CHAPITRE 5

ALGORITHMES COOPÉRATIFS POUR LE PROBLÈME THÉORIQUE D'ORDONNANCEMENT DE VOITURES

5.1 Introduction

Le chapitre précédent a permis de présenter une nouvelle approche permettant de résoudre efficacement le problème théorique d'ordonnancement de voitures en utilisant un AG. Les expérimentations numériques ont montré que l'algorithme proposé offre un bon compromis entre intensification et diversification du processus recherche notamment par l'intermédiaire de nouveaux opérateurs de croisement, de différentes méthodes de remplacement ainsi que par l'application de plusieurs opérateurs de mutation. Toutefois, l'ajout d'une procédure de recherche locale à l'AG proposé a permis d'améliorer de façon notable ses performances et de montrer l'intérêt de combiner cette approche avec d'autres méthodes d'intensification de la recherche afin de le rendre plus performant et robuste. Ce genre de combinaison communément appelée hybridation représente une avenue de recherche prometteuse sur laquelle de nombreux chercheurs se penchent depuis quelques années [Talbi 2002; Barichard 2003; Basseur 2004]. L'hybridation consiste à combiner les avantages de différentes méthodes d'optimisation en un seul algorithme. Selon Talbi [2002], ce type d'approche a permis d'obtenir de très bons résultats dans une grande variété de problèmes théoriques d'optimisation combinatoire tels le problème du voyageur de commerce et le problème d'assignation quadratique ainsi que dans des contextes réels. De nombreuses approches hybrides ont alors vu le jour, la plupart hybridant des métaheuristiques à base de populations avec des métaheuristiques à solution unique. Selon Blum *et al.* [2005], l'hybridation de métaheuristiques est la voie la plus prometteuse pour l'amélioration de la qualité des solutions dans beaucoup d'applications.

Récemment, un nombre croissant de travaux proposent de faire coopérer des métaheuristiques et des méthodes exactes [Barichard 2003; Basseur 2004]. D'une manière générale, les approches de résolution par des méthodes exactes se limitent à de petites instances pour les problèmes d'optimisation combinatoire difficiles. Par contre, la combinaison métaheuristique /méthode exacte peut devenir une alternative très efficace, car les deux méthodes ont des particularités bien différentes et associent leurs avantages pour produire de meilleurs résultats [Basseur 2004]. Malheureusement, selon Jussien et Laburthe [2004], ce type d'hybridation n'est pas une tâche aisée autant pour des raisons culturelles que technologiques.

Dans ce chapitre, nous proposons d'explorer certains mécanismes d'hybridation entre une métaheuristique à base de population et une méthode exacte pour résoudre le problème théorique d'ordonnancement de voitures sur une chaîne d'assemblage (*car-sequencing problem*). Le chapitre est organisé de la manière suivante : à la Section 5.2, nous présentons les principales classifications des stratégies d'hybridation métaheuristiques/méthode exactes. La Section 5.3 décrit la forme d'hybridation proposée entre un algorithme génétique et la programmation linéaire en nombres entiers. À la Section 5.4, nous présentons les résultats numériques obtenus sur les instances du problème théorique d'ordonnancement de voitures utilisées au chapitre précédent. Finalement, quelques conclusions et remarques sont présentées à la dernière section.

5.2 Classification des stratégies d'hybridation métaheuristiques/méthodes exactes

Les études sur l'hybridation entre métaheuristiques et méthodes exactes sont moins courantes que celle portant sur l'hybridation de deux métaheuristiques. Ce qui fait qu'il existe peu de classification pour ce type d'approche [Dimistrescu et Stützle 2003; Basseur 2004]. Toutefois, l'intérêt grandissant des chercheurs pour ce type d'approche et le nombre croissant de méthodes proposées ont récemment amené certains auteurs à proposer des classifications pour ce type de coopération. Parmi ces classifications on peut citer, sans être exhaustif, la classification de Dimitrescu et Stützle [2003], de Basseur [2004] ou encore de Puchinger et Raidl [2005].

Dans leur classification, Dimitrescu et Stützle [2003] distinguent cinq types d'approches de coopération métaheuristiques/méthodes exactes et détaillent une application trouvée dans la littérature pour chaque type d'approche. La classification proposée par les auteurs sépare les approches selon la méthode utilisée pour la coopération. Les cinq classes proposées par Dimitrescu et Stützle regroupent :

- les approches utilisant un algorithme exact pour explorer de larges voisinages dans un algorithme de recherche locale;
- les approches qui utilisent une méthode heuristique afin de réduire l'espace de recherche de la méthode exacte;
- les approches qui exploitent les bornes de la méthode exacte pour une heuristique constructive;

- les approches utilisant les informations fournies par les relaxations des problèmes linéaires pour orienter un algorithme de recherche locale ou constructif; et
- les approches utilisant une méthode exacte pour remplacer une fonction spécifique de la métaheuristique.

On note toutefois que même si l'état de l'art de Dimitrescu et Stützle survole les principales stratégies de coopération entre méthodes exactes et métaheuristiques, la plupart des exemples proposés par les auteurs utilisent des méthodes de recherches locales ou des algorithmes de recherche avec tabou comme métaheuristiques.

Dans sa thèse de doctorat, Basseur [2004] propose une classification plus détaillée pour catégoriser les méthodes hybridant métaheuristiques et méthodes exactes. Cette classification de type taxonomie s'inspire de celle proposée par Talbi [2002] pour classifier les métaheuristiques hybrides selon leur niveau d'hybridation. Ainsi, une hybridation est dite de *bas niveau* lorsque les éléments fonctionnels qui constituent une métaheuristique sont modifiés pour être remplacés par une méthode exacte et inversement. À l'opposé, une hybridation est dite de *haut niveau* lorsque l'intégrité des méthodes hybridées est conservée. Ces deux niveaux peuvent par la suite être subdivisés en deux modes : un *mode relais* et un *mode simultané*. En mode relais, les 2 méthodes hybridées opèrent l'une à la suite de l'autre en utilisant la sortie de la précédente comme entrée à la manière d'un pipeline. En mode simultané par contre, les 2 approches explorent en parallèle l'espace de recherche.

La dernière classification présentée, qui sera utilisée dans la suite de ce travail, est celle proposée par Puchinger et Raidl [2005]. Dans cette classification, les auteurs divisent les

approches hybridant métaheuristiques et méthodes exactes en deux classes : *les hybridations de type collaboratives* et *les hybridations de type intégratives*.

Les approches hybrides de type collaboratives regroupent les stratégies d'hybridation ou les méthodes hybridées échangent de l'information de manière séquentielle, parallèle ou entrelacée tout en conservant chacune leur intégrité. Au niveau des stratégies d'hybridations intégratives, une des deux approches est modifiée afin d'incorporer l'autre sous un modèle maître/esclave.

5.3 Algorithme génétique hybride

Dans ce chapitre, on propose d'hybrider un algorithme génétique avec un programme linéaire en nombre entiers (PLNE). Un PLNE consiste à formuler de façon mathématique un problème d'optimisation. Pour cela, on définit une fonction objectif linéaire à maximiser ou minimiser en fonction de contraintes, qui sont, elles aussi, exprimées de manière linéaire. Les variables de décisions doivent être entières et les solutions du programme doivent respecter les différentes contraintes du problème [Nedzela 1990]. La Figure 5.1 présente le programme linéaire proposé par Gravel *et al.* [2005] pour résoudre le POV.

