

ALGORITHMES GÉNÉTIQUES POUR LE CARP

Philippe LACOMME, Christian PRINS, Wahiba RAMDANE-CHERIF

Université de Technologie de Troyes, Laboratoire d'Optimisation des Systèmes Industriels (LOSI)
12, Rue Marie Curie, BP 2060, 10010 Troyes Cedex (France)
Mél : {Lacomme, Prins, Ramdane}@univ-troyes.fr

RÉSUMÉ : *Le CARP (Capacitated Arc Routing Problem) est un problème NP-difficile, utile pour modéliser par exemple le ramassage des ordures ménagères ou le déneigement des routes. Les méthodes exactes de résolution ne résolvent que de petites instances et les problèmes de grande taille sont traitables uniquement par des heuristiques ou des métaheuristiques. Nous présentons les premiers algorithmes génétiques de la littérature pour résoudre le CARP. Le modèle de données étendu permet de prendre en compte efficacement plusieurs extensions telles que les rues à sens unique ou à double sens et les interdictions de tourner. Notre meilleur algorithme génétique, de type hybride, a été testé sur des instances de la littérature et se montre compétitif avec les métaheuristiques déjà publiées.*

MOTS-CLÉS : *Tournées sur arcs, CARP, algorithme génétique*

1. INTRODUCTION

Les problèmes de tournées de véhicules les plus étudiés, comme le VRP (*Vehicle Routing Problem*) consistent à servir des demandes placées sur les nœuds d'un réseau. Si les demandes correspondent à des arcs, l'équivalent du VRP est un problème de tournées sur arcs ou CARP (*Capacitated Arc Routing Problem*). Le CARP de base est en général défini dans la littérature sur un graphe non orienté. Les nœuds comprennent un dépôt où est basée une flotte de véhicules identiques de capacité connue. Chaque arête a un coût de traversée et une demande. Les arêtes de demandes non nulles doivent obligatoirement être traversées pour être traitées. Le CARP consiste à déterminer un ensemble de tournées de coût total minimal (ce coût ne prend pas en compte le coût de possession des véhicules), tel que :

- chaque tournée, assurée par un véhicule, commence et finit au dépôt ;
- chaque arête peut être traversée plusieurs fois, par la même tournée ou des tournées différentes, mais elle doit être traitée par une seule tournée et en une seule traversée ;
- la somme des demandes traitées par une tournée ne doit pas excéder la capacité du véhicule.

Beaucoup d'applications concernent les réseaux routiers. Les demandes sont alors des quantités à collecter (déchets ménagers) ou à amener (sablage de routes en cas de verglas). Les coûts sont souvent des distances ou temps de parcours. Le cas non orienté concerne des rues traitables en un seul passage et dans n'importe quel sens. On peut aussi considérer un CARP orienté, où un arc désigne une rue ou un côté de rue avec un sens de traitement imposé. Le CARP orienté ou non est NP-difficile, même dans le cas particulier du postier rural (*Rural Postman Problem* ou RPP) qui désigne la version

avec un seul véhicule de capacité infinie. Le postier chinois (*Chinese Postman Problem* ou CPP) est le cas particulier du RPP où toutes les arêtes sont à traiter. Il est NP-difficile pour les graphes mixtes, mais polynomial si G ne comprend que des arcs ou bien des arêtes.

Des formulations du CARP sous forme de programmes linéaires en nombres entiers et des bornes inférieures ont été étudiées par Golden et Wong (1981), puis par Benavent *et al.* (1992). Belenguer et Benavent (1997) résolvent une formulation relaxée avec une méthode de coupes, obtenant ainsi une excellente borne inférieure. Les méthodes exactes de type branch-and-bound sont actuellement peu performantes : elles traitent des cas jusqu'à 30 arcs (Hirabayashi *et al.*, 1992).

