CHAPITRE 4

ALGORITHME ÉVOLUTIONNAIRE POUR LE PROBLÈME THÉORIQUE D'ORDONNANCEMENT DE VOITURES

4.1 Introduction

Le chapitre précédent a permis de présenter les algorithmes évolutionnaires et plus particulièrement les algorithmes génétiques comme des mécanismes de recherche généraux, puissants et robustes ayant été appliqués avec succès à la résolution de nombreux problèmes d'optimisation combinatoire. Toutefois, dans le Chapitre 2, nous avons remarqué que même si le POV intéresse la communauté scientifique depuis le milieu des années 80 et constitue maintenant un benchmark classique en optimisation combinatoire [Dincbas *et al.* 1988; Gent et Walsh 1999; Solnon 2000; Gottlieb *et al.* 2003; Gravel *et al.* 2005; Morin *et al.* 2006; Solnon 2006], très peu de travaux portent sur la conception d'algorithmes génétiques aussi bien pour la version théorique qu'industrielle du problème. À notre connaissance, Warwick et Tsang [1995] furent les premiers à appliquer les AG à la résolution du POV théorique. Dans leur approche, à chaque itération les individus sélectionnés sont croisés à l'aide d'un opérateur de croisement uniforme adaptatif (UAX). Après croisement, chaque enfant est réparé à l'aide d'une fonction de restauration des gènes puis amélioré à l'aide d'une procédure de recherche locale (hill-climbing). Plus récemment, Terada *et al.* [2006] proposèrent un algorithme génétique classique pour la résolution du POV théorique dans le but de le combiner par la suite avec une variante de recherche locale appelée : Squeaky-Wheel Optimization (SWO) [Joslin et Clements 1998]. Les expérimentations effectuées par les auteurs montrent que les approches proposées à base d'AG sont capables de résoudre des instances du problème de car sequencing avec un faible taux d'utilisation. Toutefois, l'efficacité est sérieusement diminuée pour les instances avec un taux d'utilisation plus élevé [Warwick et Tsang 1995; Terada *et al.* 2006]. On peut

supposer que cette situation s'explique en partie par la difficulté qu'il y a à définir des opérateurs génétiques efficaces et adaptés à la résolution du problème. En effet, les opérateurs de croisement génétiques sont généralement présentés dans la littérature pour résoudre des problèmes de voyageur de commerce (TSP), des problèmes utilisant une codification binaire [Michalewicz 1994; Potvin 1996] ou ceux utilisant une représentation réelle [Ben Hamida 2001] et ne sont donc pas adaptés aux spécificités du POV.

L'objectif principal de ce chapitre consiste donc à résoudre efficacement le POV théorique à l'aide d'un algorithme génétique en proposant deux nouveaux opérateurs de croisement spécifiques pour ce type de problème. Le reste du chapitre est organisé de la façon suivante. La Section 4.2 introduit les deux nouveaux opérateurs proposés. La Section 4.3, de son côté, présente les différents opérateurs de mutation utilisés. Le fonctionnement général de l'algorithme génétique est par la suite présenté à la Section 4.4. Finalement, la Section 4.5 présente les résultats expérimentaux des approches proposées et les compare avec ceux d'autres approches de résolution tirées de la littérature.

4.2 Nouveaux opérateurs de croisement pour le POV

Les opérateurs de croisement classiques, comme le croisement binaire ou encore le croisement réel, ne sont pas adaptés à la nature particulière des contraintes d'espacement du POV. Cette situation explique sans doute le fait que cette métaheuristique a très peu été appliquée à la résolution du POV. Dans cette partie, nous proposons de combler cette lacune en introduisant deux nouveaux opérateurs de croisement spécifiquement conçus

pour le POV. On note ici que chacun des opérateurs de croisement présenté dans cette section génèrent deux enfants.

4.2.1 Notion d'intérêt

Afin de présenter les différents opérateurs de croisement proprement dit, il importe de définir au préalable une notion importante qui est utilisée par ces opérateurs : la notion d'intérêt. L'intérêt de placer une voiture de la classe v à une position i donnée de la séquence en fonction des classes de voitures déjà placées est calculé selon l'équation suivante :

$$I_{vi} = \begin{cases} D_v & \text{si } NbNouveauxConflits_{vi} = 0 \\ -NbNouveauxConflits_{vi} & \text{sinon} \end{cases} \quad (4.1)$$

où $NbNouveauxConflits_{vi}$ correspond au nombre de nouveaux conflits engendrés par l'addition d'une voiture de la classe v à la position i et D_v correspond à la difficulté de la classe comme calculée à l'Équation 2.3 du chapitre 2.

4.2.2 Non Conflict Position Based Crossover (NCPX)

L'idée principale derrière tout opérateur de croisement consiste à générer différents individus potentiellement prometteurs. Pour y arriver, un bon opérateur de croisement pour le POV devrait tenir compte, autant que possible, des spécificités des contraintes d'espacement.

Le premier opérateur de croisement présenté cherche à produire un enfant à partir de gènes situés à des positions non conflictuelles dans les parents. Pour ce faire, la première opération consiste à tirer de manière aléatoire un nombre nb_g entre 0 et *nbpossconflit*.

nbpossconflit correspond au nombre de position sans conflit retrouvé dans le parent 1 (P_1). Ce nombre nb_g sert à déterminer le nombre de « bons » gènes qui conserveront la même position dans l'enfant et dans le Parent 1. Par la suite, une position de départ (Pos_d) est déterminée aléatoirement entre 1 et $|P_1|$ dans l'enfant à créer, où $|P_1|$ indique la longueur du chromosome parent. Les gènes situés à des positions non conflictuelles sont copiés, dans un premier temps, de P_1 vers l'enfant en partant de Pos_d jusqu'à la fin du chromosome. Dans un deuxième temps, si le nombre de gènes recopiés est inférieur à nb_g, le processus de copie recommence en partant du début de l'enfant jusqu'à Pos_{d-1}. Le reste des gènes de P_1 est utilisé pour constituer une liste de classe de voitures à placer L. Par la suite, on détermine aléatoirement une position (Pos) à partir de laquelle le reste du chromosome enfant va être complété. Pour finir, les classes de voitures contenues dans L sont placées en fonction de leur intérêt selon l'équation suivante :

$$v = \begin{cases} \arg\max\{I_{vi}\} & \text{si } p \leq 0.95 \\ V & \text{sinon} \end{cases} \qquad (4.2)$$