Minimiser

$$F = \sum_{k=1}^{opt} \sum_{j=1}^{nc-sk+1} Y_{kj} \qquad (1)$$

Sous Contrainte

$$\sum_{v=1}^{V} C_{vj} = 1 \qquad pour\ j = 1..\ nc \qquad (2)$$

$$\sum_{j=1}^{nc} C_{vj} = c_v \qquad pour\ v = 1..\ V \qquad (3)$$

$$\sum_{v=1}^{V} \sum_{l=j}^{j+sk-1} O_{vk} * C_{vl} \leq r_k + M * Y_{kj} \qquad pour\ k = 1..\ opt\ et\ \ j = 1..\ (nc - s_k + 1) \qquad (4)$$

C_{vj} booléen $\quad pour\ v = 1..\ V\ et\ j = 1..\ nc$

Y_{kj} booléen $\quad pour\ k = 1..\ opt\ et\ j = 1..\ nc - s_k + 1$

Figure 5.1 : Un PLNE pour le POV [Gravel *et al,* 2005]

Dans la fonction objectif (1), les auteurs cherchent à minimiser le nombre de fois où une contrainte de capacité, exprimée à l'aide d'un ratio r_k/s_k pour une option k, sera dépassée en considérant chaque sous séquence de longueur s_k. Par la suite, la Contrainte (2) empêche de placer plus d'une classe de voitures à chaque position de la séquence. La Contrainte (3) permet de s'assurer que toutes les contraintes de production du problème sont respectées. Finalement, la dernière contrainte formalise les contraintes de capacité du problème tout en permettant de les enfreindre lorsqu'aucune solution sans conflit ne peut être trouvée.

Compte tenu de la difficulté du POV, le PLNE proposé par Gravel *et al.* [2005] n'a été testé que sur les instances de l'ET1 et l'ET2. Si les résultats de l'approche ont permis de confirmer les meilleures solutions connues pour certains problèmes, ils mettent aussi en évidence les limites de ce type d'approche lors de la résolution d'instances plus difficiles (plus de 16h d'exécution pour trouver la meilleure solution connue pour l'instance 10_93). Ce PLNE obtient donc des résultats intéressants sur le POV et il utilise la fonction

d'évaluation classique de la littérature. C'est pour cette raison que nous avons choisi ce modèle comme méthode de résolution exacte dans cette partie de la thèse.

La stratégie d'hybridation introduite dans ce chapitre vise donc à intégrer le PLNE de Gravel *et al.* [2005] lors du processus de croisement pour ainsi créer un *croisement hybride*. Deux croisements hybrides sont ainsi proposés respectivement à partir des opérateurs *NCPX* et *IBX* présentés au chapitre précédent.

La première étape du *croisement hybride A* consiste à sélectionner k_{mov} positions dans le premier parent x_1 pour les transférer dans l'enfant y_1 en cours de création (voir Figure 5.2). Ces positions sont marquées comme positions libres, ce qui veut dire que les classes de voitures situées à ces positions peuvent, par la suite, être déplacées dans la séquence. Dans la seconde étape, l'opérateur de croisement *NCPX* est ensuite utilisé pour compléter les positions restantes de l'enfant y_1 sans tenir compte des k_{mov} positions marquées. À l'étape 3, on cherche, parmi les voisins potentiels de y_1, celui qui est à moindre coût. Pour cela, on résout le PLNE proposé par Gravel *et al.* [2005] où $|x|$- k_{mov} variables de la séquence sont fixées. De manière générale, la formulation du PLNE consiste à affecter une classe de voitures à chacune des positions de la séquence en cherchant à minimiser le nombre de conflits générés par les contraintes d'espacement. Pour ce croisement, le PLNE détermine les classes de voitures à placer dans les positions libres de la séquence. Le lecteur peut consulter Gravel *et al.* [2005] pour la modélisation complète du programme linéaire en nombres entiers. Les mêmes étapes sont réalisées avec le deuxième parent de manière à créer un autre enfant.

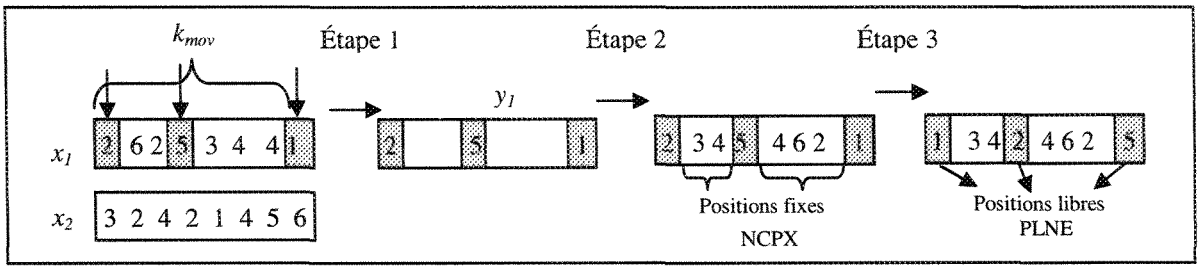

Figure 5.2 : Illustration du fonctionnement du croisement hybride A

L'efficacité de ce type de croisement dépend essentiellement de deux éléments : le nombre de positions k_{mov} du sous-problème à résoudre et le temps maximum T_{max} alloué au solveur pour résoudre le PLNE. En effet, un k_{mov} trop élevé ou un T_{max} trop petit risque de faire échouer la résolution du PLNE dans le temps imparti. D'un autre côté, un k_{mov} trop faible ne laisse pas assez de latitude au PLNE pour améliorer la solution courante. Afin de contourner ce problème, l'idée utilisée consiste à initialiser k_{mov} à une faible valeur et d'accroître par la suite cette valeur de manière constante au fil des générations. Lorsqu'une résolution échoue, la valeur de k_{mov} est décrémentée de manière à revenir à la dernière valeur ayant fourni une solution. Notons ici que dès que la valeur de k_{mov} est décrémentée le nombre de positions à déplacer est stabilisé à cette nouvelle valeur et ne variera plus jusqu'à la fin de l'algorithme comme le suggère Estellon *et al.* [2007] dans leur article. On veut ainsi permettre le plus grand choix de positions libres possibles pour une valeur T_{max} donnée.

Un autre point crucial, pour la réussite de ce croisement, concerne le choix des positions libres. Deux stratégies sont utilisées : la première consiste à choisir aléatoirement un pourcentage de positions faisant partie d'un conflit et de compléter avec des positions choisies aléatoirement tandis que la seconde stratégie consiste tout simplement à choisir

aléatoirement les k_{mov} positions de manière à ce que 2 positions j et j' sélectionnées soient séparées d'au moins s_{max} positions, où s_{max} correspond au plus grand dénominateur parmi toutes les contraintes d'espacement. Cette propriété a été à l'origine proposée par Estellon et al. [2005] et permet d'affirmer que la solution optimale du problème linéaire réduit est une solution entière. Il est aussi important de noter ici que, lors de la sélection des positions libres, on s'assure qu'au moins une des positions choisies fait partie d'un conflit.

Le *croisement hybride B* consiste à déterminer deux points de coupure aléatoirement dans chacun des parents sélectionnés x_1 et x_2 (voir Figure 5.3). Afin de créer le premier enfant, les classes de voitures comprises entre les deux points de coupure de x_1 sont directement copiées dans l'enfant y_1. La deuxième étape consiste à sélectionner aléatoirement un des deux points de coupure et à réorganiser les classes de voitures présentes à l'extérieur de ce point à l'aide de l'opérateur de croisement *IBX*. Finalement, à l'étape 3, les k positions situées à l'extérieur de l'autre point de coupure sont marquées comme libres et celles-ci sont ensuite réorganisées avec les classes de voitures restantes en utilisant le PLNE. Les mêmes étapes sont réalisées avec le deuxième parent de manière à créer un autre enfant.

