un test statistique devrait être appliqué pour comparer les résultats.

La liste FIFO utilisée lors de la correction d'une pseudo-séquence est ensuite remplacée par une liste LIFO comme dans l'algorithme OEP(*UP1,Ang,LIFO,-*) présenté au Chapitre 4. L'algorithme ainsi modifié est appelé $AG^{OEP}(\chi,\boldsymbol{LIFO,-})$. Le Tableau 5.3 présente les résultats obtenus par cet algorithme. Ce tableau présente la même structure que les tableaux précédents. La dernière ligne du tableau rappelle les résultats de l'algorithme $AG^{OEP}(\chi,\boldsymbol{FIFO,-})$.

Tableau 5.3 : Résultats obtenus par l'algorithme $AG^{OEP}(\chi,LIFO,-)$

	$AG^{OEP}(\chi,\boldsymbol{LIFO},50x20)$			$AG^{OEP}(\chi,\boldsymbol{LIFO},100x50)$		
Instances	J30	J60	J120	J30	J60	J120
1	0,430%	13,16%	39,76%	**0,213%**	12,40%	**37,79%**
2	0,412%	**13,04%**	39,82%	0,252%	**12,30%**	37,92%
3	0,489%	13,10%	39,79%	0,236%	12,34%	37,80%
4	**0,397%**	13,05%	**39,75%**	0,242%	12,35%	37,83%
5	0,502%	13,10%	39,93%	0,238%	12,37%	37,86%
Moyenne	**0,446%**	**13,09%**	**39,81%**	**0,236%**	**12,35%**	**37,84%**
$AG^{OEP}(\chi,\boldsymbol{FIFO,-})$	**0,429%**	**13,05%**	**39,82%**	**0,246%**	**12,37%**	**37,81%**

La comparaison n'est pas évidente entre les deux algorithmes. Les résultats sont plus ou moins équivalents dans la plupart des cas à ceux obtenus avec l'algorithme $AG^{OEP}(\chi,\boldsymbol{FIFO,-})$. Les déviations moyennes obtenues pour chaque ensemble d'instances se dégradent ou s'améliorent une fois sur deux. Une comparaison visuelle de ces résultats avec ceux de l'algorithme $AG^{OEP}(\chi,\boldsymbol{FIFO,-})$ est présentée par les graphes de la Figure 5.3.

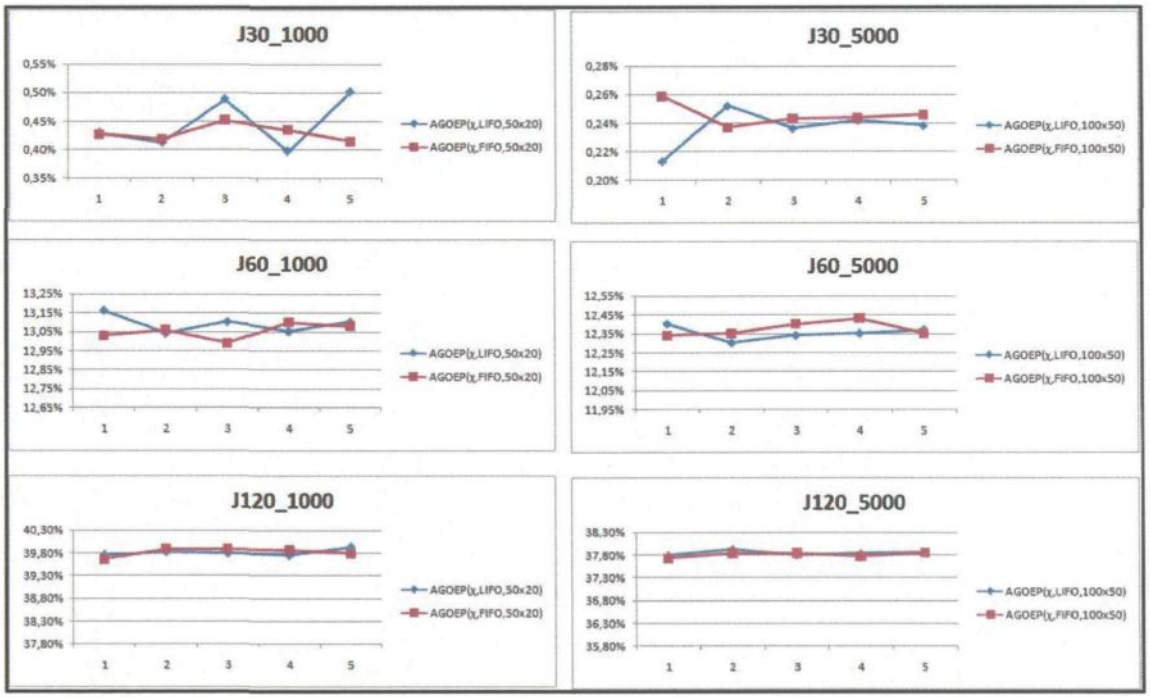


Figure 5.3 : Graphes de comparaison des résultats des algorithmes AG^{OEP} utilisant les listes FIFO et LIFO

Ces graphes montrent que l'algorithme $AG^{OEP}(\chi,\boldsymbol{LIFO},-)$ obtient trois fois sur six de meilleures déviations minimales que l'algorithme $AG^{OEP}(\chi,\boldsymbol{FIFO},-)$, notamment pour l'ensemble *J30* et lorsque *5000* solutions sont évaluées pour l'ensemble *J60*. Afin de combiner les avantages des deux types de liste, la liste mixte utilisée par l'algorithme OEP(*UP1,Ang,Mixte,*-) au Chapitre 4 est introduite dans l'algorithme. Cette méthode n'avait pas donné d'amélioration sur les résultats dans le cas de l'OEP. Toutefois, l'OEP est une méthode qui se base beaucoup plus sur la diversification alors que l'AG tire avantage de l'intensification obtenue surtout par sa méthode de croisement. L'algorithme ainsi obtenue est nommé $AG^{OEP}(\chi,\boldsymbol{Mixte},-)$.

Les résultats obtenus par l'algorithme $AG^{OEP}(\chi,\textbf{\textit{Mixte}},-)$ sont consignés dans le Tableau 5.4. Ce tableau est aussi structuré de la même manière que les tableaux précédents, avec une ligne supplémentaire à la fin. Les deux dernières lignes du tableau rappellent les résultats obtenus par les algorithmes $AG^{OEP}(\chi,\textbf{\textit{FIFO}},-)$ et $AG^{OEP}(\chi,\textbf{\textit{LIFO}},-)$ respectivement.

Tableau 5.4 : Résultats obtenues par l'algorithme $AG^{OEP}(\chi,\textit{Mixte},-)$

	$AG^{OEP}(\chi,\textbf{\textit{Mixte}},\textit{50x20})$			$AG^{OEP}(\chi,\textbf{\textit{Mixte}},\textit{100x50})$		
Instances	J30	J60	J120	J30	J60	J120
1	0,452%	12,73%	38,60%	**0,255%**	12,26%	37,40%
2	0,488%	12,77%	38,54%	0,259%	12,32%	37,36%
3	**0,396%**	**12,72%**	**38,51%**	0,258%	12,32%	37,46%
4	0,428%	12,77%	38,53%	0,267%	12,35%	**37,34%**
5	0,468%	12,86%	38,55%	0,258%	**12,19%**	37,45%
Moyenne	**0,446%**	**12,77%**	**38,55%**	**0,259%**	**12,29%**	**37,40%**
$AG^{OEP}(\chi,\textbf{\textit{FIFO}},-)$	**0,429%**	**13,05%**	**39,82%**	**0,246%**	**12,37%**	**37,81%**
$AG^{OEP}(\chi,\textbf{\textit{LIFO}},-)$	**0,446%**	**13,09%**	**39,81%**	**0,236%**	**12,35%**	**37,84%**

Cette expérience s'est révélée payante pour les instances de *60* et *120* activités qui tirent profit du compromis. Concernant les instances de *30* activités, on note une dégradation au niveau des solutions obtenues. L'algorithme $AG^{OEP}(\chi,\textbf{\textit{FIFO}},-)$ obtient la meilleure moyenne sur ces instances à *1000* évaluations de solutions et l'algorithme $AG^{OEP}(\chi,\textbf{\textit{LIFO}},-)$ obtient la meilleure moyenne à *5000* évaluations. Une comparaison graphique des résultats des trois algorithmes $AG^{OEP}(\chi,\textbf{\textit{FIFO}},-)$, $AG^{OEP}(\chi,\textit{LIFO},-)$ et $AG^{OEP}(\chi,\textit{Mixte},-)$ est présentée à la Figure 5.4.

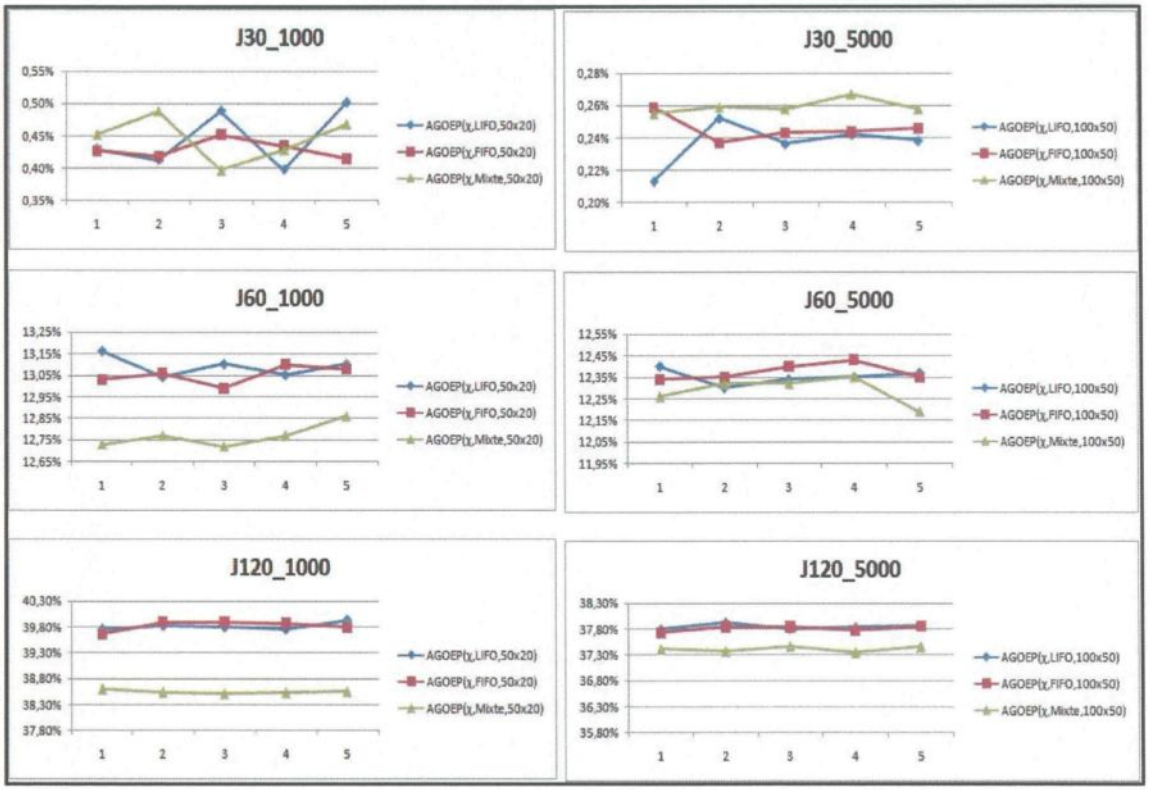


Figure 5.4 : Graphes de comparaison des résultats des algorithmes AG^{OEP} par rapport au type de liste utilisé