où p est un nombre aléatoire entre 0 et 1 et V déterminé de manière probabiliste parmi les classes de voitures introduisant le moins de conflits. On note cependant qu'en cas d'égalité sur $\arg\max\{I_{vi}\}$, si une des classes de voitures impliquée dans l'égalité se retrouve dans P_2 à la position i sans conflit alors on retient cette classe de voitures. Dans le cas contraire, on tire aléatoirement une classe de voitures parmi celles qui sont impliquées dans l'égalité. Le fonctionnement de l'opérateur de croisement *NCPX* est illustré à la Figure 4.1 pour deux individus P_1 = 21352446 et P_2 = 32621454. Si on suppose que l'évaluation de P_1 a retourné 5 positions sans conflits et qu'on a tiré $nb_g = 4$ et $Pos_d = 3$. À partir de Pos_d, on peut copier les gènes 5, 4, 4 et 2 dans l'enfant. Les gènes 2, 3, 1 et 6 de P_1 servent à constituer la liste

initiale L. Finalement, en supposant qu'on ait tiré Pos = 7, on commence à placer respectivement les gènes 3, 2, 6 et 1 de L dans l'enfant à partir de Pos. Ainsi, les gènes 1,2 et 6 sont directement hérités de P_2. L'enfant généré à partir de P_1 et P_2 avec cette technique est E_1= 22651443.

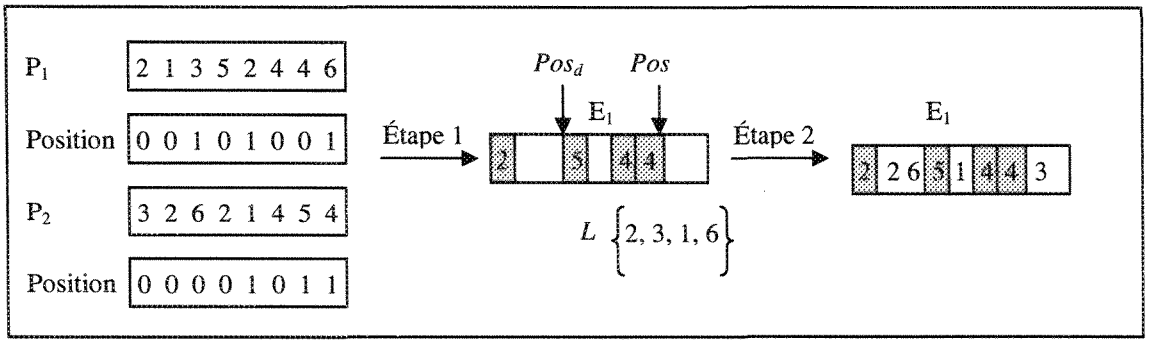

Figure 4.1 : Illustration du fonctionnement du croisement *NCPX*

L'objectif de cette technique de croisement consiste donc, au fil des générations, à favoriser le plus possible les « bons » gènes, i.e ceux situés à des positions non conflictuelles, provenant des parents de manière à minimiser le nombre de classes de voitures à repositionner.

4.2.3 Interest Based Crossover (IBX)

Le deuxième opérateur de croisement proposé s'inspire du PMX [Goldberg et Lingle 1985]. La Figure 4.2 illustre le fonctionnement de cet opérateur afin de tenir compte des contraintes de production et d'espacement du POV. Dans un premier temps, deux points de coupure sont déterminés aléatoirement sur chaque parent P_1 et P_2. Afin de créer un enfant, la sous-chaîne 351 comprise entre les deux points de coupure du premier parent ($a_1 \in P_1$) est directement recopiée dans l'enfant. Ensuite, deux listes (L_1 et L_2) non ordonnées de

classes de voitures sont respectivement créées à partir des sous-chaînes $b_3 = \{3, 2\}$ et $b_4 = \{4, 5, 6\}$ de P_2. Cependant, lors de cette opération, il se peut qu'une partie de l'information soit perdue par l'introduction de doublons supplémentaires. Dans notre exemple, on remarque à l'aide de la Figure 4.2 que les contraintes de production pour les classes de voitures 2, 3, 4 et 5 ne sont plus respectées. De manière à restaurer l'intégralité des gènes et qu'exactement c_v voitures de la classe v soient produites, un remplacement des gènes 3 et 5 (obtenu à partir de a_1-a_2) dont le nombre dépasse les contraintes de productions par les gènes 4 et 2 (obtenu à partir de a_2-a_1), dont le nombre est maintenant inférieur aux contraintes de production, est aléatoirement effectué dans les listes L_1 et L_2 à la deuxième étape. Finalement, la dernière étape consiste à reconstruire le début et la fin de l'enfant à partir des deux listes. Pour réaliser cette étape du croisement, les classes de voitures $\in L_1$ sont ordonnées en fonction de leur *Intérêt* en partant du premier point de coupure vers le début de la séquence. Pour cela, la classe v à placer est donnée en utilisant l'Équation 4.2.

Pour déterminer V, on utilise le même principe que la sélection par roulette présentée au chapitre précédent en limitant toutefois le tirage aux classes de voitures introduisant le moins de conflits. Pour cela, on associe à chaque classe candidate un segment de la roue proportionnelle à son intérêt I_{vi}. Par la suite, on concatène ces segments sur un axe gradué entre 0 et 1. Finalement, on tire aléatoirement un nombre *rnd* entre 0 et 1 et la classe V choisie est celle dont le segment correspond à *rnd*. Notons aussi que dans l'approche gourmande en cas d'égalité sur arg max$\{I_{vi}\}$, l'égalité est brisée à 35% de manière aléatoire et le reste du temps en choisissant la classe de voitures possédant l'option la plus difficile (i.e celle ayant le ratio r_o/s_o le plus petit et s'appliquant sur le plus de voitures).

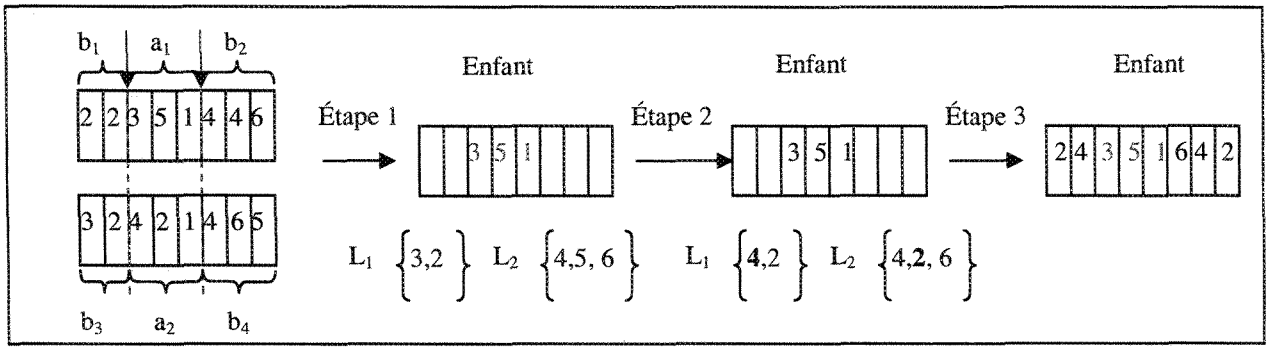

Figure 4.2 : Illustration du fonctionnement du croisement *IBX*.