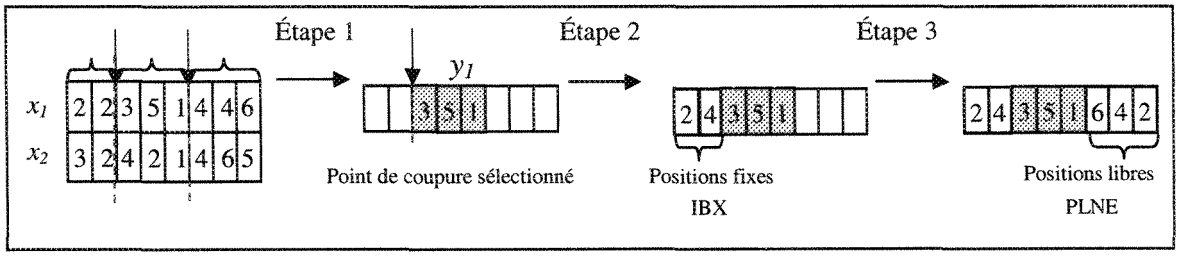

Figure 5.3 : Illustration du fonctionnement du croisement hybride B

L'algorithme 5.1 présente le fonctionnement général de l'algorithme génétique hybride. Pour le cas de l'algorithme génétique sans l'hybridation proposée, seule la phase 2 est alors requise. Dans le fonctionnement de l'AG hybride, la création de la population d'enfants à la génération t (Q_t) se fait en 3 phases à partir de la population de parents (Pop_{t-1}) : une *phase mixte* (300^{ières} générations), une *phase de diversification* (de la 301^{ième} à la 650^{ième} génération) et une *phase d'intensification* de la recherche (pendant les 50 dernières générations). La durée de chacune de ces trois phases a été fixée en relation avec les convergences moyennes obtenues par les AG proposés au chapitre précédent pour le même problème. La Phase 1 (phase mixte) consiste à créer, selon une probabilité $Prob_{ILP}$ fixée à 50%, des enfants en utilisant le croisement hybride (*Croisement Hybride A ou B)* pour le premier 10 % de la population et avec le croisement *NCPX* ou *IBX* selon le cas pour le reste de la population. La Phase 2 (phase de diversification), de son côté, génère des enfants uniquement en utilisant le croisement *NCPX* ou *IBX* selon le cas. Finalement, la Phase 3 (phase d'intensification) de l'algorithme intensifie la recherche en générant des enfants avec le Croisement Hybride *A ou B* uniquement. On note que, comme pour l'AG proposé au chapitre précédent, le critère d'arrêt utilisé dans ce chapitre correspond au nombre maximum de générations allouées à l'AG hybride.

Algorithme 5.1 : Algorithme génétique hybride pour le POV

Création aléatoire de la population initiale POP_0
Évaluer chaque individu $x \in POP_0$ et trier POP_0
Tant qu'aucun critère d'arrêt rencontré
  Tant que $|Q_t| < \lambda$
    Si p_c **alors**
      En Fonction de la phase
        Cas (*Phase 1*)
          Si $|Q_t| < 10$ % de λ **et** $\text{Prob}_{ILP} \leq 0.5$ **alors**
            Sélectionner un parent et créer un enfant à l'aide du Croisement Hybride *A ou B*
          Sinon
            Sélectionner les parents et créer 2 enfants x et y avec le *NCPX ou IBX*
        Cas (*Phase 2*)
            Sélectionner les parents et créer 2 enfants x et y avec le *NCPX ou IBX*
        Cas (*Phase 3*)
            Sélectionner un parent et créer un enfant à l'aide du Croisement Hybride *A ou B*
          Évaluer les enfants créés
        Fin Si
      Tirer aléatoirement un nombre *rnd* entre *0* et *1*
      Si $rnd < p_m$ **alors**
          Effectuer une mutation sur l'individu sélectionné et évaluer l'individu muté
      Fin Si
      Effectuer une sélection inverse avec l'enfant y s'il existe
      Ajouter x à Q_t
  Fin Tant que
  Trier $Q_t \cup POP_t$
  Conserver les μ meilleurs individus de $Q_t \cup POP_t$ pour former POP_{t+1}
Fin Tant que
Retourner le meilleur individu trouvé

5.4 Résultats et discussion

Les différents algorithmes ont tous été implémentés en C^{++} en utilisant CPLEX 10.0 comme librairie de programmation linéaire en nombres entiers. La machine utilisée pour les expérimentations numériques est un ordinateur équipé de deux processeurs Itanium 64 bits cadencés chacun à 1.4 GHz avec 4 Go de mémoire vive et tournant sous Windows Server 2003. Les paramètres N, λ, p_c, p_m et *NbGen* qui représentent respectivement la taille de la population de parents, la taille de la population d'enfants, la probabilité de croisement, la probabilité de mutation et le nombre maximum de générations de l'algorithme sont fixés

respectivement à 250, 200, 0.8, 0.09, 700 pour tous les problèmes testés. La phase 1 se termine à la $300^{ième}$ génération, la phase 2 se termine à la $650^{ième}$ génération et les 50 dernières générations constituent la $3^{ième}$ phase. Il faut toutefois noter que l'algorithme s'arrête lorsqu'une solution sans conflit est trouvée. Finalement, on note que le temps maximum alloué à CPLEX pour résoudre un PLNE est fixé à trois secondes dans la Phase 1 de l'algorithme et à dix secondes dans la Phase 3. On veut ainsi laisser plus de temps à CPLEX pour améliorer la solution courante dans la phase d'intensification.

Le Tableau 5.1 présente les expérimentations effectuées afin de déterminer le temps maximum alloué à CPLEX pendant la phase 1 et la phase 3. Ces résultats correspondent à une moyenne de 5 exécutions et sont obtenus en utilisant le problème 300_05 qui est, selon Estellon *et al.*[2007], un des plus difficiles à résoudre de l'ET3. À partir de ce tableau, on note que les meilleurs résultats en termes de qualité de solution et de temps d'exécution sont obtenus en accordant trois secondes à CPLEX dans la Phase 1 et dix secondes dans la Phase 3. En ce qui concerne la taille de k_{mov}, on note que pour chaque cas présenté au Tableau 5.3 elle varie en moyenne entre 57 et 71.5 positions. On rappelle que pour cette série d'expérimentation la taille initiale de k_{mov} est fixée à 15 positions (5% * 300 voitures) et qu'elle est par la suite incrémentée d'une position jusqu'à ce par le PLNE résolution échoue.

Limite de temps allouée à CPLEX		Moyenne	Temps (s)	k_{mov} maximum
Phase 1	Phase 3			
3	10	33.0	2601.9	70
5	10	33.6	2702.4	70
10	10	33.0	3346.8	71.5
3	3	38.5	1864.0	57
3	5	37.6	2380.6	66

Tableau 5.1 : Résultats obtenus en faisant varier le temps alloué à CPLEX dans la Phase 1 et la Phase 3 pour le problème 300_05

5.4.1 Jeux d'essai

La performance des AG hybrides est évaluée à l'aide des deux ensembles de problèmes de la librairie CSPLib (http://www.csplib.org/) non résolus de manière triviale au chapitre précédent, l'ET2 et l'ET3.

5.4.2 Comparaison expérimentale

Dans la série d'expérimentations suivantes, nous avons utilisé trois versions de notre algorithme hybride : une version avec croisement IBX ($ILPGA^{IBX}$), une version utilisant le croisement NCPX ($ILPGA^{NCPX}$) et une version utilisant les deux opérateurs de croisement ($ILPGA^{NCPX/IBX}$). Les performances des approches hybrides proposées dans cette thèse sont comparées aux résultats des deux algorithmes hybrides proposés Gravel *et al.* [2008] (*ACS-IH* et *GA-IH*), ainsi qu'à ceux du *VLNS* (very large neighbourhood search) de Estellon *et al.* [2007] et de l'approche hybride (*VNS* et *PLNE*) de Prandtstetter et Raidl [2007]. Le Tableau 5.2 résume les résultats obtenus par les différents algorithmes pour les neuf instances de l'ET2. On retrouve respectivement dans ce tableau, le nom de l'instance, la valeur de la meilleure solution connue et pour chaque algorithme le nombre moyen de conflits (*moyenne*) ainsi que le temps moyen d'exécution en secondes (*temps*) des six premiers algorithmes. Il est important de noter que les temps moyens d'exécution rapportés

ici correspondent au temps pris par chaque algorithme pour effectuer toutes les générations sauf dans le cas où une solution sans conflit est trouvée. En ce qui concerne les meilleurs résultats obtenus, ils sont indiqués en caractères gras.