On remarque, à l'aide de ces graphes, que l'algorithme $AG^{OEP}(\chi,\boldsymbol{FIFO},-)$ est le plus stable parmi les trois algorithmes sur l'ensemble *J30*. L'algorithme $AG^{OEP}(\chi,\boldsymbol{LIFO},-)$ donne cependant, quatre fois sur cinq, de meilleurs résultats que les deux autres algorithmes lorsque *5000* évaluations de solutions sont effectuées. Les algorithmes $AG^{OEP}(\chi,\boldsymbol{FIFO},-)$ et $AG^{OEP}(\chi,\boldsymbol{LIFO},-)$ sont donc retenus pour résoudre l'ensemble d'instances *J30*. D'autres expérimentations numériques effectuées par la suite permettent de déterminer celui qui convient le mieux. Les graphes des ensembles *J60* et *J120* montrent une meilleure performance de l'algorithme $AG^{OEP}(\chi,\boldsymbol{Mixte},-)$ sur ces instances. L'algorithme

$AG^{OEP}(\chi,\textbf{\textit{Mixte}},-)$ est retenu pour la résolution des ensembles *J60* et *J120* même si un test statistique devrait être appliquer pour valider la comparaison.

Dans le but de continuer à affiner les paramètres de l'algorithme AG^{OEP}, trois différentes tailles de population ont été expérimentées afin de choisir celle qui convient le mieux à la nouvelle méthode de croisement. Des tests empiriques ont ainsi été effectués avec des populations de taille *30*, *40* et *50* pour *1000* évaluations de solutions, et des populations de taille *50*, *80* et *100* pour *5000* évaluations de solutions. Pour l'ensemble d'instances *J30*, les tests ont été effectués avec les algorithmes $AG^{OEP}(\chi,\textbf{\textit{FIFO}},-)$ et $AG^{OEP}(\chi,\textbf{\textit{LIFO}},-)$ afin de déterminer celui qui donne les meilleurs résultats. L'algorithme $AG^{OEP}(\chi,\textbf{\textit{Mixte}},-)$ a été utilisé pour les ensembles *J60* et *J120*. Les Tableaux 5.5, 5.6, 5.7 et 5.8 présentent les résultats obtenus respectivement pour l'ensemble d'instances *J30* avec l'algorithme $AG^{OEP}(\chi,\textbf{\textit{FIFO}},-)$, pour l'ensemble *J30* avec l'algorithme $AG^{OEP}(\chi,\textbf{\textit{LIFO}},-)$, et pour les ensembles *J60* et *J120* avec l'algorithme $AG^{OEP}(\chi,\textbf{\textit{Mixte}},-)$. Ces tableaux sont structurés de la manière suivante. Dans la première ligne, on retrouve le nom de l'algorithme utilisé. La première cellule de cette ligne rappelle l'ensemble d'instances traité. La deuxième ligne définit la taille de la population et la troisième ligne, le nombre de générations effectuées par l'algorithme. Dans la quatrième ligne des tableaux, le nombre de solutions générées par l'algorithme est calculé en faisant le produit de la taille de la population par le nombre de générations. De la cinquième à la neuvième ligne se trouvent les déviations moyennes obtenues par les cinq exécutions de l'algorithme. À la dixième ligne, la moyenne obtenue par ces cinq exécutions est calculée.

Tableau 5.5 : Résultats obtenus en variant la taille de population sur l'ensemble *J30* avec l'algorithme $AG^{OEP}(\chi,FIFO,-)$

J30	$AG^{OEP}(\chi,FIFO,-)$					
Taille de la population	**30**	**40**	**50**	**50**	**80**	**100**
Nombre de générations	**33**	**25**	**20**	**100**	**62**	**50**
Nombre de solutions	990	1000	1000	5000	4960	5000
1	0,408%	**0,367%**	0,427%	0,277%	0,219%	0,259%
2	**0,367%**	0,461%	0,418%	0,251%	0,237%	**0,237%**
3	0,464%	0,427%	0,452%	**0,226%**	0,263%	0,243%
4	0,381%	0,473%	0,435%	0,235%	**0,217%**	0,244%
5	0,430%	0,454%	**0,415%**	0,227%	0,240%	0,246%
Moyenne	**0,410%**	**0,436%**	**0,429%**	**0,243%**	**0,235%**	**0,246%**

Tableau 5.6 : Résultats obtenus en variant la taille de population sur l'ensemble *J30* avec l'algorithme $AG^{OEP}(\chi,LIFO,-)$

J30	$AG^{OEP}(\chi,LIFO,-)$					
Taille de la population	**30**	**40**	**50**	**50**	**80**	**100**
Nombre de générations	**33**	**25**	**20**	**100**	**62**	**50**
Nombre de solutions	990	1000	1000	5000	4960	5000
1	0,448%	0,476%	0,430%	0,249%	0,240%	**0,213%**
2	**0,421%**	0,446%	0,412%	0,244%	0,236%	0,252%
3	0,432%	0,424%	0,489%	**0,224%**	**0,233%**	0,236%
4	0,444%	**0,409%**	**0,397%**	0,250%	0,238%	0,242%
5	0,484%	0,429%	0,502%	0,276%	0,249%	0,238%
Moyenne	**0,446%**	**0,437%**	**0,446%**	**0,249%**	**0,239%**	**0,236%**

La comparaison par taille de population entre les moyennes obtenues par les algorithmes $AG^{OEP}(\chi,\boldsymbol{FIFO},-)$ et $AG^{OEP}(\chi,\boldsymbol{LIFO},-)$, pour l'ensemble d'instances *J30*, montre de meilleurs résultats pour l'algorithme avec la liste FIFO dans la plupart des cas. L'algorithme $AG^{OEP}(\chi,\boldsymbol{LIFO},-)$ l'emporte une seule fois sur six, lorsque la taille de la population est de *100* individus. Les écarts observés entre les diverses tailles de population sont assez faibles. Les meilleures déviations moyennes ont cependant été obtenues avec les

algorithmes $AG^{OEP}(\chi,\textit{FIFO},30x33)$ et $AG^{OEP}(\chi,\textit{FIFO},80x62)$. L'algorithme $AG^{OEP}(\chi,\textit{FIFO},-)$ est donc retenu pour la résolution de l'ensemble d'instances *J30* dans la suite même si un test statistique serait nécessaire pour valider la comparaison, étant donné les faibles écarts observés.

Tableau 5.7 : Résultats obtenus en variant la taille de population sur l'ensemble *J60* avec l'algorithme $AG^{OEP}(\chi,\textit{Mixte},-)$

J60	$AG^{OEP}(\chi,\textit{Mixte},-)$					
Taille de la population	**30**	**40**	**50**	**50**	**80**	**100**
Nombre de générations	**33**	**25**	**20**	**100**	**62**	**50**
Nombre de solutions	990	1000	1000	5000	4960	5000
1	12,78%	12,76%	12,73%	12,33%	**12,25%**	12,26%
2	**12,72%**	**12,74%**	12,77%	12,34%	12,30%	12,32%
3	12,73%	12,79%	**12,72%**	12,25%	12,28%	12,32%
4	12,74%	12,75%	12,77%	12,24%	12,33%	12,35%
5	12,76%	12,78%	12,86%	**12,22%**	12,29%	**12,19%**
Moyenne	**12,75%**	**12,76%**	**12,77%**	**12,28%**	**12,29%**	**12,29%**

Tableau 5.8 : Résultats obtenues en variant la taille de population sur l'ensemble *J120* avec l'algorithme $AG^{OEP}(\chi,\textit{Mixte},-)$

J120	$AG^{OEP}(\chi,\textit{Mixte},-)$					
Taille de la population	**30**	**40**	**50**	**50**	**80**	**100**
Nombre de générations	**33**	**25**	**20**	**100**	**62**	**50**
Nombre de solutions	990	1000	1000	5000	4960	5000
1	**38,50%**	**38,47%**	38,60%	37,46%	37,46%	37,40%
2	38,54%	38,59%	38,54%	37,34%	37,48%	37,36%
3	38,65%	38,55%	**38,51%**	37,36%	37,49%	37,46%
4	38,56%	38,51%	38,53%	37,36%	**37,40%**	**37,34%**
5	38,57%	38,61%	38,55%	**37,33%**	37,45%	37,45%
Moyenne	**38,56%**	**38,55%**	**38,55%**	**37,37%**	**37,46%**	**37,40%**

En ce qui concerne l'ensemble *J60*, il n'y pas de variations notables ni pour les moyennes, ni pour les déviations minimales obtenues, par rapport à la taille de la

population. Quant à l'ensemble d'instances *J120*, il n'y a pas de différences notables à *1000* évaluations de solutions. La meilleure déviation minimale est obtenue avec une population de *40* individus. Une taille de population de *50* individus semble mieux performer pour *5000* évaluations de solutions. Les algorithmes retenus à cette étape pour résoudre les ensembles d'instances *J60* et *J120* sont donc $AG^{OEP}(\chi,Mixte,40x25)$ et $AG^{OEP}(\chi,Mixte,50x100)$ même si statistiquement, ce paramètre ne devrait pas être considéré pertinent étant donné les faibles écarts.