Les voitures sont ordonnées de manière similaire à partir du deuxième point de coupure vers la fin de la séquence à l'aide de la liste L_2. Un deuxième enfant est créé de manière similaire en partant cette fois ci du parent 2.

Cette technique de croisement contrairement au PMX n'essaie pas de préserver la position absolue des gènes mais cherche plutôt à garder les gènes dans la même portion du chromosome que celles qu'ils occupaient dans l'un des deux parents. Dans les faits, le nombre de gènes qui ne seront pas situés dans la même zone qu'ils occupaient dans l'un des deux parents sera au plus égal à la longueur de la sous-chaîne comprise entre les deux points de coupure. Dans l'exemple précédent, seul les gènes 4 et 2 se retrouvent dans des portions du chromosome différentes de celles occupées à l'origine dans l'un des deux parents.

4.3 Opérateurs de mutation

Quatre opérateurs de mutation de base ont été utilisés : l'*échange*, l'*inversion*, le *mélange aléatoire* et le *déplacement*. Les trois premiers opérateurs ont souvent été utilisés

dans la littérature à l'intérieur d'algorithmes de recherche locale pour le POV. Un échange consiste simplement à échanger la position de deux voitures appartenant à des classes différentes dont au moins une participe à un conflit. Une inversion entre deux classes de voitures v_i et v_l consiste à inverser la sous-séquence comprise entre les positions i et l en s'assurant que l'une des deux classes de voitures participe à un conflit. Le mélange aléatoire entre deux classes de voitures v_i et v_l revient à mélanger aléatoirement les positions des voitures à l'intérieur de la sous-séquence bornée par les classes v_i et v_l. Finalement, l'opérateur de mutation par déplacement sélectionne une sous-séquence (un bloc) de voitures et l'insère à une autre position dans la séquence. Deux types de déplacement sont utilisés ici, le déplacement d'un bloc de voitures dont au moins une participe à un conflit ($depl_{cflt}$), ou le déplacement d'un bloc de voitures choisi aléatoirement ($depl_{aléa}$).

4.4 Algorithme génétique pour le POV

Dans cette section, nous présentons un nouvel AG pour résoudre le POV. Les différents éléments de son implémentation y sont présentés plus en détails.

4.4.1 Représentation des individus

Au lieu d'opter pour une représentation classique sous forme de chaîne de bits, qui apparaît peu adaptée pour ce type de problème, chaque individu de l'algorithme génétique proposé est représenté à l'aide d'un vecteur d'entiers de longueur *nc*, où chaque valeur correspond à une classe de voitures du problème à résoudre.

4.4.2 Création de la population initiale

Généralement, les individus de la population de départ sont initialisés de manière aléatoire et parfois à l'aide d'un algorithme glouton et stochastique [Basseur 2004]. Dans l'implémentation proposée, les individus de la population initiale sont générés de 2 façons : à 70 % de manière aléatoire en veillant à ce que les individus produits soient des solutions valides, et à 30 % en utilisant une heuristique gourmande basée sur l'Équation 4.1.

4.4.3 Sélection des individus

À chaque itération, lors de la phase de génération de la population d'enfants, la première étape effectuée par l'AG consiste à sélectionner des individus pour le croisement. La procédure de sélection utilisée est une sélection par tournoi, car cette technique de sélection n'est pas coûteuse à mettre en œuvre et à exécuter. Comme notre algorithme est élitiste, une faible taille de tournoi permet de limiter la pression de la sélection et par conséquence une convergence trop hâtive. C'est dans cette optique que, dans notre implémentation, la taille du tournoi est limitée à deux individus.

4.4.4 Application des opérateurs génétiques

Une fois sélectionnés, les individus sont croisés selon une certaine probabilité (p_c). Les opérateurs de croisement utilisés sont choisis parmi ceux présentés à la Section 4. On note que si le critère p_c n'est pas rencontré, alors un individu α est introduit dans la population. Cet individu, que l'on appelle *migrant,* est constitué à l'aide de l'heuristique gourmande utilisée lors de la création de la population initiale. Après avoir été générés par croisement, les enfants subissent une mutation selon une certaine probabilité (p_m). Les opérateurs de mutation utilisés sont sélectionnés par paire (*déplacement-inversion* ou *échange-mélange*)

parmi les quatre opérateurs présentés à la section précédente, chaque paire ayant une chance sur deux d'être tirée. Si la paire échange-mélange est sélectionnée, la probabilité de choisir un des deux opérateurs est respectivement de 0.99 et 0.01. Dans le cas où la paire déplacement-inversion est tirée, les probabilités de choisir l'un ou l'autre passe respectivement à 0.75 et 0.25. Finalement, on note que lorsqu'une mutation par déplacement est appliquée, on effectue à 90% du temps un $depl_{aléa}$. Mentionnons que les probabilités utilisées pour les différents opérateurs de croisement ont été déterminées de manière empirique.

4.4.5 Procédure de remplacement

Deux procédures de remplacement élitiste de la population sont utilisées dans l'algorithme génétique proposé. Un remplacement déterministe de type $(\mu+\lambda)$ et un de type *sélection inverse*. Le remplacement $\mu+\lambda$ consiste simplement à réunir les populations parent et enfant et à conserver les μ meilleurs individus. La stratégie de remplacement par sélection inverse consiste quant à elle, lorsque un enfant x donné est produit, à remplacer le pire individu de la population parent par l'individu x créé. Toutefois, une particularité de notre implémentation consiste à permettre le remplacement du pire individu de la population par un enfant ayant une moins bonne fitness dans 1% des cas afin d'éviter une convergence prématurée de l'algorithme. Comme chacun des opérateurs de croisement génèrent deux enfants, un remplacement de type $\mu+\lambda$ est appliqué en utilisant tous les premiers des deux enfants créés par croisement plus les éventuels migrants afin de former une population enfant de taille λ. Le deuxième des deux enfants sera introduit ou non dans

la population en utilisant le remplacement par sélection inverse. Il est important de préciser que dans notre algorithme $\mu = \lambda = N$.