Les expérimentations numériques de l'AG^{NCPX} et de l'AG^{IBX} ayant été effectuées en utilisant un ordinateur différent, nous avons donc utilisé les références suivantes : http://www.spec.org/cpu/results/res2004q2/cpu2000-20040329-02938.html et http://www.spec.org/cpu/results/res2004q2/cpu2000-20040614-03077.html afin de déterminer le ratio de performances entre nos deux ordinateurs. À l'aide de ces deux sites Internet, on note que le temps d'exécution sur un ordinateur équipé de deux processeurs Itanium 64 bits cadencés chacun à 1.4 GHz avec 4 Go de mémoire vive et tournant sous Windows Server 2003 est en moyenne de 16% à 43% plus lent qu'un ordinateur équipé d'un processeur Pentium Xeon 3.6 Ghz avec 1 Go de mémoire vive et tournant sous Windows XP. Nos propres expérimentations numériques montrent que, pour les AG présentés au chapitre précédent, ce ratio est d'environ 33%. Ainsi, les temps d'exécution de l'AG^{NCPX} et de l'AG^{IBX} présentés au Tableau 5.2 doivent être augmentés de 33% pour une comparaison équitable avec l'*ILPGA*^{NCPX} et l'*ILPGA*^{IBX}.

Chaque instance a été résolue 10 fois par l'*ILPGA*^{NCPX}, l'*ILPGA*^{IBX}, le *VLNS* [Estellon *et al.* 2007] et le *VNS/ILP* [Prandtstetter et Raidl 2007], 100 fois par l'AG^{NCPX} et l'AG^{IBX} et 5 fois par l'*ACS-IH* et l'*AG-IH*. Le lecteur peut consulter Gravel *et al.* [2008] pour plus de détails sur l'implémentation de l'*ACS-IH et de l'AG-IH*.

On note que, pour le *VNS/ILP*, les auteurs [Prandtstetter et Raidl 2007] n'indiquent ni le type d'hybridation utilisée ni les temps d'exécution de leur approche. Finalement, il est

important de mentionner qu'il est impossible de comparer les temps d'exécution du *VLNS* avec ceux de nos approches car les auteurs présentent uniquement les temps de calcul pris par leur algorithme pour atteindre le nombre moyen de conflits présenté dans leur article.

L'analyse comparative des résultats de l'AG^{NCPX} et de l'$ILPGA^{NCPX}$ montre que l'approche hybride obtient la meilleure solution connue à chaque exécution pour toutes les instances de l'ET2 tandis que l'AG^{NCPX} trouve la meilleure solution connue pour 7 des 9 instances. Lorsque l'on compare maintenant l'AG^{IBX} avec l'$ILPGA^{IBX}$, on note que l'approche hybride obtient de meilleurs résultats pour 6 instances tout en obtenant la meilleure solution connue à chaque exécution pour 8 des 9 instances. Lorsqu'on compare maintenant les deux algorithmes hybrides, on constate que l'$ILPGA^{NCPX}$ obtient de meilleurs résultats que l'$ILPGA^{IBX}$ pour une instance tout en obtenant des résultats identiques pour les 8 instances restantes. On note aussi que l'$ILPGA^{NCPX}$ s'exécute plus rapidement que l'$ILPGA^{IBX}$ pour toutes les instances de l'ET2. À l'aide du Tableau 5.2, on remarque aussi que l'$ILPGA^{NCPX}$, l'AG-IH, l'ACS-IH et le VLNS obtiennent des résultats identiques (*moyenne*) pour toutes les instances à l'exception de l'instance 10-93 pour laquelle l'ACS-IH est moins bon que les 3 autres algorithmes. Finalement, le VNS/ILP obtient les pires résultats pour toutes les instances à l'exception de l'instance 41_66.

		AG^{NCPX}		$ILPGA^{NCPX}$		AG^{IBX}		$ILPGA^{IBX}$		*ACS-IH*		*GA-IH*		*VLNS*	*VNS/ILP*
Instance	**Meilleure Solution Connue**	**Moyenne**	**Temps (s)**	**Moyenne**	**Temps (s)**	**Moyenne**	**Temps (s)**	**Moyenne**	**Temps (s)**	**Moyenne**	**Temps (s)**	**Moyenne**	**Temps (s)**	**Moyenne**	**Moyenne**
10_93	3	3.55	42.3	**3.00**	585.1	4.00	45.3	**3.20**	626.9	3.40	1757.6	**3.00**	2567.4	**3.00**	16.7
16_81	0	0.03	4.6	**0.00**	28.8	0.65	39.2	**0.00**	71.8	**0.00**	37.4	**0.00**	46.2	**0.00**	19.6
19_71	2	**2.00**	38.4	**2.00**	529.8	2.26	43.3	**2.00**	911.8	**2.00**	2796.4	**2.00**	3286.4	**2.00**	9.5
21_90	2	**2.00**	34.8	**2.00**	618.7	2.25	40.0	**2.00**	707.7	**2.00**	2758.6	**2.00**	3490.8	**2.00**	4.1
26_82	0	**0.00**	2.0	**0.00**	36.1	0.01	5.2	**0.00**	142.6	**0.00**	0.4	**0.00**	3.8	**0.00**	3.4
36_92	2	**2.00**	36.4	**2.00**	405.6	2.41	39.7	**2.00**	623.6	**2.00**	2589.4	**2.00**	3071.6	**2.00**	9.4
4_72	0	**0.00**	0.4	**0.00**	10.6	**0.00**	1.3	**0.00**	19.9	**0.00**	3.8	**0.00**	0.4	**0.00**	7.9
41_66	0	**0.00**	0.1	**0.00**	0.7	**0.00**	1.1	**0.00**	1.0	**0.00**	0.2	**0.00**	0.01	**0.00**	0.0
6_76	6	**6.00**	31.7	**6.00**	621.2	**6.00**	37.5	**6.00**	915.2	**6.00**	4180.2	**6.00**	3687.8	**6.00**	7.3

Tableau 5.2 : Résultats de l'AG^{NCPX} vs $ILPGA^{NCPX}$, AG^{IBX} vs $ILPGA^{IBX}$, ACS-IH, AG-IH, VLNS et VNS/ILP sur les instances l'ET2

Le Tableau 5.3 présente les résultats obtenus par les différents algorithmes pour les instances de l'ET3 de manière similaire au Tableau 5.2. Il est important de préciser que Prandtstetter et Raidl [2007] n'ont pas utilisé leur algorithme pour résoudre cet ensemble d'instances. Les résultats présentés dans la première partie du Tableau 5.3 montrent que le croisement hybride *A* utilisé dans l'algorithme $ILPGA^{NCPX}$ améliore la qualité des solutions en moyenne de 26,39 % par rapport à l'AG^{NCPX}. Il permet même d'obtenir la meilleure solution connue à chacune des exécutions pour 16 des 30 problèmes, soit 12 de plus que l'AG^{NCPX}. Ces résultats sont indiqués en gras dans le tableau. On note cependant que l'amélioration de la qualité n'est pas fonction de la taille de l'instance et reste relativement stable pour les différentes tailles d'instances avec une amélioration moyenne de 28,55 % pour les instances de 200 voitures, 25,36 % pour les instances de 300 voitures et 25,27 % pour les instances contenant 400 voitures. En observant maintenant les temps d'exécution, on remarque que l'approche hybride nécessite beaucoup plus de temps que l'AG^{NCPX} avec, en moyenne, 25 minutes pour l'$ILPGA^{NCPX}$ contre 125 secondes pour l'AG^{NCPX} (en ajustant les temps d'exécution en fonction des différents ordinateurs utilisés).