Le Tableau 5.9 présente une comparaison des résultats des algorithmes retenus avec ceux de l'algorithme initial avec un croisement à deux points AG^{2P}, et ceux de l'algorithme de départ $AG^{OEP}(Ang,FIFO,-)$. Dans la première ligne du tableau se trouve le nombre de solutions générées et dans la deuxième ligne les noms des ensembles d'instances résolus. La troisième ligne du tableau rappelle les résultats moyens de l'algorithme AG^{2P}, la quatrième ligne ceux de l'algorithme $AG^{OEP}(Ang,FIFO,-)$ et la cinquième ligne rappelle les résultats des algorithmes retenus pour chaque ensemble d'instances. Dans la dernière ligne du tableau, l'écart relatif des résultats des algorithmes retenus par rapport à ceux de l'algorithme AG^{2P} est calculé.

Tableau 5.9 : Comparaison des résultats moyens obtenus par les algorithmes retenus avec ceux des algorithmes AG^{2P} et $AG^{OEP}(Ang,FIFO,-)$

Nombre de solutions	1000			5000		
Instances	**J30**	**J60**	**J120**	**J30**	**J60**	**J120**
(A) AG^{2P}	0,454%	12,75%	38,11%	0,230%	12,02%	36,02%
(B) $AG^{OEP}(Ang,FIFO,-)$	0,422%	13,05%	39,86%	0,256%	12,38%	37,92%
(C) Algorithmes retenus	0,410%	12,76%	38,55%	0,235%	12,28%	37,37%
Écart relatif entre (A) et (C)	**-9,69%**	**0,08%**	**1,15%**	**2,17%**	**2,16%**	**3,75%**

On observe une amélioration des résultats des algorithmes retenus par rapport à l'algorithme AG^{OEP}(*Ang,FIFO,-*) sur tous les ensembles d'instances traités et ce, quelle que soit le nombre de solutions évaluées. Par rapport au croisement à deux points (AG^{2P}) à *1000* évaluations de solutions, le croisement de type OEP montre une meilleure performance sur l'ensemble d'instances *J30* avec environ 10% d'écart et une performance à peu près égale sur l'ensemble d'instances *J60*. Sur l'ensemble *J120*, le croisement à deux points est meilleur que le croisement de type OEP d'environ *1%* seulement. À *5000* évaluations de solutions, le croisement à deux points est meilleur que le croisement de type OEP sur les ensembles *J30* et *J60* d'environ *2%* et sur l'ensemble *J120* d'environ *4%*. Le croisement de type OEP est donc resté compétitif par rapport au croisement à deux points. Cette compétitivité est meilleure sur *1000* évaluations de solutions que sur *5000* évaluations de solutions. Ce constat pourrait s'expliquer par le fait que l'OEP est plus axée sur la diversification que sur l'intensification. L'algorithme couvre assez vite une bonne partie de l'espace de solution, ce qui explique ses performances à *1000* évaluations. À *5000* évaluations de solutions, le croisement à deux points a le temps d'améliorer l'intensification et aussi de profiter de la mutation pour couvrir un plus large espace de recherche.

Afin d'exploiter les avantages des deux méthodes de croisement, à savoir le croisement de type OEP et le croisement à deux points, une hybridation plus modérée de l'AG est effectuée à la prochaine section. Les deux méthodes de croisement sont utilisées dans des proportions déterminées par des expérimentations numériques.

5.3 Combinaison du croisement à deux points avec le croisement de type OEP

Afin de combiner les avantages des deux méthodes de croisement discutées dans la section précédente, ces deux méthodes sont combinées dans l'algorithme avec des pourcentages d'utilisation définis pour chaque croisement. Pour déterminer le meilleur taux de croisement de type OEP à utiliser pour chaque ensemble d'instances, divers tests ont été effectués avec des pourcentages différents. Une fois ces pourcentages définis, d'autres expérimentations numériques ont été effectuées afin de déterminer les tailles de population qui conviennent à chaque algorithme.

L'AG combinant les deux méthodes de croisement a la même structure que les algorithmes AG^{2P} et AG^{OEP}. La différence entre ces algorithmes se situe au niveau du croisement. Lorsque deux individus sont sélectionnés pour la reproduction, la méthode de croisement à appliquer est déterminée de manière aléatoire, proportionnellement aux pourcentages attribués à chaque méthode. La méthode de croisement sélectionnée est appliquée à ces deux individus pour générer les deux enfants issus du croisement. Lorsque le croisement de type OEP est sélectionné, le troisième parent du croisement est la meilleure solution globale comme dans la section précédente.

5.3.1 Essais numériques et résultats obtenus

Les conditions expérimentales sont les mêmes que celles de la Section 5.2.2. Les algorithmes présentés dans cette section sont nommés suivant le modèle $AG^{OEP/2P}(\chi,b,c,d)$. Comme à la Section 5.2.2, χ représente les paramètres de calibration de l'OEP liés à l'utilisation du facteur de constriction ($w=0.7$, $c_1=1.5$, $c_2=1.5$), ces paramètres ayant été

retenus pour le croisement de type OEP. L'élément '*b*' désigne le type de liste utilisé par la procédure de correction de la pseudo-séquence, les listes utilisées étant de type FIFO ou mixte comme dans les algorithmes retenus à la Section 5.2.2. L'élément '*c*' se présente sous la forme *N*x*G* où *N* désigne la taille de la population et *G* le nombre de générations. L'élément '*d*' représente la proportion de croisement de type OEP utilisée par l'algorithme. Le symbole '-' est utilisé pour désigner un paramètre qui varie pour le même algorithme. Lorsque deux configurations de l'algorithme sont comparées, le paramètre introduisant la différence entre ces deux configurations est spécifié en gras dans le nom de l'algorithme. L'algorithme nommé $AG^{OEP/2P}(\chi,FIFO,-,-)$ est donc utilisé pour l'ensemble d'instances *J30* et l'algorithme $AG^{OEP/2P}(\chi,Mixte,-,-)$ est utilisé pour résoudre les ensembles *J60* et *J120*.

Des expérimentations numériques ont été effectuées dans un premier temps en utilisant le croisement de type OEP avec trois proportions différentes à savoir *20%*, *50%* et *80%*. Les résultats obtenus pour chaque ensemble d'instances sont présentés dans les Tableaux 5.10, 5.11 et 5.12 respectivement pour les ensembles *J30*, *J60* et *J120*. Ces tableaux présentent dans leur première ligne le nom de l'ensemble traité. Dans la deuxième ligne se trouve l'algorithme utilisé avec ses paramètres et dans la troisième ligne, les pourcentages d'utilisation du croisement de type OEP. De la quatrième à la huitième ligne sont présentées les déviations moyennes obtenues pour chaque exécution de l'algorithme considéré. La neuvième ligne des tableaux présente la moyenne des cinq exécutions.

Tableau 5.10 : Résultats obtenus en variant le pourcentage d'utilisation du croisement OEP sur l'ensemble *J30*

J30						
	$AG^{OEP/2P}(\chi, FIFO, 50x20, -)$			$AG^{OEP/2P}(\chi, FIFO, 100x50, -)$		
Pourcentage OEP	**20%**	**50%**	**80%**	**20%**	**50%**	**80%**
1	**0,368%**	0,387%	0,408%	0,183%	0,204%	0,228%
2	0,379%	0,425%	0,416%	0,159%	0,206%	0,204%
3	0,389%	**0,383%**	0,412%	0,187%	**0,179%**	0,221%
4	0,372%	0,386%	0,423%	0,168%	0,200%	**0,199%**
5	0,388%	0,393%	**0,409%**	**0,155%**	0,192%	0,245%
Moyenne	**0,379%**	**0,395%**	**0,414%**	**0,170%**	**0,196%**	**0,219%**

Tableau 5.11 : Résultats obtenus en variant le pourcentage d'utilisation du croisement OEP sur l'ensemble *J60*

J60						
	$AG^{OEP/2P}(\chi, Mixte, 50x20, -)$			$AG^{OEP/2P}(\chi, Mixte, 100x50, -)$		
Pourcentage OEP	**20%**	**50%**	**80%**	**20%**	**50%**	**80%**
1	12,78%	13,02%	13,10%	11,98%	12,21%	12,37%
2	12,82%	12,91%	13,06%	11,97%	**12,16%**	**12,30%**
3	12,77%	**12,86%**	13,13%	11,97%	12,25%	12,36%
4	12,80%	12,91%	13,07%	**11,95%**	12,23%	12,32%
5	**12,71%**	13,01%	**13,05%**	11,96%	12,24%	12,38%
Moyenne	**12,78%**	**12,94%**	**13,08%**	**11,97%**	**12,22%**	**12,35%**

Tableau 5.12 : Résultats obtenus en variant le pourcentage d'utilisation du croisement OEP sur l'ensemble *J120*

J120						
	$AG^{OEP/2P}(\chi, Mixte, 50x20, -)$			$AG^{OEP/2P}(\chi, Mixte, 100x50, -)$		
Pourcentage OEP	**20%**	**50%**	**80%**	**20%**	**50%**	**80%**
1	38,62%	**39,29%**	39,85%	**36,41%**	37,37%	**37,68%**
2	**38,59%**	39,35%	39,83%	36,44%	**37,23%**	37,68%
3	38,64%	39,34%	**39,59%**	36,41%	37,32%	37,68%
4	38,61%	39,43%	39,64%	36,45%	37,33%	37,69%
5	38,65%	39,44%	39,77%	36,48%	37,26%	37,68%
Moyenne	**38,62%**	**39,37%**	**39,74%**	**36,44%**	**37,30%**	**37,68%**

Les meilleurs résultats sont obtenus en utilisant le croisement de type OEP avec la plus faible proportion (*20%*) pour tous les ensembles d'instances. Pour affiner ces résultats, de nouvelles expérimentations ont été effectuées avec des proportions de *5%*, *10%*, *15%*, *25%* et *30%* de croisement de type OEP.