4.5 Expérimentations numériques

Les différents algorithmes génétiques présentés dans cet article ont tous été implémentés en C^{++} avec Visual Studio .Net 2005. L'ordinateur utilisé pour les expérimentations numériques est un Dell équipé d'un processeur pentium Xeon 3.6 Ghz avec 1 Go de mémoire vive et tournant sous Windows XP.

Dans les expérimentations numériques réalisées pour évaluer la performance des différentes versions de l'algorithme génétique proposé, les paramètres N, p_c, p_m, $NbGen$ qui représentent respectivement la taille de la population, la probabilité de croisement, la probabilité de mutation et le nombre maximum de générations sont fixés à 250, 0.8, 0.09 et 700. Ces paramètres ont été déterminés de manière empirique afin d'obtenir une base de comparaison équitable avec d'autres algorithmes de la littérature.

4.5.1 Jeux d'essai

La performance des opérateurs de croisement est évaluée à l'aide de trois ensembles d'instances de POV disponibles sur Internet. Ces instances de problèmes présentent entre 100 et 400 voitures. Il s'agit des mêmes instances ayant servi à Gravel *et al.* [2005] et à Gagné *et al.* [2006] pour démontrer la performance de leurs algorithmes. Les trois ensembles sont tirés de CSPLib (http://www.csplib.org/). Le premier ensemble (ET1) contient 70 instances contenant chacune 200 voitures, 5 options et entre 17 et 30 classes de voitures. Ces 70 instances sont divisées en 7 groupes selon le taux d'utilisation. Pour

chacune de ces instances, il existe une solution sans conflit. Le deuxième ensemble (ET2) [Gent et Walsh 1999] est composé de 9 instances plus difficiles dont 5 pour lesquelles aucune solution sans conflits n'est connue. Chacune des instances de l'ET2 est composée de 100 voitures ayant chacune 5 options et entre 18 et 24 classes de voitures. Finalement, le dernier ensemble (ET3) proposé par Gravel *et al.*[2005] contient 30 instances difficiles de taille variant entre 200 et 400 voitures avec 5 options et entre 18 et 24 classes de voitures. Parmi les 30 instances de l'ET3, il y en a 23 pour lesquels aucune solution sans conflits n'est connue.

4.5.2 Comparaison expérimentale des différents opérateurs de croisement

L'objectif de la première série d'expérimentation est de comparer différentes versions de l'AG proposé avec d'autres AG de la littérature (GAcSP [Warwick et Tsang 1995] et le GA [Terada *et al.* 2006]) ainsi qu'avec un algorithme d'optimisation par colonie de fourmis qui s'est avéré particulièrement performant sur les différents jeux d'essai, l'ACS-3D [Gagné *et al.* 2008]. Les différentes versions de notre algorithme se divisent en trois : une version avec croisement NCPX (AG-NCPX), une version utilisant le croisement IBX (AG-IBX) et une version utilisant les deux opérateurs de croisement (AG-NCPX/IBX). On veut ainsi déterminer, d'une part, lequel des opérateurs proposés est le plus performant sur les différents ensembles tests et, d'autre part, démontrer la performance des AG proposés comparativement aux algorithmes de la littérature. On note que les résultats des différents algorithmes de la littérature ont été directement extraits des articles correspondants. De plus, les AG de la littérature n'ont été testés que sur les instances de l'ET1, la comparaison

entre ces approches et les méthodes proposées dans cette thèse article se limite donc à ces instances.

Les résultats de la comparaison du GAcSP, du GA, de l'ACS-3D, de l'AG-IBX, de l'AG-NCPX et de l'AG-NCPX/IBX sont présentés au Tableau 4.1. Chaque ligne du tableau donne le nom du groupe d'instances, suivi pour chaque algorithme du pourcentage d'exécutions ayant trouvé une solution sans conflit. Chaque groupe d'instances a été résolu à 100 reprises par le GAcSP, l'ACS-3D, l'AG-IBX, l'AG-NCPX et l'AG-NCPX/IBX et à 24 reprises par le GA. En ce qui concerne les meilleurs résultats obtenus, ils sont indiqués avec une trame en gris. On constate une nette différence entre les résultats de l'AG-IBX, l'AG-NCPX, l'AG-NCPX/IBX et de l'ACS-3D et ceux du GAcSP et du GA. En effet, l'AG-IBX, l'AG-NCPX et l'AG-NCPX/IBX et l'ACS-3D obtiennent un taux de succès de 100 % pour toutes les instances de l'ET1. À l'opposé, les performances du GAcSP et du GA sont sérieusement dégradées pour les groupes d'instances 80, 85 et 90 et ce, même lorsque les auteurs augmentent le nombre de générations en le faisant passer de 1000 à 5000 générations [Terada *et al.* 2006].

Par ailleurs, l'écart entre les performances du GAcSP et du GA avec celles de l'AG-IBX, de l'AG-NCPX et de l'AG-NCPX/IBX mettent en évidence l'importance d'intégrer des opérateurs propres au problème considéré pour améliorer la performance globale des algorithmes. En effet, le GA proposé par Terada *et al.* est un algorithme classique utilisant un croisement à un point de coupure qui gagnerait probablement en performance s'il intégrait des opérateurs génétiques ou des heuristiques dédiés au POV. Les auteurs mentionnent, dans ce sens, que l'ajout d'une heuristique de réparation après croisement

améliore les performances de l'algorithme [Terada *et al.* 2006] pour les groupes d'instances 80-*, 85-* et 90-*. Ces résultats doivent cependant être nuancés, par le fait que le GAcSP n'a pas été testé sur toutes les instances de l'ET1 par ses auteurs et que le monde des métaheuristiques ainsi que l'informatique ont beaucoup évolué depuis la publication des résultats de Warwick et Tsang en 1995.

Cet ensemble test ne peut toutefois pas être utilisé pour différencier les performances des trois approches proposées entre elles ni par rapport à l'ACS-3D, mais il permet de mettre en valeur l'efficacité de l'AG-IBX, de l'AG-NCPX et de l'AG-NCPX/IBX sur des instances dites « faciles ».