		AG^{NCPX}		$ILPGA^{NCPX}$		AG^{IBX}		$ILPGA^{IBX}$		*ACS-IH*		*GA-IH*		*VLNS*
Instance	**Meilleure Solution Connue**	**Moyenne**	**Temps (s)**	**Moyenne**	**Temps (s)**	**Moyenne**	**Temps (s)**	**Moyenne**	**Temps (s)**	**Moyenne**	**Temps (s)**	**Moyenne**	**Temps (s)**	**Moyenne**
200_01	0	1.23	67.73	**0.00**	1147.00	3.13	77.28	0.90	1992.8	0.00	788.8	0.00	526.2	0.1
200_02	2	2.94	69.73	2.20	1287.00	3.93	76.05	**2.00**	2273.3	2.80	2942.2	2.60	3021.0	2.4
200_03	3	7.41	79.44	5.30	1841.40	11.47	81.10	**4.60**	2240.0	6.00	3086.2	5.00	3215.2	5.8
200_04	7	7.39	77.68	**7.00**	1343.70	7.47	80.43	7.10	2301.4	7.00	2853.2	7.00	2875.2	7.2
200_05	6	6.69	71.40	**6.00**	1091.70	7.71	74.71	6.10	1842.6	6.00	2785.2	6.00	3052.2	6.0
200_06	6	**6.00**	71.55	**6.00**	1773.60	**6.00**	75.92	**6.00**	2031.7	6.00	3360.0	6.00	3508.0	6.0
200_07	0	0.15	21.92	**0.00**	413.90	3.44	76.35	**0.00**	1583.9	0.00	199.8	0.00	192.8	0.0
200_08	8	**8.00**	62.29	**8.00**	1604.80	**8.00**	70.32	**8.00**	1669.0	8.00	3609.2	8.00	3540.8	8.0
200_09	10	10.53	75.41	**10.00**	1567.80	11.40	78.25	**10.00**	2068.3	10.00	3167.6	10.00	3264.0	10.0
200_10	19	21.40	67.40	**19.00**	1422.30	20.72	69.08	19.10	2040.4	19.60	3205.0	19.60	3190.2	19.0
300_01	0	2.79	103.98	0.80	1325.60	5.55	115.81	3.00	2240.9	1.00	3125.0	1.00	3216.8	1.2
300_02	12	12.02	112.73	**12.00**	1461.10	14.78	116.52	12.10	2250.8	12.40	3153.0	12.00	3335.6	12.0
300_03	13	13.11	120.63	**13.00**	1554.80	15.92	123.90	**13.00**	2349.6	13.20	3398.2	13.00	3621.2	13.0
300_04	7	7.71	115.98	7.50	1620.50	10.98	117.49	7.80	2659.2	8.20	3459.6	7.80	3380.6	8.8
300_05	27	42.83	115.90	33.00	2601.90	39.39	118.54	**31.90**	2855.7	33.00	3151.8	32.40	3343.0	33.1
300_06	2	5.30	106.61	3.60	1597.70	8.24	115.14	4.70	2855.2	3.60	3187.4	2.80	3505.0	3.1
300_07	0	0.08	23.17	**0.00**	426.40	3.78	116.77	**0.00**	2217.4	0.00	434.2	0.00	45.6	0.0
300_08	8	**8.00**	105.03	**8.00**	1680.10	9.11	113.27	**8.00**	2298.7	8.00	3497.2	8.00	3618.0	8.0
300_09	7	7.36	105.31	7.20	1649.70	9.40	108.79	7.30	2433.5	7.60	3310.4	7.00	3375.0	8.0
300_10	21	28.48	100.37	22.4	2338.90	32.23	106.13	22.5	2926.1	23.40	3265.0	22.2	3540.6	21.4
400_01	1	1.81	137.89	1.90	1782.00	2.98	151.98	2.20	3313.6	1.60	3383.6	1.00	3713.2	1.6
400_02	15	19.31	147.51	18.30	2081.10	23.41	148.10	18.70	2502.6	17.40	3531.4	17.00	3798.0	16.3
400_03	9	10.79	128.73	9.90	2338.70	12.21	143.06	10.40	3146.2	9.60	3716.2	9.80	4232.2	12.0
400_04	19	19.12	164.57	**19.00**	2354.90	20.37	161.75	**19.00**	3287.1	19.00	3790.4	19.00	4293.4	20.1
400_05	0	**0.00**	2.45	**0.00**	19.80	5.06	139.81	**0.00**	2860.3	0.00	149.6	0.00	4.2	0.0
400_06	0	0.16	38.69	**0.00**	604.80	4.44	146.06	**0.00**	2461.0	0.00	416.8	0.00	64.4	0.0
400_07	4	4.72	130.98	4.70	1479.50	5.59	141.81	4.90	2785.8	4.20	3405.4	4.00	3592.2	4.6
400_08	4	4.73	131.45	4.20	1203.30	7.22	144.15	5.80	2789.0	4.00	3164.0	4.20	3529.6	4.1
400_09	5	10.58	149.18	7.20	2136.50	17.38	155.08	8.70	3312.4	7.20	3421.0	6.00	3870.0	8.3
400_10	0	0.71	102.27	**0.00**	1890.90	4.79	147.75	0.20	2613.2	0.00	98.0	0.00	248.0	0.0

Tableau 5.3 : Résultats de l'AG^{NCPX} vs $ILPGA^{NCPX}$, AG^{IBX} vs $ILPGA^{IBX}$, ACS-IH, AG-IH et VLNS sur les instances de L'ET3

La Figure 5.4 montre le pourcentage de fois où chaque phase trouve la meilleure solution. On remarque, à l'aide de cette figure, que la Phase 2 (phase de diversification) de l'approche hybride trouve rarement la meilleure solution. En effet, sur l'ensemble des problèmes la meilleure solution est trouvée dans la Phase 2 (phase de diversification) de l'algorithme dans moins de 10% des cas. À l'opposé, on note que pour le groupe d'instances contenant 200 voitures la meilleure solution est trouvée dans 56% des cas par la Phase 1 (phase mixte) contre 36% pour la Phase 3 (phase d'intensification) où seul le croisement hybride est utilisé. Pour les groupes de 300 et 400 voitures la meilleure solution est trouvée généralement par la Phase 3 dans respectivement 54 et 56 % des cas contre 42 et 44% pour la Phase 1.

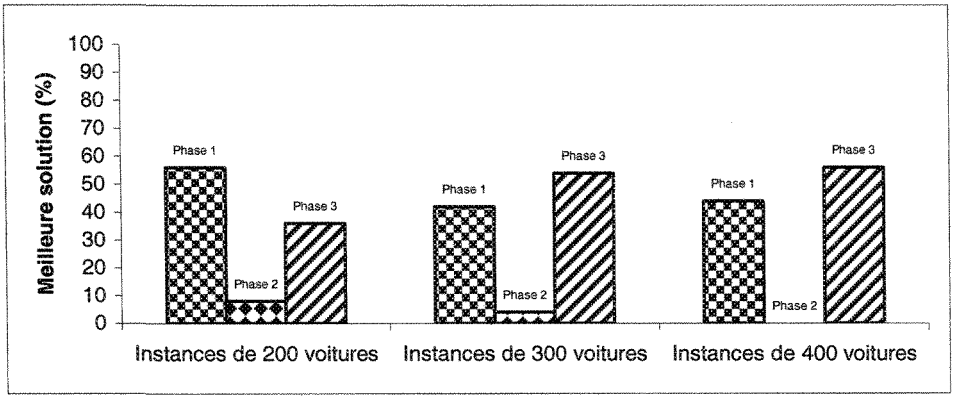

Figure 5.4: Pourcentage de fois où la meilleure solution est trouvée durant la Phase 1, Phase 2 et la Phase 3 de l'ILPGA^{NCPX}

À titre de comparaison, la Figure 5.5 indique, de manière similaire à la Figure 5.4, le pourcentage de fois où chaque étape de l'AG^{NCPX} trouve la meilleure solution. On note ainsi que dans 76% des cas la meilleure solution est trouvée avant la 300^{ième} génération, dans 23% des cas entre la 301^{ième} et la 650^{ième} génération et seulement dans 1% des cas

après la 650^{ième} génération. Il est important de mentionner que ces statistiques ont été utilisées pour définir les différentes phases des algorithmes hybrides.