Le Tableau 5.13 présente les résultats obtenus en affinant le pourcentage d'utilisation du croisement de type OEP sur l'ensemble d'instances *J30*. Ce tableau présente dans sa première ligne l'ensemble d'instances résolu. La suite du tableau est constituée de deux parties. La première partie présente les résultats obtenus pour *1000* évaluations de solutions et la seconde partie présente les résultats obtenus en effectuant *5000* évaluations de solutions. La première ligne de chaque partie présente l'algorithme utilisé et la deuxième ligne, les pourcentages d'utilisation du croisement de type OEP. De la troisième à la septième ligne sont présentées les déviations moyennes obtenues pour chaque exécution de l'algorithme considéré. La huitième ligne de chaque partie présente la moyenne des cinq exécutions.

Tableau 5.13 : Résultats obtenus en affinant le pourcentage d'utilisation du croisement OEP sur l'ensemble *J30*

J30					
	$AG^{OEP/2P}(\chi, FIFO, 50x20, -)$				
Pourcentage OEP	**5%**	**10%**	**15%**	**25%**	**30%**
1	0,460%	0,368%	0,380%	0,386%	**0,338%**
2	0,463%	0,400%	0,392%	0,350%	0,384%
3	0,424%	0,431%	0,408%	0,402%	0,360%
4	0,468%	**0,365%**	**0,379%**	0,345%	0,394%
5	**0,419%**	0,407%	0,379%	**0,335%**	0,396%
Moyenne	**0,447%**	**0,394%**	**0,388%**	**0,364%**	**0,374%**
	$AG^{OEP/2P}(\chi, FIFO, 100x50, -)$				
Pourcentage OEP	**5%**	**10%**	**15%**	**25%**	**30%**
1	0,215%	0,166%	0,166%	0,180%	**0,141%**
2	0,190%	0,164%	0,182%	0,163%	0,179%
3	**0,162%**	0,174%	0,171%	**0,158%**	0,186%
4	0,195%	0,161%	0,185%	0,174%	0,155%
5	0,182%	**0,160%**	**0,165%**	0,169%	0,182%
Moyenne	**0,189%**	**0,165%**	**0,174%**	**0,169%**	**0,169%**

La meilleure moyenne est obtenue à *25%* du croisement de type OEP pour *1000* évaluations de solutions sur l'ensemble d'instances *J30*. À *5000* évaluations, la meilleure moyenne est obtenue avec *10%* de croisement de type OEP. Une représentation graphique des résultats obtenus avec ces deux pourcentages d'utilisation est donnée par le graphe de la Figure 5.5.

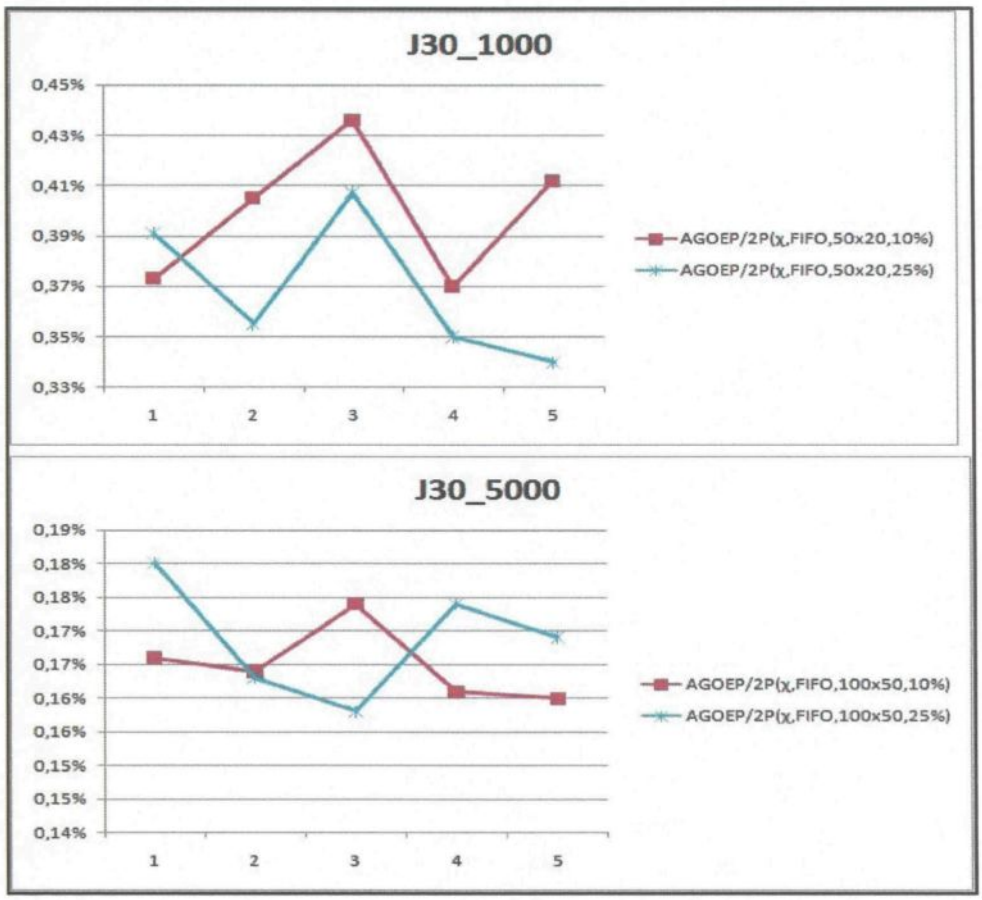


Figure 5.5 : Graphes de comparaison des algorithmes $AG^{OEP/2P}$ à *10%* et à *25%* de croisement de type OEP sur l'ensemble d'instances *J30*

Ces deux graphes montrent des écarts plus ou moins importants entre les deux courbes à *1000* évaluations de solutions. L'algorithme à *25%* est meilleur que l'algorithme à *10%* quatre fois sur cinq. À *5000* évaluations de solutions, les écarts sont moins grands entre les deux courbes et la courbe de l'algorithme à *25%* réussit à battre celle de l'algorithme à *10%* deux fois sur cinq. De plus, les meilleures déviations minimales sont obtenues avec l'algorithme $AG^{OEP/2P}(\chi,FIFO,-,\mathbf{25\%})$ dans les deux cas. C'est donc

l'algorithme à $AG^{OEP/2P}(\chi,FIFO,-,25\%)$ qui est retenu pour la résolution de l'ensemble d'instances *J30*.

Le Tableau 5.14 présente les résultats obtenus en affinant le pourcentage d'utilisation du croisement de type OEP sur l'ensemble d'instances *J60*. Ce tableau a la même structure que le tableau précédent.

Tableau 5.14 : Résultats obtenus en affinant le pourcentage d'utilisation du croisement OEP sur l'ensemble *J60*

J60					
	$AG^{OEP/2P}(\chi,Mixte,50x20,-)$				
Pourcentage OEP	**5%**	**10%**	**15%**	**25%**	**30%**
1	12,82%	**12,74%**	**12,68%**	12,83%	12,84%
2	12,81%	12,82%	12,79%	12,86%	12,86%
3	12,77%	12,76%	12,72%	12,85%	**12,80%**
4	**12,71%**	12,82%	12,70%	12,84%	12,80%
5	12,75%	12,75%	12,76%	**12,78%**	12,84%
Moyenne	**12,77%**	**12,78%**	**12,73%**	**12,83%**	**12,83%**
	$AG^{OEP/2P}(\chi,Mixte,100x50,-)$				
Pourcentage OEP	**5%**	**10%**	**15%**	**25%**	**30%**
1	12,00%	11,96%	11,97%	11,98%	12,06%
2	**11,94%**	11,93%	**11,91%**	12,04%	12,05%
3	12,00%	11,97%	12,01%	**11,94%**	12,03%
4	12,01%	11,94%	11,96%	11,99%	12,08%
5	12,00%	**11,90%**	11,94%	12,01%	**11,99%**
Moyenne	**11,99%**	**11,94%**	**11,96%**	**11,99%**	**12,04%**

Quoique les résultats soient très similaires, la meilleure moyenne s'obtient à *15%* de croisement de type OEP pour *1000* évaluations de solutions sur l'ensemble d'instances *J60* et à *10%* de croisement de type OEP à *5000* évaluations de solutions. La Figure 5.6 donne une représentation visuelle de ces résultats.

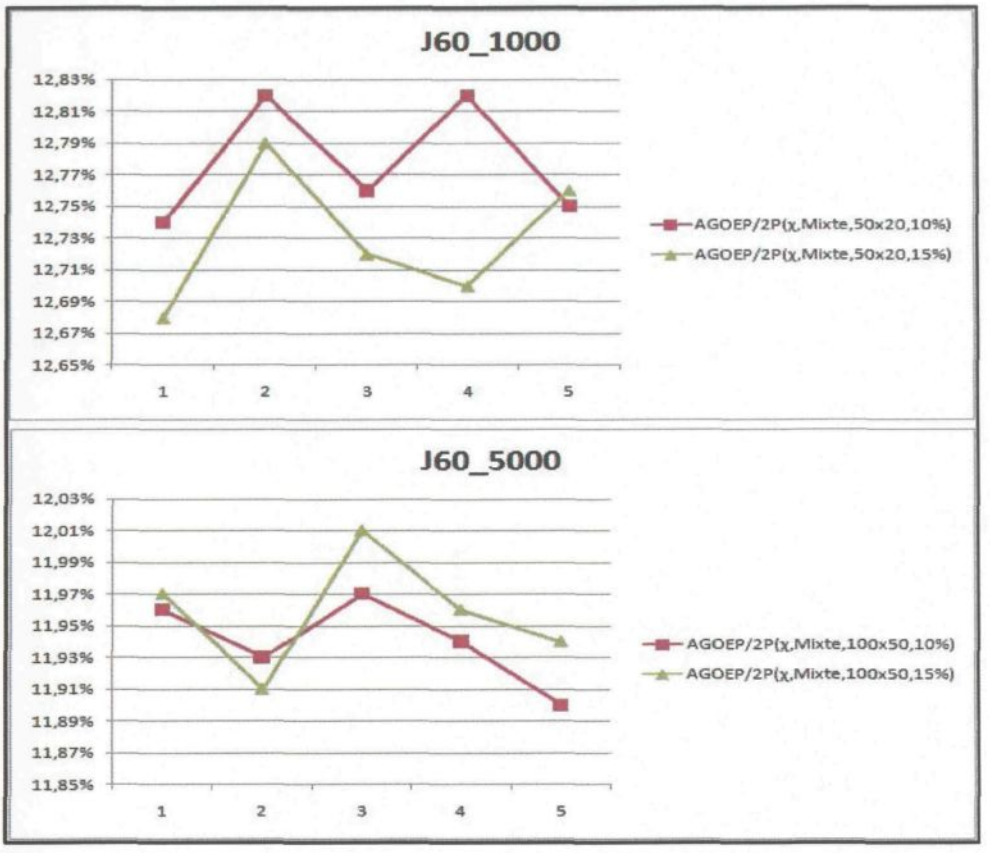


Figure 5.6 : Graphes de comparaison des algorithmes $AG^{OEP/2P}$ à *10%* et à *15%* de croisement de type OEP sur l'ensemble d'instances *J60*

Des écarts plus ou moins importants et une forte tendance sont observés pour la courbe de l'algorithme à *15%* de croisement de type OEP à *1000* évaluations de solutions. À *5000* évaluations, les écarts entre les deux courbes sont moins importants. L'algorithme $AG^{OEP/2P}(\chi,Mixte,-,15\%)$ est donc retenu pour la résolution de l'ensemble *J60*.