Instance	GAcSP	GA	ACS-3D	AG-IBX	AG-NCPX	AG-NCPX/IBX
	%	%	%	%	%	%
60-*	19	100	100	100	100	100
65-*	-	100	100	100	100	100
70-*	23	100	100	100	100	100
75-*	-	80	100	100	100	100
80-*	9	16	100	100	100	100
85-*	-	2	100	100	100	100
90-*	-	1	100	100	100	100

Tableau 4.1 :Résultats expérimentaux du GAcSP [Warwick et Tsang 1995], du GA [Terada *et al.* 2006], de l'ACS-3D [Gagné *et al.* 2008], de l'AG-IBX, de l'AG-NCPX et de l'AG-NCPX/IBX sur l'ET1

Le Tableau 4.2 présente les résultats de l'AG-IBX, de l'AG-NCPX, de l'AG-NCPX/IBX et de l'ACS-3D pour les neuf instances de l'ET2. Chaque instance a été résolue à 100 reprises par chaque algorithme et on retrouve respectivement dans le tableau, le nom de l'instance, la valeur de la meilleure solution connue et pour chaque algorithme le nombre moyen de conflits (*moyenne*) et le nombre de fois où chaque algorithme parvient à obtenir la meilleure solution connue (*# fois meilleure solution*). En ce qui concerne les

meilleurs résultats obtenus, ils sont indiqués avec une trame en gris. En comparant, dans un premier temps, les nombres moyens de conflits obtenus par les trois algorithmes génétiques, on note que l'AG-NCPX et l'AG-NCPX/IBX obtiennent de meilleurs résultats que l'AG-IBX sur six des neuf instances tout en obtenant des résultats identiques sur les trois instances restantes. On remarque aussi que l'AG-NCPX permet d'obtenir plus régulièrement la meilleure solution connue que l'AG-IBX pour toutes les instances de l'ET2. En particulier, l'AG-NCPX obtient la meilleure solution connue à toutes les exécutions pour sept des neuf instances. Si on compare l'AG-NCPX et l'AG-NCPX/IBX, on remarque que l'AG-NCPX/IBX surclasse l'AG-NCPX sur deux instances tout en obtenant des résultats identiques sur les instances restantes. On note aussi que l'AG-NCPX/IBX obtient la meilleure solution connue à toutes les exécutions pour huit des neuf instances. Dans un deuxième temps, en comparant les résultats obtenus par les trois approches proposé à ceux de l'ACS-3D, on remarque que l'ACS-3D surclasse l'AG-IBX sur cinq instances, obtient de moins bon résultats sur deux instances tout en obtenant des performances identiques sur les deux instances restantes. Par contre, lorsque l'on compare maintenant les performances de l'AG-NCPX et l'AG-NCPX/IBX à celle de l'ACS-3D, on note, cette fois ci, que l'AG-NCPX et l'AG-NCPX/IBX obtiennent un nombre moyen de conflits inférieur à celui de l'ACS-3D sur six des neuf instances tout en obtenant des résultats identiques sur les trois autres instances. Si on s'intéresse maintenant au nombre de fois où chaque algorithme arrive à obtenir la meilleure solution connue, on constate qu'à ce niveau aussi, l'AG-NCPX et l'AG-NCPX/IBX dominent l'ACS-3D en obtenant plus régulièrement la meilleure solution connue, l'ACS-3D obtenant, de son coté plus

fréquemment la meilleure solution que l'AG-IBX. Cela se remarque en particulier pour le problème 10_93 où l'AG-NCPX et l'AG-NCPX/IBX parviennent à obtenir la meilleure solution connue respectivement à 45 et 52 reprises comparativement à 18 reprises pour l'ACS-3D et à aucune reprise pour l'AG-IBX.

Instance	Meilleure Solution Connue	AG-IBX		AG-NCPX		AG-NCPX/IBX		ACS-3D	
		Moyenne	# fois meilleure solution	Moyenne	# fois meilleure solution	Moyenne	# fois meilleure solution	Moyenne	# fois meilleure solution
10_93	3	4.00	0	3.55	45	3.48	52	4.03	18
16_81	0	0.65	35	0.03	97	0.00	100	0.58	47
19_71	2	2.26	74	2.00	100	2.00	100	2.04	96
21_90	2	2.25	75	2.00	100	2.00	100	2.02	98
26_82	0	0.01	99	0.00	100	0.00	100	0.00	100
36_92	2	2.41	59	2.00	100	2.00	100	2.03	97
4_72	0	0.00	100	0.00	100	0.00	100	0.01	99
41_66	0	0.00	100	0.00	100	0.00	100	0.00	100
6_76	6	6.00	100	6.00	100	6.00	100	6.00	100

Tableau 4.2 : Résultats obtenus par l'AG-IBX, l'AG-NCPX et de l'AG-NCPX/IBX et l'ACS-3D [Gagné *et al.* 2008] pour l'ET2

En complément, nous avons comparé la convergence des trois algorithmes génétiques sur les différentes instances de l'ET2 et la Figure 4.3 montre l'évolution du nombre moyen de générations pris par l'AG-NCPX, l'AG-IBX et l'AG-NCPX/IBX pour converger vers la meilleure solution en fonction de l'instance. On peut remarquer à l'aide du graphique que la convergence vers la meilleure solution se fait relativement rapidement pour les trois algorithmes. On note toutefois un net avantage pour l'AG-NCPX qui converge beaucoup plus rapidement que l'AG-IBX et l'AG-NCPX/IBX. En effet, l'AG-NCPX converge vers la meilleure solution en moyenne en moins de 90 générations alors que l'AG-NCPX/IBX, de son côté prend 110 générations et l'AG-IBX utilise en moyenne jusqu'à 303 générations. Cette situation peut s'expliquer par l'exploitation de l'information sur les positions ne

générant aucun conflit effectué par l'AG-NCPX qui permet à cet AG de mieux intensifier la recherche que son vis-à-vis l'AG-IBX.