En comparant ces résultats à ceux de la Figure 5.4, on remarque que la combinaison AG/PLNE permet à l'algorithme hybride de se sortir du piège de l'optimum local. Ainsi, l'*ILPGA*^{NCPX} continue à améliorer la qualité des solutions trouvées après la 301^{ième} génération alors que l'AG ^{NCPX} a, en général, stagné après cette génération. La contribution du PLNE sur la qualité finale des solutions de l'*ILPGA*^{NCPX} est importante puisque c'est en général dans les phases où il est utilisé que la meilleure solution est trouvée.

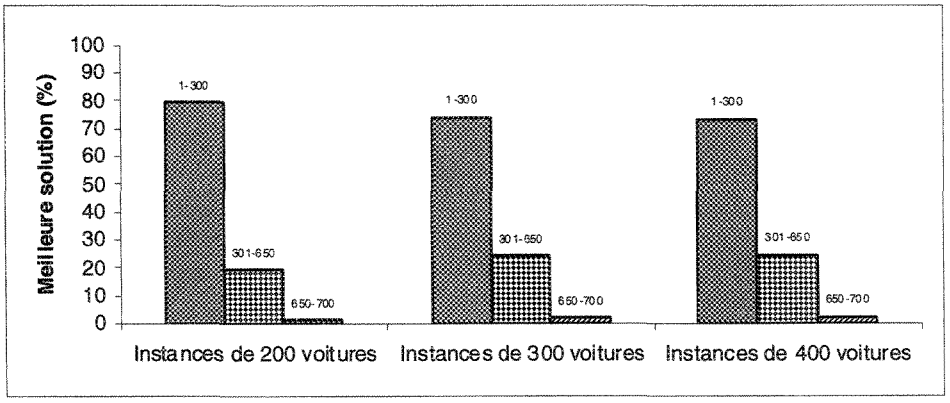

Figure 5.5: Pourcentage de fois où chaque étape de l'AG^{NCPX} trouve la meilleure solution

La seconde partie du Tableau 5.2 présente les résultats comparatifs entre l'AG^{IBX} et l'*ILPGA*^{IBX} sur l'ET3. Lorsque l'on compare la performance de l'approche hybride par rapport au l'*AG*^{IBX}, on remarque que la combinaison de l'AG et du PLNE permet un gain moyen de 37 % en terme de qualité de solutions sur l'ensemble des instances. Ainsi, l'*ILPGA*^{IBX} obtient la meilleure solution connue à chaque exécution pour 11 des 30 problèmes contre seulement deux pour l'AG^{IBX}. Par ailleurs, on note, en observant les résultats plus attentivement, que contrairement à l'*ILPGA*^{NCPX} les gains moyens obtenus par

l'*ILPGA*^{IBX} par rapport à l'A*G*^{IBX} augmentent en fonction de la taille de l'instance. En effet, le gain moyen passe respectivement de 33 % pour le groupe d'instances avec 200 voitures à 34 % pour les instances à 300 voitures et à plus de 44 % pour les instances avec 400 voitures. Cette situation s'explique sans doute en partie par les performances de l'A*G*^{IBX} qui sont inférieures à celles de l'A*G*^{NCPX}, particulièrement pour les instances de grande taille. Lorsque l'on observe maintenant les temps d'exécution, on constate que l'approche hybride requiert un temps d'exécution nettement plus important que celui de l'A*G*^{IBX}.

Lorsque l'on veut maintenant savoir à quelle étape de l'*ILPGA*^{IBX} la meilleure solution est découverte, on remarque à l'aide de la Figure 5.6 que les résultats sont biens différents de ceux obtenus avec l'*ILPGA*^{NCPX}. En effet, pour les trois groupes d'instances on observe que la meilleure solution est essentiellement trouvée lors de la phase d'intensification de la recherche c'est-à-dire la Phase 3 de l'algorithme, en moyenne dans 82% des cas, contre seulement 6% et 12% des cas respectivement pour la Phase 1 et 2.

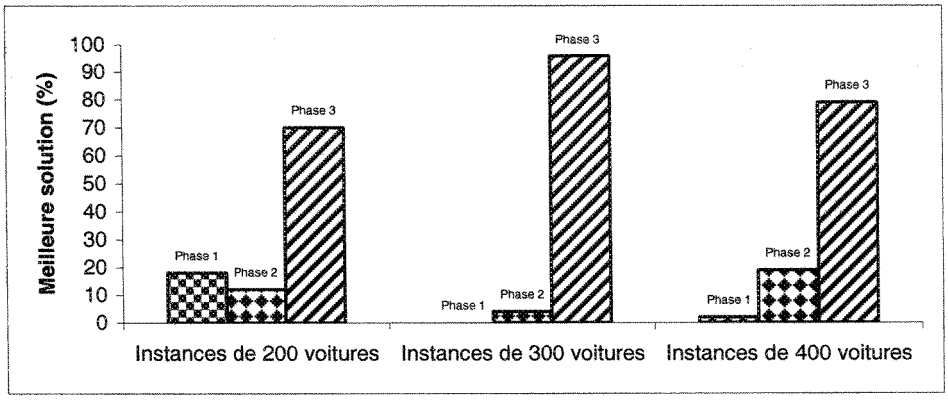

Figure 5.6: Pourcentage de fois où la meilleure solution est trouvée durant la Phase 1, Phase 2 et la Phase 3 de l'ILPGA^{IBX}

La Figure 5.7 présente, en utilisant les mêmes phases que l'ILPGA^{IBX}, le pourcentage de fois où chaque phase trouve la meilleure solution lors de l'exécution de l'*AG*^{IBX}. On note ainsi que l'*AG*^{IBX} converge vers la meilleure solution dans 49% des cas avant la 300^{ième} génération, dans 44% des cas entre la 301^{ième} et la 650^{ième} génération et seulement dans 7% des cas après la 650^{ième} génération. Si on compare ces résultats avec ceux de la Figure 5.6, on peut en conclure, une fois de plus, que la version hybride permet d'éviter une convergence prématurée de l'algorithme et ainsi de sortir du piège de l'optimum local.

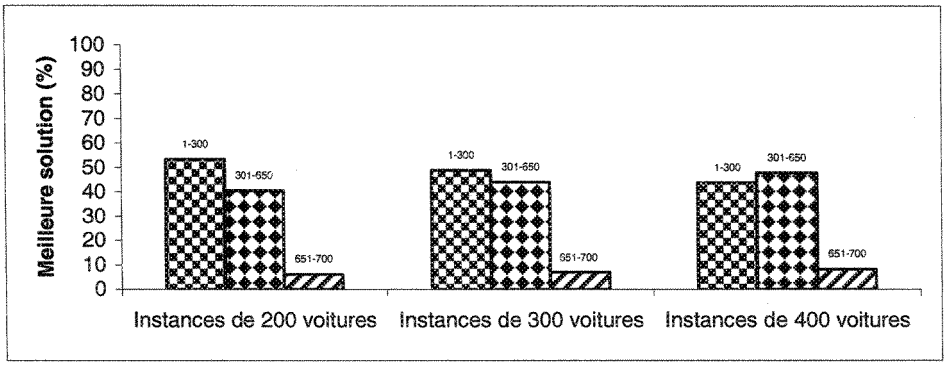

Figure 5.7: Pourcentage de fois où chaque étape de l'AG^{IBX} trouve la meilleure solution

Globalement, on observe que l'*ILPGA*^{NCPX} obtient de meilleurs résultats l'*ILPGA*^{IBX} pour 17 des 30 instances alors que l'*ILPGA*^{IBX} est meilleur pour seulement 3 instances. Ces résultats montrent la supériorité du croisement hybride *NCPX* par rapport au croisement hybride *IBX*.