Le Tableau 5.15 présente les résultats obtenus en affinant le pourcentage d'utilisation du croisement de type OEP sur l'ensemble d'instances *J120*. Ce tableau a la même structure que les deux tableaux précédents.

Tableau 5.15 : Résultats obtenus en affinant le pourcentage d'utilisation du croisement OEP sur l'ensemble *J120*

J120					
	$AG^{OEP/2P}(\chi,Mixte,50x20,-)$				
Pourcentage OEP	**5%**	**10%**	**15%**	**25%**	**30%**
1	**38,19%**	38,44%	38,48%	38,85%	38,91%
2	38,28%	38,48%	38,55%	38,73%	**38,85%**
3	38,20%	**38,37%**	38,51%	38,79%	38,91%
4	38,31%	38,40%	38,46%	**38,70%**	38,88%
5	38,35%	38,43%	**38,34%**	38,81%	38,86%
Moyenne	**38,27%**	**38,42%**	**38,47%**	**38,78%**	**38,88%**
	$AG^{OEP/2P}(\chi,Mixte,100x50,-)$				
Pourcentage OEP	**5%**	**10%**	**15%**	**25%**	**30%**
1	36,13%	36,22%	36,33%	36,63%	36,80%
2	**36,11%**	36,21%	36,33%	36,56%	**36,64%**
3	36,17%	36,23%	36,34%	36,64%	36,75%
4	36,14%	**36,16%**	36,37%	36,60%	36,70%
5	36,15%	36,19%	**36,25%**	**36,54%**	36,77%
Moyenne	**36,14%**	**36,20%**	**36,32%**	**36,59%**	**36,73%**

Encore une fois, on observe très peu d'écart entre les résultats obtenus. Les meilleurs résultats sont obtenus pour l'ensemble d'instances *J120* à *5%* de croisement de type OEP, quelque soit le nombre d'évaluations de solutions effectuées. L'algorithme $AG^{OEP/2P}(\chi,Mixte,-,5\%)$ est donc retenu pour résoudre cet ensemble d'instances.

La calibration finale des algorithmes retenus a été effectuée en expérimentant diverses tailles de population. Les mêmes tailles de la population que pour les algorithmes AG^{OEP} ont été utilisées. Les résultats sont consignés dans les Tableaux 5.16, 5.17 et 5.18 respectivement pour les ensembles d'instances *J30*, *J60* et *J120*. Dans la première ligne de ces tableaux, le nom de l'algorithme utilisé est spécifié. La première cellule de cette ligne rappelle l'ensemble d'instances résolu. La deuxième ligne définit la taille de la population

et la troisième ligne, le nombre de générations effectuées par l'algorithme. Dans la quatrième ligne des tableaux, le nombre de solutions générées par l'algorithme est calculé en faisant le produit de la taille de la population par le nombre de générations. De la cinquième à la neuvième ligne se trouvent les déviations moyennes obtenues par les cinq exécutions de l'algorithme. À la dixième ligne, la moyenne obtenue par ces cinq exécutions est calculée.

Tableau 5.16 : Résultats obtenus en variant la taille de la population sur l'ensemble *J30* avec l'algorithme $AG^{OEP/2P}(\chi,FIFO,-,25\%)$

J30	$AG^{OEP/2P}(\chi,FIFO,-,25\%)$					
Taille de la population	30	40	50	50	80	100
Nombre de générations	33	25	20	100	62	50
Nombre de solutions	990	1000	1000	5000	4960	5000
1	0,365%	0,382%	0,386%	0,204%	0,176%	0,180%
2	0,383%	0,352%	0,350%	0,185%	0,163%	0,163%
3	0,361%	**0,332%**	0,402%	0,189%	0,176%	**0,158%**
4	**0,329%**	0,398%	0,345%	0,187%	**0,161%**	0,174%
5	0,349%	0,374%	**0,335%**	**0,163%**	0,177%	0,169%
Moyenne	**0,357%**	**0,368%**	**0,364%**	**0,186%**	**0,171%**	**0,169%**

Tableau 5.17 : Résultats obtenus en variant la taille de la population sur l'ensemble *J60* avec l'algorithme $AG^{OEP/2P}(\chi,Mixte,-,15\%)$

J60	$AG^{OEP}(\chi,Mixte,-,15\%)$					
Taille de la population	30	40	50	50	80	100
Nombre de générations	33	25	20	100	62	50
Nombre de solutions	990	1000	1000	5000	4960	5000
1	12,83%	12,73%	**12,68%**	12,12%	11,96%	11,97%
2	**12,69%**	12,69%	12,79%	12,04%	11,98%	**11,91%**
3	12,72%	12,69%	12,72%	12,06%	11,98%	12,01%
4	12,81%	**12,68%**	12,70%	12,06%	12,01%	11,96%
5	12,80%	12,70%	12,76%	**11,98%**	**11,93%**	11,94%
Moyenne	**12,77%**	**12,70%**	**12,73%**	**12,05%**	**11,97%**	**11,96%**

Tableau 5.18 : Résultats obtenus en variant la taille de la population sur l'ensemble *J120* avec l'algorithme $AG^{OEP/2P}(\chi,Mixte,-,5\%)$

J120	$AG^{OEP}(\chi,Mixte,-,5\%)$					
Taille de la population	30	40	50	50	80	100
Nombre de générations	33	25	20	100	62	50
Nombre de solutions	990	1000	1000	5000	4960	5000
1	38,26%	38,18%	**38,19%**	**36,22%**	36,07%	36,13%
2	**38,21%**	38,09%	38,28%	36,32%	36,05%	**36,11%**
3	38,21%	**38,07%**	38,20%	36,24%	36,12%	36,17%
4	38,32%	38,22%	38,31%	36,27%	36,11%	36,14%
5	38,32%	38,31%	38,35%	36,39%	**36,02%**	36,15%
Moyenne	**38,26%**	**38,17%**	**38,27%**	**36,29%**	**36,07%**	**36,14%**

Lorsque *1000* solutions sont évaluées, les meilleurs résultats, tant au niveau des moyennes qu'au niveau des déviations minimales, sont obtenus avec une taille de *30* individus pour l'ensemble d'instances *J30* et une taille de *40* individus pour les ensembles *J60* et *J120*. Ces résultats permettent donc de maintenir les mêmes tailles de population que pour l'algorithme AG^{OEP} à *1000* évaluations de solutions, i.e. une taille de *30* individus pour l'ensemble *J30* qui utilise une liste de type FIFO et une taille de *40* individus pour les ensembles *J60* et *J120* qui utilisent la liste de type mixte.

À *5000* évaluations de solutions, une qualité de solution similaire est obtenue avec les populations de *80* individus et de *100* individus autant pour l'ensemble *J30* que pour l'ensemble *J60*. L'algorithme utilisant une population de *80* individus montre néanmoins une meilleure stabilité par rapport à l'algorithme qui utilise une population de *100* individus. En ce qui concerne l'ensemble d'instances *J120*, les meilleurs résultats ont été obtenus avec la population de *80* individus. La taille de la population de *80* individus est

donc retenue pour tous les ensembles d'instances à *5000* évaluations de solutions. Compte tenu des faibles écarts observés, nous sommes toutefois conscients que ce paramètre affecte peu les résultats obtenus.

Les algorithmes retenus sont donc $AG^{OEP/2P}(\chi,FIFO,30x33,25\%)$ et $AG^{OEP/2P}(\chi,FIFO,80x62,25\%)$ pour l'ensemble d'instances *J30*, $AG^{OEP/2P}(\chi,Mixte,40x25,15\%)$ et $AG^{OEP/2P}(\chi,Mixte,80x62,15\%)$ pour l'ensemble d'instances *J60*, et enfin $AG^{OEP/2P}(\chi,Mixte,40x25,5\%)$ et $AG^{OEP/2P}(\chi,Mixte,80x62,5\%)$ pour l'ensemble d'instances *J120*.

5.3.2 Comparaisons et analyse des résultats

Le Tableau 5.19 présente une comparaison entre les résultats des AG retenus à chaque étape et ceux de l'algorithme initial AG^{2P}. Dans la première ligne du tableau se trouve le nombre de solutions générées et dans la deuxième ligne les noms des ensembles d'instances résolus. La troisième ligne du tableau rappelle les résultats moyens de l'algorithme AG^{2P} et la quatrième ligne ceux des algorithmes retenus pour le croisement de type OEP (AG^{OEP}). La cinquième ligne rappelle les résultats moyens des algorithmes retenus dans cette section pour la combinaison des deux méthodes de croisements ($AG^{OEP/2P}$). Dans la dernière ligne du tableau, l'écart relatif des résultats des algorithmes retenus par rapport à ceux de l'algorithme AG^{2P} est calculé.