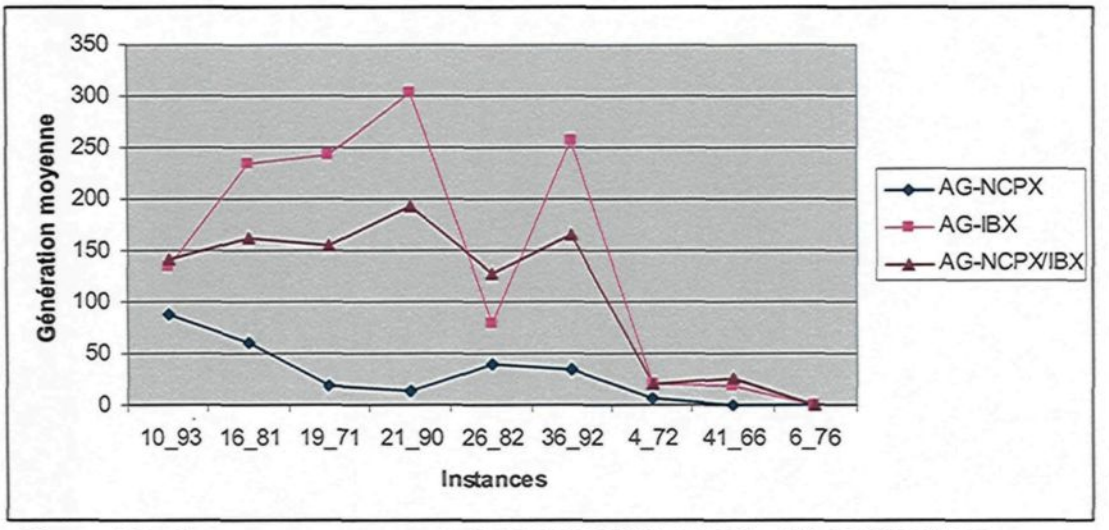

Figure 4.3 : Convergence moyenne de l'AG-NCPX, l'AG-IBX et l'AG-NCPX/IBX sur l'ET2

Le Tableau 4.3 présente, de manière similaire au Tableau 4.2, les résultats des quatre algorithmes pour les 30 instances de l'ET3. On note, en comparant les résultats de l'AG-IBX à ceux de l'AG-NCPX, que ce dernier domine l'AG-IBX sur 26 des 29 instances, obtient des résultats identiques sur 2 (200_06 et 200_08) et inférieurs sur les instances 200_10 et 300_05. La baisse du nombre moyen de conflits obtenu par l'AG-NCPX par rapport à l'AG-IBX varie entre 1 et 100 % pour la totalité des instances de l'ET3. Il est important de préciser ici que, même si ce groupe d'instances est plus difficile, l'AG-NCPX arrive à trouver les meilleures solutions connues pour 22 problèmes sur 30 alors que l'AG-IBX ne l'obtient que pour 11 instances. En comparant maintenant l'AG-NCPX à l'AG-NCPX/IBX, on remarque que l'AG-NCPX/IBX surclasse l'AG-NCPX sur toutes les

instances exceptées cinq où les deux algorithmes obtiennent des performances similaires. Si on compare maintenant les performances des trois approches proposés à celles de l'ACS-3D, on remarque, une fois de plus, que l'AG-NCPX et l'AG-NCPX/IBX obtiennent de meilleurs résultats que l'ACS-3D respectivement pour 26 et 27 instances et ont des performances similaires sur une instance (200_08). En fait, l'ACS-3D obtient de meilleurs résultats que l'AG-NCPX uniquement pour trois instances. Par rapport à l'AG-NCPX/IBX, l'ACS-3D obtient une meilleure moyenne pour seulement deux instances sur les trente. On note, par ailleurs, que l'écart entre les deux algorithmes génétiques et l'ACS-3D semble plus important pour les instances de plus grande taille. Lorsqu'on compare l'ACS-3D à l'AG-IBX, on note, cette fois ci, que l'ACS-3D surclasse l'AG-IBX sur 20 instances, obtient des résultats identiques sur une instance et inférieurs sur les dix instances restantes. En observant maintenant le nombre de fois où chaque algorithme arrive à atteindre la meilleure solution connue, on observe que l'AG-NCPX et l'AG-NCPX/IBX obtiennent, encore une fois, plus régulièrement la meilleure solution connue avec 22 instances sur 30 pour l'AG-NCPX comparativement à 14 instances pour l'ACS-3D et 11 pour l'AG-IBX. On note toutefois que pour le problème 200_02, l'ACS-3D obtient plus régulièrement la meilleure solution connue.

Instance	Meilleure Solution Connue	AG-IBX		AG-NCPX		AG-NCPX/IBX		ACS-3D	
		Moyenne	# fois meilleure solution	Moyenne	# fois meilleure solution	Moyenne	# fois meilleure solution	Moyenne	# fois meilleure solution
200_01	0	3.13	0	1.23	14	1.12	17	2.00	0
200_02	2	3.93	0	2.94	8	2.80	21	2.38	62
200_03	3	11.47	0	7.41	0	7.07	0	7.45	0
200_04	7	7.47	59	7.39	63	7.29	72	7.87	13
200_05	6	7.71	3	6.69	33	6.66	36	7.29	0
200_06	6	6.00	100	6.00	100	6.00	100	6.03	97
200_07	0	3.44	1	0.15	85	0.15	85	0.67	36
200_08	8	8.00	100	8.00	100	8.00	100	8.00	100
200_09	10	11.40	5	10.53	48	10.46	55	10.97	3
200_10	19	20.72	0	21.40	0	21.29	0	20.19	11
300_01	0	5.55	0	2.79	0	2.69	0	3.89	0
300_02	12	14.78	0	12.02	98	12.00	100	12.57	43
300_03	13	15.92	1	13.11	89	13.09	91	13.85	15
300_04	7	10.98	0	7.71	40	7.61	46	8.69	2
300_05	28	39.39	0	42.83	0	41.56	0	42.54	0
300_06	2	8.24	0	5.30	0	5.05	0	5.79	0
300_07	0	3.78	0	0.08	92	0.02	98	0.97	14
300_08	8	9.11	17	8.00	100	8.00	100	8.95	5
300_09	7	9.40	0	7.36	66	7.27	74	8.00	2
300_10	21	32.23	0	28.48	0	27.94	0	32.56	0
400_01	1	2.98	1	1.81	26	1.74	33	3.50	0
400_02	15	23.41	0	19.31	0	19.30	0	23.82	0
400_03	9	12.21	0	10.79	1	10.67	3	13.64	0
400_04	19	20.37	14	19.12	88	19.09	91	20.38	1
400_05	0	5.06	0	0.00	100	0.00	100	2.68	0
400_06	0	4.44	0	0.16	84	0.10	90	1.53	3
400_07	4	5.59	5	4.72	39	4.56	50	8.68	0
400_08	4	7.22	0	4.73	43	4.66	47	12.67	0
400_09	5	17.38	0	10.58	0	10.12	0	16.01	0
400_10	0	4.79	0	0.71	47	0.50	56	2.66	0

Tableau 4.3 : Résultats obtenus par l'AG-IBX, l'AG-NCPX et de l'AG-NCPX/IBX et l'ACS-3D [Gagné *et al.* 2008] pour l'ET3

En analysant maintenant la convergence des trois AG proposés sur l'ET3, on observe une fois de plus, à l'aide de la Figure 4.4, qu'aucun des algorithmes n'utilise les 700 générations disponibles et que l'AG-NCPX converge plus rapidement, en moyenne moins de 200 générations, que l'AG-IBX et l'AG-NCPX/IBX qui eux prennent en moyenne près de 300 générations pour converger.