Lorsqu'on compare maintenant les performances de l'*ILPGA*^{NCPX} et de l'*ILPGA*^{IBX} à celles de l'*ACS-IH*, on remarque que l'*ILPGA*^{NCPX} surclasse l'*ACS-IH* sur 9 instances, obtient de moins bons résultats pour 5 instances tout en obtenant des performances identiques sur les instances restantes. De plus, le temps moyen d'exécution de l'*ACS-IH* (2702 secondes) est plus important que celui de l'*ILPGA*^{NCPX} (1521 secondes). On en

conclut donc qu'en accordant le même temps moyen d'exécution aux deux algorithmes la différence serait encore plus importante. De son côté, l'*ILPGA*^{IBX} obtient de meilleurs résultats que l'*ACS-IH* pour 9 instances, est moins bon sur 12 instances et obtient une performance identique sur les instances restantes. Dans ce cas, les temps d'exécution des deux algorithmes sont comparables. En effet, le temps moyen d'exécution de l'*ACS-IH* est de 2702 secondes contre 2473 secondes pour l'*ILPGA*^{IBX}. Toutefois, on note que l'*ACS-IH* effectue plus d'évaluations que l'*ILPGA*^{NCPX} et l'*ILPGA*^{IBX}.

Si on compare maintenant l'*ILPGA*^{NCPX} et l'*ILPGA*^{IBX} à l'*AG-IH*, on remarque cette fois-ci que l'*ILPGA*^{NCPX} obtient de meilleurs résultats pour 4 instances et est moins bon sur 10 instances. Toutefois, le temps moyen d'exécution de l'*AG-IH* (2825 secondes) est beaucoup plus important que celui de l'*ILPGA*^{NCPX} (1521 secondes). De son côté, l'*ILPGA*^{IBX} surclasse l'*AG-IH* sur 5 instances et est moins bon sur 14 instances. Le temps moyen d'exécution l'*ILPGA*^{IBX} (2473 secondes) est légèrement inférieur à celui de l'*AG-IH* (2825 secondes). Ces résultats ne sont pas réellement surprenants, en effet l'*AG-IH* utilise aussi l'opérateur de croisement *NCPX* proposé au chapitre précédent et effectue plus de générations que les deux approches proposées dans ce chapitre. De plus, nous allons montrer ultérieurement qu'une version de notre algorithme intégrant les deux opérateurs de croisement hybrides permet de surclasser clairement l'*AG-IH*.

Si on compare les performances de l'*ILPGA*^{NCPX} et de l'*ILPGA*^{IBX} avec celles du *VLNS* qui sont présentées dans la dernière colonne du Tableau 5.2, on en conclut que les deux approches hybrides obtiennent de meilleurs résultats. En effet, l'*ILPGA*^{NCPX} surclasse le *VLNS* sur 24 des 30 instances tandis que l'*ILPGA*^{IBX} obtient de résultats au moins

identiques à ceux du VLNS pour 17 des 30 instances. Ainsi, on peut en conclure que les heuristiques à base de populations, comme les AG, contribuent grandement sur les performances des approches hybrides proposées dans cette thèse. En effet, contrairement à l'*ILPGA*^{NCPX} et à l'*ILPGA*^{IBX} le *VLNS* est une heuristique à solution unique qui appelle de manière itérative un PLNE.

Au Tableau 5.4, nous analysons l'influence de la valeur initiale de k_{mov} sur la qualité de la solution finale et sur le temps d'exécution de l'*ILPGA*^{NCPX}. Ces résultats ont été obtenus sur le problème 300_05 en accordant 3 secondes à CPLEX dans la première phase et 10 secondes dans la 3^{ième}. Les résultats présentés dans ce tableau correspondent aux résultats moyens obtenus après 5 exécutions consécutives de l'*ILPGA*^{NCPX}. À partir de ce tableau, on note que la valeur initiale de k_{mov} peut varier entre 5% et 15% de la taille du problème (ce qui correspond à entre 15 et 45 positions pour ce problème) sans que la qualité de la solution finale soit dégradée. Entre cet intervalle, les temps d'exécution de l'algorithme sont équivalents. Dans leur article, Estellon *et al.* [2007] utilisent un k_{mov} compris entre 10 et 50 positions pour tous les problèmes. Cependant, nous pensons qu'il est plus judicieux de commencer avec une plus petite valeur initiale de k_{mov} et de l'augmenter progressivement. Pour des problèmes plus difficiles comme le problème 300_05, cette stratégie permet d'éviter que la résolution du PLNE par CPLEX échoue dans le temps imparti.

*% * nc*	**moyenne**	**Temps (s)**	k_{mov} **maximum**
3%	33.6	2049.8	70
5%	33.0	2702.4	70
10%	33.0	2846.8	71
15%	33.0	2586.4	71
20%	33.2	2456.6	64.6

Tableau 5.4: Résultats obtenus en faisant varier la valeur initiale de k_{mov} pour le problème 300_05

Les croisements hybrides présentés dans ce chapitre ont un fonctionnement différent l'un de l'autre sans pour autant être complètement opposées. Ainsi, on peut donc facilement les combiner dans un même algorithme en supposant que leur utilisation simultanée peut amener un certain gain aussi bien en termes de diversification de la recherche que de qualité finale des solutions. Dans l'algorithme résultant (*ILPGA*$^{NCPX/IBX}$), les différents opérateurs de croisement sont appliqués de la manière suivante en fonction de la phase de création de la population enfant :

- Phase 1: utilisation aléatoire d'un des deux croisements hybrides (croisement hybride *A ou B)* pour créer 10 % de la population Enfant et à parts égales du croisement *NCPX* et *IBX* pour le reste de la population;
- Phase 2: utilisation du croisement NCPX dans 65 % des cas et du croisement IBX pour le reste;
- Phase 3 : utilisation aléatoire d'un des deux croisements hybrides (*A ou B).*

La supériorité du croisement *NCPX* par rapport au croisement *IBX* observée au chapitre précédent explique les pourcentages utilisés pour les deux opérateurs à la Phase 2. Les résultats de l'*ILPGA*$^{NCPX/IBX}$ sont présentés dans la partie gauche du Tableau 5.5.

Lorsque l'on compare les résultats de l'ILPGA$^{NCPX/IBX}$ à ceux de l'*ILPGA*^{NCPX} et de l'*ILPGA*^{IBX}, on note que l'approche mixte fournit globalement de meilleurs résultats que les deux autres algorithmes. En effet, l'ILPGA$^{NCPX/IBX}$ obtient des résultats identiques ou meilleurs à ceux de l'*ILPGA*^{NCPX} et de l'*ILPGA*^{IBX} pour 29 des 30 instances. De plus, si on compare maintenant les résultats de l'ILPGA$^{NCPX/IBX}$ à ceux de l'*ACS-IH* et de l'A*G-IH,* on

note que l'ILPGA$^{NCPX/IBX}$ obtient de meilleurs résultats pour 29 des 30 instances par rapport à l'ACS-IH et pour 28 des 30 instances par rapport à l'AG-IH. On peut donc conclure que l'intégration des différents opérateurs de croisement dans l'ILPGA$^{NCPX/IBX}$ permet à l'algorithme de mieux diversifier sa recherche tout en conservant un bon équilibre entre intensification et diversification.