Tableau 5.19 : Comparaison des résultats obtenus par les algorithmes AG^{2P}, $AG^{OEP}(\chi,FIFO,-)$ et $AG^{OEP/2P}(\chi,FIFO,-,25\%)$

Nombre de solutions	1000			5000		
Instances	**J30**	**J60**	**J120**	**J30**	**J60**	**J120**
(A) AG^{2P}	0,454%	12,75%	38,11%	0,230%	12,02%	36,02%
(B) AG^{OEP}	0,410%	12,76%	38,55%	0,235%	12,28%	37,37%
(C) $AG^{OEP/2P}$	0,357%	12,70%	38,17%	0,171%	11,97%	36,07%
Écart relatif entre (A) et (C)	**-21,37%**	**-0,42%**	**0,16%**	**-25,65%**	**-0,39%**	**0,14%**

Ces résultats montrent que l'algorithme $AG^{OEP/2P}$ obtient généralement une meilleure performance que les algorithmes à croisement unique AG^{2P} et AG^{OEP} sur les ensembles d'instances *J30* et *J60*. On observe en effet plus de *20%* d'amélioration sur les résultats de l'ensemble *J30* et une performance quasi-identique sur les ensembles *J60* et *J120*. L'hybridation a donc permis d'améliorer la qualité des résultats obtenus aussi bien au niveau du croisement à deux points qu'au niveau du croisement de type OEP.

Les résultats obtenus par l'algorithme $AG^{OEP/2P}$ montrent que plus la taille du problème augmente, plus grande est la proportion de croisement à deux points nécessaire à l'algorithme. Cette remarque confirme l'hypothèse selon laquelle le croisement de type OEP serait beaucoup plus axé sur la diversification et le croisement à deux points sur l'intensification. L'hybridation de ces deux méthodes de croisement apporte donc un équilibre entre ces deux composantes nécessaires à l'AG.

En partant de l'algorithme AG^{2P} et en modifiant uniquement la méthode de croisement, les résultats obtenus ont été améliorés principalement sur l'ensemble d'instances *J30*. Les résultats de l'algorithme initial AG^{AM} conçu par Alcaraz et Maroto [2006] n'avaient pas pu être reproduits par l'algorithme AG^{2P}. Il peut néanmoins être

supposé que si l'algorithme original des auteurs avait été disponible, leurs résultats auraient pu être améliorés dans le même sens. Notons d'ailleurs dans le Tableau 5.16 avec l'algorithme $AG^{OEP/2P}(\chi,FIFO,30x33,25\%)$ retenue pour résoudre l'ensemble d'instances *J30*, la meilleure déviation moyenne obtenue à savoir *0.329%* est plus faible que la déviation moyenne obtenue par les auteurs qui est de *0.334%*.

5.4 Conclusion

Ce chapitre offre une autre perspective par rapport à l'OEP. Dans la première partie du chapitre, cette méthode a été utilisée pour concevoir un croisement pour les AG. Cette technique s'est révélée efficace pour la résolution de l'ensemble d'instances *J30*. En ce qui concerne les ensembles d'instances *J60* et *J120*, une dégradation des solutions obtenues avec le croisement à deux points a été observée, ceci pouvant être dû à un manque d'intensification de la méthode employée. Il serait donc intéressant d'introduire une méthode de recherche locale pour améliorer l'intensification au sein de l'algorithme. Cependant le but de ce travail étant de vérifier le potentiel de l'OEP pour résoudre le RCPSP, cette technique n'a pas été utilisée. Il a aussi été constaté, comme dans l'algorithme OEP, que les ensembles d'instances *J60* et *J120* réagissent différemment que l'ensemble *J30* en fonction du type de liste utilisé pour la correction des pseudo-séquences.

Dans la seconde partie de ce chapitre, la méthode de croisement à deux points de l'algorithme AG^{2P} reproduit au Chapitre 4 a été combinée avec la nouvelle méthode de croisement de type OEP. Des expérimentations numériques ont permis de déterminer le niveau de contribution de chaque type de croisement dans l'AG. Il a été remarqué que les

problèmes de petite taille nécessitent une plus grande proportion de croisement de type OEP que les problèmes de plus grande taille. L'OEP est donc plus efficace pour résoudre les problèmes de petite taille du RCPSP que pour les problèmes de taille plus grande. Cet algorithme a démontré en général de meilleures performances que les autres algorithmes étudiés.

CHAPITRE 6

CONCLUSION

La résolution du RCPSP consiste en la détermination des dates de début ou de fin d'exécution des activités au sein d'un projet de manière à satisfaire les objectifs fixés tout en respectant les contraintes associées au projet. Le RCPSP possède plusieurs champs d'application aussi bien dans le domaine industriel que sur le plan organisationnel. Il possède plusieurs variantes qui se distinguent selon les objectifs à atteindre, les contraintes de préséances et les modes d'exécution des activités. La version classique, qui constitue l'objet de ce mémoire, a pour objectif la minimisation de la durée totale du projet et possède des contraintes de préséance *FS* (*Finish to Start*) avec un seul mode d'exécution pour chacune des activités du projet.

Plusieurs chercheurs se sont intéressés à ce problème d'optimisation NP-difficile dans la littérature. De nombreuses méthodes ont été proposées pour l'appréhender notamment les méthodes heuristiques et métaheuristiques. Parmi les métaheuristiques, les AG se révèlent être parmi celles qui performent le mieux pour ce problème. Il existe aussi des méthodes émergentes assez peu étudiées telle que l'OEP. Cette méthode, conçue originellement pour des problèmes d'optimisation à variables continues, est de plus en plus adaptée pour résoudre les problèmes à variables discrètes tels que le RCPSP. Toutefois, un seul travail utilisant cette méthode a été recensé dans la littérature sur le RCPSP classique. Deux algorithmes ont été conçus par les auteurs pour résoudre ce problème avec la méthode OEP. Des études récentes ont cependant été menées avec l'OEP discrète sur des problèmes similaires au RCPSP et ont donné de très bons résultats. L'hybridation constitue aussi un nouveau champ d'investigation dans le domaine des métaheuristiques. Cette technique combine les propriétés de diverses méthodes afin de générer un algorithme plus performant.

L'objectif principal de ce travail de recherche est *de contribuer à l'amélioration des outils de décision en ordonnancement de projet*. L'atteinte de cet objectif passe par une meilleure connaissance du problème d'ordonnancement de projet sous contrainte de ressources. Une présentation détaillée du problème ainsi qu'une revue de la littérature sur les méthodes utilisées pour le résoudre ont donc été réalisées dans le Chapitre 2. Dans le Chapitre 3, les deux métaheuristiques utilisées dans ce document pour résoudre le RCPSP sont décrites de façon plus détaillée. Il s'agit de l'AG qui compte parmi les méthodes les plus performantes de la littérature sur le RCPSP, et de l'OEP discrète, une méthode récente qui suscite actuellement beaucoup d'intérêt dans le domaine de l'optimisation. Ce mémoire a donc permis d'attirer l'attention sur une méthode émergente de la littérature pour résoudre les problèmes d'optimisation à variables discrètes.

Le premier objectif spécifique de ce mémoire était de *valider le potentiel de l'OEP pour la résolution du RCPSP*. Cet objectif a été atteint, au Chapitre 4 de ce document, par la conception d'un algorithme d'OEP pour la résolution du RCPSP. Cet algorithme s'inspire d'un algorithme d'OEP conçu pour un autre problème d'optimisation à variables discrètes de la littérature. L'algorithme proposé utilise une notion de distance entre la position des particules pour calculer leur vitesse. Un modèle discret a été utilisé aussi bien pour représenter une particule que pour définir sa vitesse. La particule constitue une séquence d'activités et la vitesse est un ensemble de déplacements qui sont appliquées aux activités de cette séquence. L'adaptation de cette technique au RCPSP passe aussi par l'introduction de la gestion des contraintes au niveau des préséances entre les activités et au niveau des ressources du projet. Le réglage des paramètres a également permis d'étudier

l'effet du facteur de constriction sur la performance de l'algorithme d'OEP. Les diverses expérimentations numériques effectuées ont permis de mettre en évidence le fait que l'OEP tire un meilleur avantage de la diversification au sein de la population que de l'intensification. Toutefois, lorsque la diversification est suffisante, notamment dans le cas des problèmes de petite taille, un peu plus d'intensification peut amener à de meilleurs résultats. Les performances de l'algorithme OEP(*UP1,Ang,LIFO,*-) ont été testées sur des ensembles d'instances de tests existant dans la littérature du RCPSP. Dans les expérimentations réalisées, il a été possible d'observer l'efficacité de l'algorithme en comparant les résultats obtenus avec ceux des deux seuls algorithmes de l'état de l'art qui utilisent aussi la méthode de l'OEP pour résoudre le RCPSP. Les résultats observés montrent de meilleures performances de l'algorithme OEP(*UP1,Ang,LIFO,*-) sur ses congénères et ce, quelque soit le nombre d'évaluations de solutions effectué.

La tendance générale au sein de la littérature est d'associer une procédure d'intensification à l'algorithme d'OEP afin d'en améliorer les performances. L'algorithme OEP(*UP1,Ang,LIFO,*-) n'utilise pas cette technique car le but de cette partie du travail de recherche était, d'une part, d'étudier le comportement de la méthode pour une utilisation ultérieure. D'autre part, nous voulions maintenir une base de comparaison la plus équitable possible par rapport aux algorithmes d'OEP existants car ces derniers n'en utilisent pas. Cela suscite toutefois un intérêt certain pour de futurs travaux de recherche.

Le second objectif de ce mémoire visait à *proposer un schéma d'hybridation entre l'OEP et les AG afin de combiner les forces des deux métaheuristiques.* Un AG de la littérature a été reproduit dans le Chapitre 4 afin de servir de base à l'hybridation. Cet

algorithme offre les meilleurs résultats de la littérature n'intégrant pas la recherche locale. Cette reproduction n'a pas été tout à fait conforme à l'algorithme original, ce qui pourrait être dû à une mauvaise interprétation de certains aspects de l'algorithme ou à des informations qui seraient manquantes dans l'article [Alcaraz et Maroto 2006]. L'algorithme reproduit AG^{2P} constitue alors la base de comparaison pour la suite des travaux effectués.

Dans le Chapitre 5, l'algorithme d'OEP a servi à la conception d'une méthode de croisement qui a remplacé la méthode de croisement à deux points de l'algorithme AG^{2P}. Le nouvel algorithme AG^{OEP} montre de meilleures performances que l'algorithme AG^{2P} sur les problèmes de petite taille avec un faible nombre d'évaluations de solutions. Les performances sont similaires ou moins bonnes par rapport à celles obtenues par l'algorithme AG^{2P} lorsque le nombre de solutions évaluées ou la taille des problèmes augmente. Cette situation s'explique sans doute par le fait que l'OEP est plus axée sur la diversification que sur l'intensification.