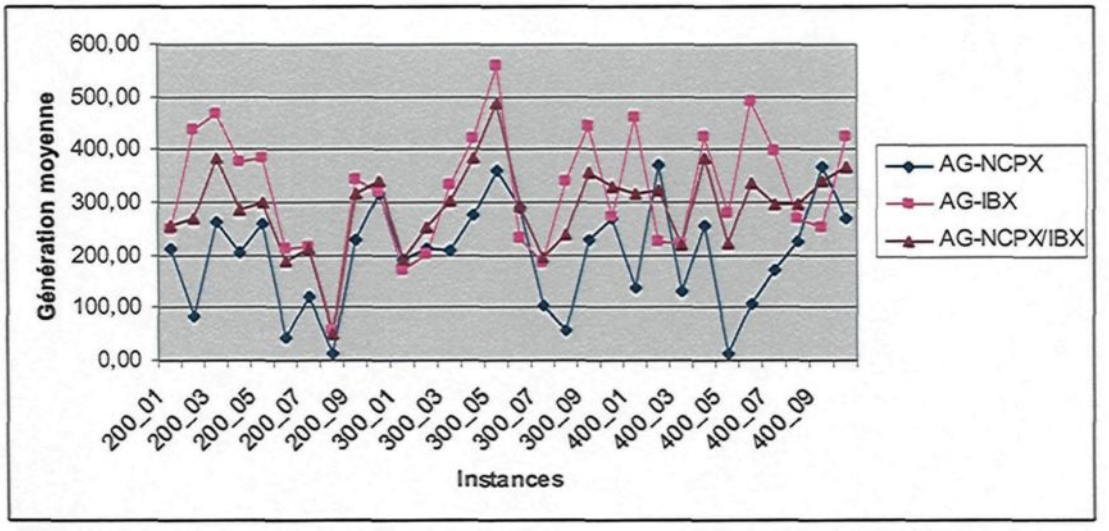

Figure 4.4 : Convergence moyenne de l'AG-IBX, l'AG-NCPX et l'AG-NCPX/IBX sur l'ET3

Les résultats obtenus dans cette première série d'expérimentation permettent de conclure, d'une part, que l'AG-NCPX/IBX obtient les meilleures performances sur l'ensemble de problèmes testés suivi par l'AG-NCPX et l'AG-IBX. Les bonnes performances du croisement NCPX par rapport au croisement IBX s'explique sans doute par l'exploitation de l'information sur les positions non conflictuelles opérées par le croisement NCPX. En effet, cette particularité du croisement NCPX semble lui permettre « d'apprendre » au fil des générations afin de l'aider à utiliser l'expérience passée pour intensifier plus rapidement la recherche par rapport au croisement IBX. Ces résultats montrent aussi que la combinaison des deux opérateurs de croisement dans un seul algorithme permet un gain au niveau des performances. En effet, l'utilisation simultanée des deux opérateurs de croisement semble apporter une diversité supplémentaire dans la population et ainsi évite une convergence trop rapide de l'algorithme. D'autre part, la

comparaison des performances des approches proposées par rapport à d'autres approches de la littérature permet de conclure que les AG sont des techniques d'optimisation efficaces pour résoudre le POV lorsque les spécificités du problème sont bien prises en compte. En effet, les performances de l'AG-NCPX/IBX et de l'AG-NCPX se comparent avantageusement à ceux des meilleurs résultats publiés à ce jour dans la littérature.

4.5.3 Ajout d'une procédure de recherche locale

Dans les expérimentations numériques suivantes, une procédure de recherche locale est combinée à l'AG-NCPX/IBX afin de comparer la performance de l'AG-NCPX/IBX avec recherche locale (AG-NCPX/IBX + LS) à celle de l'ACS-3D, lui aussi avec recherche locale. Dans l'ACS-3D, une procédure de recherche locale est appliquée sur la meilleure solution trouvée à chaque cycle ainsi que sur la meilleure solution globale à la fin de l'algorithme [Gagné *et al.* 2008]. Dans l'AG-NCPX/IBX, la procédure de recherche locale est appliquée avec une probabilité de 20% pendant les 250 premières générations de l'algorithme à la meilleure solution courante trouvée lorsque cette solution est améliorée ainsi que sur la meilleure solution obtenue par l'algorithme à la fin des 700 générations. Les Tableaux 4.4 et 4.5 présentent respectivement les résultats obtenus par les deux algorithmes avec recherche locale pour l'ET2 et l'ET3. En ce qui concerne les meilleurs résultats obtenus, ils sont indiqués avec une trame en gris. Les instances de l'ET1 ne sont pas utilisées pour comparer les deux algorithmes avec recherche locale car les versions sans recherche locale des deux algorithmes solutionnent presque instantanément toutes ces instances. On note, par ailleurs, qu'une colonne supplémentaire est ajoutée dans les deux tableaux afin de montrer les résultats que l'on pourrait obtenir si on laissait l'AG-

NCPX/IBX + LS s'exécuter pendant dix minutes au lieu de limiter le nombre maximum de générations alloué à 700 générations comme dans les expérimentations précédentes. En effet, les paramètres précédemment utilisés avaient été déterminés de manière à obtenir une comparaison la plus équitable possible avec l'ACS-3D. En examinant les résultats des deux tableaux, on remarque que l'addition des procédures de recherche locale améliore nettement les performances de l'AG-NCPX/IBX et de l'ACS-3D sur les deux ensembles tests. En comparant maintenant la performance des deux algorithmes sur l'ET2, on remarque que l'AG-NCPX/IBX + LS obtient la meilleure solution connue à chaque exécution sur tous les problèmes sauf l'instance 10_93, l'écart pour ce problème est toutefois minime. L'ACS-3D n'obtenant, pour sa part, pas la meilleure solution connue pour les problèmes 10_93 et 16_81, l'écart pour le problème 16_81 est toutefois négligeable. L'AG-NCPX/IBX surclasse l'ACS-3D sur le problème 10_93 en obtenant la meilleure solution connue 97 fois sur 100 comparativement à 63 fois pour l'ACS-3D. Lorsqu'on exécute l'AG-NCPX/IBX + LS pendant dix minutes on obtient la meilleure solution connue à toutes les exécutions pour toutes les instances de l'ET2.