	ILPGA$^{NCPX/IBX}$		*AG$^{NCPX/IBX}$* + LS		*VLNS*
Instance	**Moyenne**	**Temps (s)**	**Moyenne**	**Temps (s)**	**Moyenne**
200_01	**0.00**	2155.1	0.14	47.50	0.1
200_02	**2.00**	1620.1	**2.00**	89.99	2.4
200_03*	**4.10**	1456.5	5.13	101.21	5.8
200_04	**7.00**	2344.9	**7.00**	98.01	7.2
200_05	**6.00**	1637.0	**6.00**	90.39	6.0
200_06	**6.00**	2031.7	**6.00**	93.71	6.0
200_07	**0.00**	1583.9	**0.00**	10.11	0.0
200_08	**8.00**	1669.0	**8.00**	80.44	8.0
200_09	**10.00**	1771.0	**10.00**	97.26	10.0
200_10	**19.00**	1739.7	**19.00**	83.91	19.0
300_01	0.30	1606.9	**0.08**	135.29	1.2
300_02	**12.00**	1882.9	**12.00**	145.60	12.0
300_03	**13.00**	2222.6	**13.00**	157.54	13.0
300_04*	**7.00**	1357.0	7.26	140.05	8.8
300_05*	**30.50**	2045.6	32.08	147.88	33.1
300_06*	2.60	1454.9	**2.55**	137.86	3.1
300_07	**0.00**	879.2	**0.00**	11.60	0.0
300_08	**8.00**	2064.6	**8.00**	148.46	8.0
300_09*	7.30	2197.7	**7.03**	139.73	8.0
300_10	**21.20**	2725.7	21.30	134.04	21.4
400_01*	**1.10**	2594.9	1.29	171.78	1.6
400_02*	**16.70**	2101.4	16.75	175.24	16.3
400_03*	**9.60**	2761.2	10.07	160.11	12.0
400_04	**19.00**	3028.2	**19.00**	211.72	20.1
400_05	**0.00**	64.6	**0.00**	5.26	0.0
400_06	**0.00**	899.1	**0.00**	7.57	0.0
400_07	**4.00**	2051.1	4.09	163.79	4.6
400_08	4.30	2185.5	**4.00**	165.66	4.1
400_09*	**5.80**	1598.8	**6.69**	185.74	8.3
400_10	**0.00**	2487.9	**0.00**	33.15	0.0

Tableau 5.5: Résultats de l'ILPGA$^{NCPX/IBX}$, de l'AG$^{NCPX/IBX}$ + LS et du VLNS sur les instances de l'ET3

Le Tableau 5.5 permet également de comparer la performance de l'*ILPGA$^{NCPX/IBX}$* aux résultats de l'*AG$^{NCPX/IBX}$* + LS. Dans le chapitre précédent, il a été montré que l'approche

mixte $AG^{NCPX/IBX}$ + LS s'est avérée l'algorithme le plus performant pour résoudre l'ensemble des problèmes de la librairie CSPLib. Au Tableau 5.5, on note que les performances des deux algorithmes sont très proches l'une de l'autre. Toutefois, l'$ILPGA^{NCPX/IBX}$ obtient des résultats identiques ou meilleurs pour 26 des 30 instances. Sa performance est particulièrement intéressante sur les instances 200_3 et 300_05 où, en moyenne, il existe une différence de plus d'un conflit entre les solutions des deux algorithmes. Les meilleurs résultats sont montrés en caractères gras au Tableau 5.5.

En plus des résultats présentés au Tableau 5.5, nous avons aussi effectué un test statistique pour déterminer si l'$ILPGA^{NCPX/IBX}$ est réellement plus performant que l'$AG^{NCPX/IBX}$ +LS. Comme les deux algorithmes obtiennent souvent des résultats identiques et à cause de la faible variabilité des résultats, il est difficile d'utiliser un test statistique pour différencier les deux approches de résolution. Cependant, les travaux de Conover et Iman [1981] expliquent comment effectuer un test paramétrique *t* (*t*-test) sur la moyenne des rangs dans les cas où l'on a plus de 80% de résultats identiques. Ainsi, uniquement les 9 instances identifiées par un astérisque (*) au Tableau 5.5 ont été considérées dans le test paramétrique présenté au Tableau 5.6. Ces résultats indiquent, avec un niveau de confiance de 99%, que l'$ILPGA^{NCPX/IBX}$ obtient de meilleurs résultats que l'$AG^{NCPX/IBX}$ + LS.

	$ILPGA^{NCPX/IBX}$	$AG^{NCPX/IBX}$ + LS
Moyenne des rangs	56.94	41.14
Variance	698.64	834.05
Nombres de cas	900	90
Valeur *t*	5.357	
Probabilité 2-tail	0,000	

Tableau 5.6: t-test pour la comparaison de la moyenne des rangs entre $ILPGA^{NCPX/IBX}$ et l'$AG^{NCPX/IBX}$ +LS pour les 9 instances considérées (deux échantillons indépendants) avec un niveau de signification de 1%.

Il existe toutefois un écart important entre les temps moyens d'exécution des deux algorithmes. En effet, l'$AG^{NCPX/IBX}$ utilisant une recherche locale prend en moyenne 149 secondes par problème comparativement à 1874 secondes pour l'ILPGA$^{NCPX/IBX}$. Toutefois, un temps d'exécution d'environ 30 minutes ne constitue pas un réel handicap dans l'industrie automobile car les opérations liées à l'ordonnancement de voitures sont effectuées une journée avant le début de la production. Ainsi, si l'utilisation d'une méthode exacte dans un algorithme hybride permet d'améliorer la qualité des solutions obtenues, ce type d'approche sera probablement implémenté en contexte industriel. Il serait donc intéressant de tester les approches proposées dans ce chapitre sur des problèmes de taille industrielle afin de voir si la différence entre les approches hybrides et heuristiques augmente ou non.

Lorsqu'on compare maintenant l'*ILPGA*$^{NCPX/IBX}$ au VLNS, on note que l'*ILPGA*$^{NCPX/IBX}$ obtient généralement de meilleurs résultats que le *VLNS*. En effet, l'*ILPGA*$^{NCPX/IBX}$ obtient des résultats identiques ou meilleurs à ceux du VLNS pour 28 des 30 instances de l'ET3.

5.5 Conclusion

Dans ce chapitre, nous avons présenté deux techniques hybrides permettant de résoudre efficacement le problème théorique d'ordonnancement de voitures. Ces différentes méthodes sont basées sur des métaheuristiques à base de populations, les algorithmes génétiques, et sur un programme linéaire en nombres entiers. En particulier, les approches proposées reposent sur deux nouveaux croisements hybrides. La nature différente des

opérateurs de croisement proposés nous a permis de les combiner et de montrer que l'algorithme résultant permettait d'obtenir des résultats compétitifs, en termes de qualité de solutions, par rapport aux résultats obtenus aux résultats de l'AG-NCPX avec recherche locale du chapitre précédent. Les travaux réalisés dans ce chapitre montrent donc l'intérêt de ce type de méthode lorsque les connaissances du domaine sont bien prises en compte afin de développer des méthodes adaptées qui permettent d'exploiter les forces de chacune des approches hybridées.

Toutefois, les expérimentations numériques effectuées ont mis en évidence l'écart considérable entre le temps d'exécution des approches hybrides et ceux des AG proposés au Chapitre 4. Ces différences, même si elles doivent être nuancées du fait de différentes configurations expérimentales, laissent entrevoir des améliorations possibles au niveau de l'implémentation des méthodes hybrides pour les rendre encore plus compétitives. Ces améliorations peuvent, par exemple, passer par le développement de versions parallèles des algorithmes proposés ou encore par l'utilisation de technologies permettant de développer des puissances de calculs importantes comme les grilles de calculs ou encore le calcul pair à pair.

D'autres perspectives peuvent aussi être envisagées, comme le remplacement de l'opérateur de mutation par une méthode exacte en s'inspirant du fonctionnement des algorithmes mimétiques. De plus, il serait intéressant d'évaluer les coopérations de type collaboratives ainsi qu'une éventuelle combinaison avec les approches proposées dans cette partie. Finalement, afin de valider les différents mécanismes présentés dans cette partie, il serait intéressant de les tester sur d'autres types de problèmes.