Afin d'améliorer l'intensification au sein de l'algorithme, les deux méthodes de croisement, à savoir le croisement à deux points et le croisement de type OEP, ont été combinées dans l'AG. Cette hybridation, présentée toujours dans le Chapitre 5, montre de meilleurs résultats que ceux obtenus précédemment. La combinaison se fait de façon complémentaire pour les deux méthodes et des pourcentages d'utilisation de chacune des méthodes de croisement sont définis pour chaque taille de problème. Les performances obtenues par l'algorithme à croisement hybride $AG^{OEP/2P}$ sont meilleures que celles des deux algorithmes à croisement unique AG^{OEP} et AG^{2P} sur les problèmes de petite taille. En ce qui concerne les problèmes de *60* et *120* activités, les performances obtenues par les trois

algorithmes semblent équivalentes. La méthode d'OEP, utilisée judicieusement avec une procédure qui favorise l'intensification, offre donc un meilleur compromis entre l'exploitation et l'exploration de l'espace de recherche. De plus, l'intégration d'un croisement hybridant l'OEP avec un croisement à deux points a permis d'améliorer de manière significative les performances de l'AG sur les problèmes de petite taille (*30* activités) et de taille moyenne (*60* activités). Toutefois, un travail reste à faire au niveau des problèmes de grande taille (*120* activités). Ces derniers suscitent donc notre intérêt pour des recherches futures. Également, le choix des meilleures configurations d'algorithme aurait avantage à être fait en utilisant une analyse de variance pour identifier clairement, à partir d'essais numériques plus importants, les paramètres réellement significatifs d'un point de vue statistique.

Ce travail de recherche a donc non seulement atteint ses objectifs, mais a également ouvert la voie à d'autres avenues aussi bien au niveau de l'amélioration de l'OEP pour la résolution des problèmes discrets que de son hybridation avec d'autres méthodes, notamment avec les AG. L'importance d'un bon compromis entre l'intensification et la diversification a aussi été mise en évidence.

Des travaux supplémentaires dans le sens de l'intensification permettraient d'améliorer l'efficacité de la méthode pour la résolution des problèmes de taille plus grande. Étant donné les nombreuses applications du RCPSP aux problèmes de la vie réelle notamment dans l'industrie et dans l'organisation, il serait aussi intéressant d'adapter cette méthode à la résolution de tels problèmes en prenant en compte la qualité des livrables et les reprises d'activités. Le RCPSPR (*RCPSP with Rework*) est en effet un problème

d'optimisation multi-objectifs dont le but est de minimiser la durée totale du projet tout en générant des ordonnancements robustes qui prennent en compte la possibilité de reprise des activités au cours du projet. Quelques d'auteurs ont essayé de résoudre ce problème dans la littérature, en agrégeant les objectifs pour en faire un problème uni-objectif. Par ailleurs, l'AG est reconnu comme étant une méthode efficace pour résoudre les problèmes d'optimisation multi-objectifs. L'AG développé dans ce mémoire pourrait donc être étendu lors de prochaines recherches pour la résolution du RCPSPR dans sa dimension multi-objectifs.

BIBLIOGRAPHIE

Alander, J.T. (1992). "On optimal population size of genetic algorithms." CompEuro'92.'Computer Systems and Software Engineering', Proceedings.: 65-70.

Alcaraz, J. et C. Maroto (2001). "A Robust Genetic Algorithm for Resource Allocation in Project Scheduling." Annals of Operations Research **102**(1-4): 83-109.

Alcaraz, J. et C. Maroto (2006). A Hybrid Genetic Algorithm Based on Intelligent Encoding for Project Scheduling. Perspectives in Modern Project Scheduling. S. US. **92:** 249-274.

Allahverdi, A., C.T. Ng, T.C.E. Cheng et M.Y. Kovalyov (2008). "A survey of scheduling problems with setup times or costs." European Journal of Operational Research **187**(3): 985-1032.

Anghinolfi, D. et M. Paolucci (2009). "A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup times." European Journal of Operational Research **193**(1): 73-85.

Artigues, C., S. Demassey et E. Neron (2008). Resource-constrained project scheduling : models, algorithms, extensions and applications. London; Hoboken, NJ, ISTE ; Wiley.

Baar, T., P. Brucker et S. Knust (1998). Tabu-search algorithms and lower bounds for the resource-constrained project scheduling problem. Meta-heuristics : Advances and Trends in Local Search Paradigms for Optimization. S. Voss, S. Martello, I. H. Osman et C. Roucairol, Kluwer Academic Publishers: 1-18.

Baker, J.E. (1985). Adaptive Selection Methods for Genetic Algorithms. Proceedings of the 1st International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Inc.

Baptiste, P., V. Giard, A. Haït et F. Soumis (2005). Gestion de production et ressources humaines : méthodes de planification dans les systèmes productifs. Montréal, Presses internationales Polytechnique.

Baptiste, P. et C. Le Pape (1996). "Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling." Proceedings of the Fifteenth Workshop of the UK Planning Special Interest Group **335**: 339-345.

Baptiste, P., C. Le Pape et W. Nuijten (2001). Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems, Springer.

Bean, J.C. et A.B. Hadj-Alouane (1993). A Dual Genetic Algorithm for Bounded Integer Programs, University of Michigan, College of Engineering.

Boctor, F.F. (1996). "Resource-constrained project scheduling by simulated annealing." International Journal of Production Research **34**(8): 2335.

Bouleimen, K. et H. Lecocq (2003). "A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version." European Journal of Operational Research **149**: 268-281.

Brindle, A. (1981). Genetic Algorithms for Function Optimization. Edmonton, Canada, Department of Computer Science, University of Alberta.

Brucker, P., S. Knust, A. Schoo et O. Thiele (1998). "A branch and bound algorithm for the resource-constrained project scheduling problem." European Journal of Operational Research **107**(2): 272-288.

Carlier, J. et B. Latapie (1991). "Une méthode arborescente pour résoudre les problèmes cumulatifs." RAIRO-Recherche Opérationnelle **25**(3): 311-340.

Carlier, J. et E. Néron (2003). "On linear lower bounds for the resource constrained project scheduling problem." European Journal of Operational Research **149**(2): 314-324.

Carlier, J. et E. Néron (2007). "Computing redundant resources for the resource constrained project scheduling problem." European Journal of Operational Research **176**(3): 1452-1463.

Chen, A.-l., G.-k. Yang et Z.-m. Wu (2006). "Hybrid discrete particle swarm optimization algorithm for capacitated vehicle routing problem." Journal of Zhejiang University - Science A **7**(4): 607-614.

Chen, P.-H. et H. Weng (2009). "A two-phase GA model for resource-constrained project scheduling." Automation in Construction **18**(4): 485-498.

Clerc, M. (1999). The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on.

Clerc, M. (2005). L'optimisation par essaims particulaires. Paris, Hermès Science Publications

Clerc, M. et J. Kennedy (2002). "The particle swarm - explosion, stability, and convergence in a multidimensional complex space." Evolutionary Computation, IEEE Transactions on **6**(1): 58-73.

Damay, J. et E. Sanlaville (2006). Méthodes par Séparation/Evaluation pour le RCPSP préemptif. 7ème congrès de la société Française de Recherche Opérationnelle et d'Aide à la Décision, Lille, France, Presses Universitaires de Valenciennes.

Darwin, C. (1859). "On the Origin of Species by Means of Natural Selection or the Preservation of Favoured Races in the Struggle for Life. Reprinted 1964." Cambridge (MA): Harvard University.

Davis, L. (1991). Handbook of genetic algorithms. New York, Van Nostrand Reinhold.

Debels, D., B.D. Reyck, R. Leus et M. Vanhoucke (2006). "A hybrid scatter search / Electromagnetism meta-heuristic for project scheduling." European Journal of Operational Research **169**: 638-653.

Demassey, S. (2003). Méthodes Hybrides de Programmation par Contraintes et Programmation Linéaire pour le Problème d'Ordonnancement de Projet à Contraintes de Ressources. Laboratoire Informatique d'Avignon, Université d'Avignon et des Pays de Vaucluse.

Demeulemeester, E.L. et W. Herroelen (2002). Project Scheduling: A Research Handbook, Kluwer Academic Publishers.

Demeulemeester, E.L. et W.S. Herroelen (1996). "An efficient optimal solution procedure for the preemptive resource-constrained project scheduling problem." European Journal of Operational Research **90**(2): 334-348.

Demeulemeester, E.L. et W.S. Herroelen (1997). "A Branch-And-Bound Procedure for the Generalized Resource-Constrained Project Scheduling Problem." Operations Research **45**(2): 201-212.

Dorigo, M. (1992). Optimization, Learning and Natural Algorithms. Dipartimento di Elettronica. Milano, Politecnico di Milano, Italy. **Ph.D.**

Dorndorf, U., E. Pesch et T. Phan-Huy (2000). "A branch-and-bound algorithm for the resource-constrained project scheduling problem." Mathematical Methods of Operations Research (ZOR) **52**(3): 413-439.

Dréo, J., A. Pétrowski, P. Siarry et É. Taillard (2003). Métaheuristiques pour l'optimisation difficile. Paris, Eyrolles.

Eberhart, R.C. et Y. Shi (2000). Comparing inertia weights and constriction factors in particle swarm optimization. Evolutionary Computation, 2000. Proceedings of the 2000 Congress on.

Gagnon, M. (2002). Modélisation et résolution heuristique de l'allocation des ressources en gestion de projets. Département Opérations et Systèmes de décision. Québec, Université Laval. **Ph.D.**

Gagnon, M., F.F. Boctor et G. d'Avignon (2005). A tabu search algorithm for the multiple mode resource-constrained project scheduling problem. Gestion des opérations et production - ASAC 2005, Toronto, Ontario.

Gen, M. et R. Cheng (1997). Genetic algorithms and engineering design. New York, N.Y., J. Wiley and Sons.

Glover, F. (1986). "Future paths for integer programming and links to artificial intelligence." Computers & Operations Research **13**(5): 533-549.

Goldberg, D.E. (1989). Genetic algorithms in search, optimization, and machine learning. Reading, Mass., Addison-Wesley.

Goldberg, D.E. et R. Lingle (1985). "Alleles, loci and the traveling salesman problem." Proceedings of the First International Conference on Genetic Algorithms and Their Applications: 154-159.

Hartmann, S. (1998). "A competitive genetic algorithm for resource-constrained project scheduling." Naval Research Logistics **45**: 733-750.