Instance	Meilleure Solution Connue	ACS-3D + LS		AG-NCPX/IBX+ LS		AG-NCPX/IBX + LS (10min)	
		Moyenne	# fois meilleure solution	Moyenne	# fois meilleure solution	Moyenne	# fois meilleure solution
10_93	3	3.37	63	3.03	97	3.00	100
16_81	0	0.03	97	0.00	100	0.00	100
19_71	2	2.00	100	2.00	100	2.00	100
21_90	2	2.00	100	2.00	100	2.00	100
26_82	0	0.00	100	0.00	100	0.00	100
36_92	2	2.00	100	2.00	100	2.00	100
4_72	0	0.00	100	0.00	100	0.00	100
41_66	0	0.00	100	0.00	100	0.00	100
6_76	6	6.00	100	6.00	100	6.00	100

Tableau 4.4 : Résultats comparatifs de l'ACS-3D + LS [Gagné *et al.* 2008] et l'AG-NCPX/IBX + LS sur l'ET2

Quand on compare maintenant les résultats sur l'ET3 à l'aide du Tableau 4.5, on constate, une fois de plus, que l'AG-NCPX/IBX surclasse l'ACS-3D sur la majorité des instances, à savoir 23 instances au total. Pour les sept instances restantes, l'ACS-3D et l'AG-NCPX/IBX obtiennent des résultats identiques. On note que lorsqu'on ne limite pas le nombre de générations à 700 l'AG-NCPX/IBX + LS parvient à trouver la meilleure solution connue à chaque exécution pour 28 des 30 instances de l'ET3.

Instance	Meilleure Solution Connue	ACS-3D + LS		AG-NCPX/IBX+ LS		AG-NCPX/IBX+ LS (10min)	
		Moyenne	# fois meilleure solution	Moyenne	# fois meilleure solution	Moyenne	# fois meilleure solution
200_01	0	0.41	61	0.14	86	0.00	100
200_02	2	2.00	100	2.00	100	2.00	75
200_03	3	5.76	0	5.13	0	3.25	100
200_04	7	7.00	100	7.00	100	7.00	100
200_05	6	6.16	84	6.00	100	6.00	100
200_06	6	6.00	100	6.00	100	6.00	100
200_07	0	0.00	100	0.00	100	0.00	100
200_08	8	8.00	100	8.00	100	8.00	100
200_09	10	10.00	100	10.00	100	10.00	100
200_10	19	19.02	98	19.00	96	19.00	100
300_01	0	1.87	1	0.08	92	0.00	100
300_02	12	12.01	99	12.00	100	12.00	100
300_03	13	13.02	98	13.00	100	13.00	100
300_04	7	7.70	42	7.26	74	7.00	100
300_05	28	32.53	0	32.08	0	28.90	10
300_06	2	3.59	10	2.55	50	2.00	100
300_07	0	0.08	92	0.00	100	0.00	100
300_08	8	8.00	100	8.00	100	8.00	100
300_09	7	7.18	82	7.03	97	7.00	100
300_10	21	22.25	16	21.30	69	21.00	100
400_01	1	2.55	1	1.29	69	1.00	100
400_02	15	17.46	2	16.75	4	15.00	100
400_03	9	10.08	4	10.07	6	9.00	100
400_04	19	19.01	99	19.00	100	19.00	100
400_05	0	0.02	98	0.00	100	0.00	100
400_06	0	0.10	90	0.00	100	0.00	100
400_07	4	5.58	5	4.09	91	4.00	100
400_08	4	5.24	21	4.00	100	4.00	100
400_09	5	7.31	2	6.69	9	5.00	100
400_10	0	0.89	28	0.00	100	0.00	100

Tableau 4.5 : Résultats comparatifs de l'ACS-3D + LS [Gagné *et al.* 2008] et l'AG-NCPX/IBX + LS sur l'ET3

4.6 Conclusion

Dans ce chapitre, nous avons proposé un algorithme génétique permettant de résoudre efficacement le problème théorique d'ordonnancement de voitures. En particulier, deux nouveaux opérateurs de croisement spécifiques pour la résolution du problème ont été proposés. Les performances des différents opérateurs de croisement ont été validées à l'aide de 3 ensembles test contenant au total 109 instances publiées. Les résultats expérimentaux ont montré, dans un premier temps, que le l'AG avec croisement NCPX obtenait les meilleurs résultats sur les instances testées par rapport au second opérateur de croisement proposé dans ce chapitre. Dans un deuxième temps, Nous avons aussi montré l'intérêt de combiner ces deux opérateurs de croisement. En effet, la combinaison des deux opérateurs de croisement dans un même algorithme permet d'obtenir des résultats se comparant avantageusement par rapport à ceux d'autres algorithmes réputés de la littérature pour ce problème. Chacun des travaux réalisés montre que nous avons su prendre en compte des connaissances du domaine afin de développer des opérateurs de croisement les mieux adaptés. Les résultats des différentes approches ont aussi permis de remarquer l'importance d'utiliser des opérateurs génétiques dédiés à la problématique et ce, même lorsque l'emploi d'opérateurs naturels est possible.

Finalement, les dernières expérimentations ont consisté à ajouter une procédure de recherche locale à l'algorithme génétique afin, d'une part, de confirmer les performances de l'approche proposée par rapport à une autre approche intégrant elle aussi des procédures de recherche locale. D'autre part, cet ajout a aussi permis de mettre en évidence l'importance des mécanismes d'intensification sur la qualité globale des solutions obtenues. Une fois de

plus, les résultats obtenus sont probants. En effet, l'hybridation d'une procédure de recherche locale à l'AG-NCPX/IBX améliore de façon notable ses performances qui restent supérieures à celles de l'autre algorithme en compétition. Ces derniers résultats sont encourageants car les mécanismes d'intégration utilisés dans ce chapitre pour incorporer la procédure de recherche locale sont très simplistes et laisse beaucoup de place à l'amélioration. Cette étape représente donc un premier pas vers l'intégration d'algorithmes d'intensification plus évolués de type heuristiques ou méthodes exactes. Le chapitre suivant approfondit cette avenue de recherche en proposant des approches de résolutions coopératives combinant un algorithme génétique avec d'autres méthodes d'intensification de la recherche.