Hartmann, S. (2002). "A self-adapting genetic algorithm for project scheduling under resource constraints." Naval Research Logistics **49**: 433-448.

Hartmann, S. et R. Kolisch (2000). "Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem." European Journal of Operational Research **127**: 394-407.

Hartmann, S. et R. Kolisch (2006). "Experimental Investigation of Heuristics for Resource-Constrained Project Scheduling: An Update." European Journal of Operational Research **174**: 23-37.

Heppner, F. et U. Grenander (1990). A stochastic nonlinear model for coordinated bird flocks, The Ubiquity of Chaos, AAAS, Washington DC.

Herroelen, W., B. De Reyck et E. Demeulemeester (1998). "Resource-constrained project scheduling: A survey of recent developments." Computers & Operations Research **25**(4): 279-302.

Herroelen, W., B. De Reyck et E. Demeulemeester (2001). "A note on the paper "Resource-constrained project scheduling: Notation, classification, models and methods" by Brucker et al." European Journal of Operational Research **128**(3): 679-688.

Holland, J.H. (1962). "Outline for a Logical Theory of Adaptive Systems." Journal of the ACM **9**(3): 297-314.

Holland, J.H. (1975). Adaptation in Natural and Artificial System, Ann Arbor: The University of Michigan Press.

Hu, X., R.C. Eberhart et Y. Shi (2003). Swarm intelligence for permutation optimization: a case study of n-queens problem. Swarm Intelligence Symposium, 2003. SIS '03. Proceedings of the 2003 IEEE.

Icmeli-Tukel, O. et W.O. Rom (1997). "Ensuring quality in resource constrained project scheduling." European Journal of Operational Research **103**(3): 483-496.

Jeffcoat, D.E. et R.L. Bulfin (1993). "Simulated annealing for resource-constrained scheduling." European Journal of Operational Research **70**(1): 43-51.

Kennedy, J. (2006). Swarm Intelligence. Handbook of Nature-Inspired and Innovative Computing: 187-219.

Kennedy, J. et R. Eberhart (1995). Particle swarm optimization. Proceedings, IEEE International Conference on Neural Networks, Perth, WA, Australia.

Kennedy, J. et R.C. Eberhart (1997). A discrete binary version of the particle swarm algorithm. Systems, Man, and Cybernetics, 1997. 'Computational Cybernetics and Simulation'., 1997 IEEE International Conference on.

Kirkpatrick, S., C.D. Gelatt, Jr. et M.P. Vecchi (1983). "Optimization by Simulated Annealing." Science **220**(4598): 671-680.

Klein, R. (2000). "Project scheduling with time-varying resource constraints." International Journal of Production Research **38**(16): 3937-3952.

Kolisch, R. (1996a). "Efficient priority rules for the resource-constrained project scheduling problem." Journal of Operations Management **14**(3): 179-192.

Kolisch, R. (1996b). "Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation." European Journal of Operational Research **90**(2): 320-333.

Kolisch, R. et A. Sprecher (1997). "PSPLIB-A project scheduling problem library OR Software-ORSEP Operations Research Software Exchange Program." European Journal of Operational Research **96**(1): 205-216.

Kolisch, R., A. Sprecher et A. Drexl (1995). "Characterisation and generation of a general class of R.C.P.S.P." Management Science **41**(10): 1693–1703.

Lian, Z., X. Gu et B. Jiao (2008). "A novel particle swarm optimization algorithm for permutation flow-shop scheduling to minimize makespan." Chaos, Solitons & Fractals **35**(5): 851-861.

Lian, Z., B. Jiao et X. Gu (2006). "A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan." Applied Mathematics and Computation **183**(2): 1008-1017.

Liao, C.-J., T. Chao-Tang et P. Luarn (2007). "A discrete version of particle swarm optimization for flowshop scheduling problems." Computers & Operations Research **34**(10): 3099-3111.

Lova, A., P. Tormos, M. Cervantes et F. Barber (2009). "An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes." International Journal of Production Economics **117**(2): 302-316.

Mendes, J.J.M., J.F. Goncalves et M.G.C. Resende (2009). "A random key based genetic algorithm for the resource constrained project scheduling problem." Computers & Operations Research **36**(1): 92-109.

Merkle, D., M. Middendorf et H. Schmeck (2002). "Ant colony optimization for resource-constrained project scheduling." Evolutionary Computation, IEEE Transactions on **6**(4): 333-346.

Mingozzi, A., L. Maniezzo, R. S. et B. L. (1998). "An exact algorithm for the multiple ressource-constrained project scheduling problem based on a new mathematical formulation." Management Science **44**: 714-729.

Nonobe, K. et T. Ibaraki (2002). Formulation and tabu search algorithm for the resource constrained project scheduling problem. Essays and surveys in metaheuristics. C. C. Ribeiro et P. Hansen. Boston, Kluwer Academic Publishers: 557-588.

Oliver, I., D. Smith et J.R. Holland (1987). A study of permutation crossover operators on the traveling salesman problem. Genetic algorithms and their applications : proceedings of the second International Conference on Genetic Algorithms. J. Grefenstette. Hillsdale, N.J., L. Erlbaum Associates: 224-230.

Osman, I.H. et G. Laporte (1996). "Metaheuristics: A bibliography." Annals of Operations Research **63**(1-4): 513-623.

Pan, N.-H., P.-W. Hsaio et K.-Y. Chen (2008). "A study of project scheduling optimization using Tabu Search algorithm." Engineering Applications of Artificial Intelligence **21**(7): 1101-1112.

Parsopoulos, K.E. et M.N. Vrahatis (2007). "Parameter selection and adaptation in Unified Particle Swarm Optimization." Mathematical and Computer Modelling **46**(1-2): 198-213.

Pinson, E., P. C. et R. F. (1994). Using tabu search for solving the resource-constrained project scheduling problem. Proceedings of the Fourth International Workshop on Project Management and Scheduling, Katholieke Universiteit Leuven.

Potvin, J.-Y. (1996). "Genetic algorithms for the traveling salesman problem." Annals of Operations Research **63**(3): 337-370.

Pritsker, A., W. L. et W. P. (1969). "Multi-project scheduling with limited resources: a zero-one programming approach." Management Science **16**: 93-108.

Reeves, C.R. (1993). Genetic algorithms. Modern heuristic techniques for combinatorial problems, John Wiley \& Sons, Inc.: 151-196.

Reeves, C.R. (1995). "Genetic algorithms and combinatorial optimization". Applications of modern heuristic methods. V. J. Rayward-Smith. Henley-on-Thames, A. Waller Ltd.: 111-125.

Reynolds, C.W. (1987). Flocks, herds and schools: A distributed behavioral model. Proceedings of the 14th annual conference on Computer graphics and interactive techniques, ACM.

Sait, S.M. et H. Youssef (1999). Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems, IEEE Computer Society Press Los Alamitos, CA, USA.

Sampson, S.E. et E.N. Weiss (1993). "Local search techniques for the generalized resource constrained project scheduling problem." Naval Research Logistics **40**(5): 665-675.

Sha, D.Y. et C.-Y. Hsu (2006). "A hybrid particle swarm optimization for job shop scheduling problem." Computers & Industrial Engineering **51**(4): 791-808.

Shan, M., J. Wu et D. Peng (2007). "Particle Swarm and Ant Colony Algorithms Hybridized for Multi-Mode Resource-constrained Project Scheduling Problem with Minimum Time Lag." Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on: 5893-5897.

Shi, Y. et R.C. Eberhart (1998). A modified particle swarm optimizer. IEEE World Congress on Computational Intelligence, Anchorage, Alaska, Evolutionary Computation Proceedings.

Shi, Y. et R.C. Eberhart (1999). Empirical Study of Particle Swarm Optimization. Proceedings of the 1999 Congress of Evolutionary Computation, IEEE Press.

Shipeng, L., W. Cheng et W. Jinwen (2003). Ant Colony Optimization for Resource-Constrained Project Scheduling with Generalized Precedence Relations. Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence, IEEE Computer Society.

Silver, E.A., R. Victor, V. Vidal et D. de Werra (1980). "A tutorial on heuristic methods." European Journal of Operational Research **5**(3): 153-162.

Tasgetiren, M.F., Y.-C. Liang, M. Sevkli et G. Gencyilmaz (2007a). "A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem." European Journal of Operational Research **177**(3): 1930-1947.

Tasgetiren, M.F., P.N. Suganthan et Q.-Q. Pan (2007b). A discrete particle swarm optimization algorithm for the generalized traveling salesman problem. Proceedings of the 9th annual conference on Genetic and evolutionary computation. London, England, ACM.

Thomas, P.R. et S. Salhi (1998). "A Tabu Search Approach for the Resource Constrained Project Scheduling Problem." Journal of Heuristics **4**(2): 123-139.

Tukel, O.I. et W.O. Rom (1998). "Analysis of the characteristics of projects in diverse industries." Journal of Operations Management **16**(1): 43-61.

Valls, V., F. Ballestin et M.S. Quintanilla (2005). "Justification and RCPSP: A technique that pays." European Journal of Operational Research **165**: 375-386.

Valls, V., F. Ballestin et M.S. Quintanilla (2008). "A hybrid genetic algorithm for the resource-constrained project scheduling problem." European Journal of Operational Research **185**(2): 495-508.

Vanhoucke, M. et D. Debels (2008). "The impact of various activity assumptions on the lead time and resource utilization of resource-constrained projects." Computers & Industrial Engineering **54**(1): 140-154.

Vignier, A., J.C. Billaut et C. Proust (1999). "Les problemes d'ordonnancement de type flow-shop hybride: etat de l'art." Operations Research **33**(2): 117-183.

Whitley, L.D., S. Timothy et F. D'Ann (1989). Scheduling Problems and Traveling Salesmen: The Genetic Edge Recombination Operator. Proceedings of the 3rd International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc.

Wilson, D.G. et B.D. Rudin (1992). "Introduction to the IBM Optimization Subroutine Library." IBM Systems Journal **31**(1): 4-10.

Yang, X.-S. (2008). Introduction to computational mathematics. Singapore; Hackensack, N.J., World Scientific Pub.

Zhang, H., X. Li, H. Li et F. Huang (2005). "Particle swarm optimization-based schemes for resource-constrained project scheduling." Automation in Construction **14**(3): 393-404.