Actuellement, les seules méthodes utilisables sur de grandes instances sont des heuristiques gloutonnes ou des méthodes itératives d'amélioration. Les heuristiques gloutonnes sont souvent ingénieuses et efficaces, citons Path-Scanning (Golden et Wong, 1981), Construct-Strike (version améliorée de Pearn, 1989), Augment-Insert (Pearn, 1991), Augment-Merge (Golden *et al.*, 1983) et la méthode de type Construct first/Cluster second d'Ulusoy (1985). Elles présentent le double intérêt de donner une bonne solution dans des temps de calcul raisonnables et de servir de point de départ pour des métaheuristiques. Eglese (1994) a ainsi utilisé le recuit simulé pour un problème de sablage de route. La meilleure métaheuristique est actuellement une méthode tabou sophistiquée de Hertz *et al.* (2000).

Nous adoptons par la suite le vocabulaire relatif à la collecte des ordures ménagères, afin que les exemples soient plus représentatifs et que l'intérêt de l'étude de tels problèmes apparaisse clairement. Nous présentons notre extension du CARP (section 2) et des structures de

données (section 3). La section 4 propose une famille d'algorithmes génétiques, le meilleur étant évalué sur des jeux-tests de la littérature dans la section 5.

2. CARP ÉTENDU

Dans leur version actuelle, nos algorithmes traitent déjà un CARP étendu par rapport à la littérature. Il est basé sur un *graphe mixte* $G = (V, U, E)$ avec un ensemble d'arcs U et un ensemble d'arêtes E . Les n nœuds de V modélisent des carrefours ou des localités, ils incluent un dépôt s où sont basés des camions de capacité Q . Les camions sont supposés être en nombre suffisant. Un arc (i, j) de U représente une rue à sens unique ou un côté de rue à collecter de i vers j . Une arête de E correspond à une rue à double sens, collectable dans n'importe quel sens. Ce type de rue se trouve fréquemment dans les habitats pavillonnaires à faible circulation. Nous appelons *lien* un arc ou une arête. Chacun des m liens (i, j) a un coût de traversée $c_{ij} \geq 0$ et une demande $r_{ij} \geq 0$ (quantité à collecter). Nous appelons *tâches* les t liens à traiter (ceux de demande non nulle).

Un aspect non réaliste du CARP de base est un coût identique pour la collecte d'un lien et sa traversée sans collecter (appelée passage en *haut-le-pied* ou *HLP* par la profession). Nous distinguons donc pour tout lien un *coût de collecte* $w_{ij} \geq 0$ et un *coût de traversée en HLP* $c_{ij} \geq 0$. Enfin, nous gérons des interdictions de tourner et des pénalités à certains carrefours. Un cas simple d'interdiction de tourner est le demi-tour, trop fréquent dans les solutions obtenues par les heuristiques publiées. Quant aux pénalités, elles permettent par exemple de modéliser l'attente supplémentaire quand on veut tourner à gauche à un carrefour avec feux.

3. STRUCTURES DE DONNÉES

3.1 Codage du réseau

Le graphe mixte et les interdictions de tourner ont conduit à un graphe interne complètement orienté $H=(V, A)$, codé par des listes d'arcs. L'ensemble A comprend $nai = |U|+2 \cdot |E|+1$ arcs internes (AI) indexés de 1 à nai . Chaque arc de U est codé par un AI. Chaque arête $[i, j]$ de E donne lieu à deux arcs internes (i, j) et (j, i) reprenant les coûts et la demande de l'arête. Enfin, le dépôt s est codé comme un arc interne artificiel, une boucle (s, s) d'index nai . Un arc interne u est défini par un nœud de départ $b(u)$, un nœud d'arrivée $e(u)$, une demande $r(u)$, un coût de collecte $w(u)$, un coût de traversée en HLP $c(u)$, un pointeur $inv(u)$ vers l'arc inverse (pour les arêtes) et une liste $Succ(u)$ d'AI- vers lesquels on peut tourner. Pour un AI u non collectable, on pose par convention $r(u) = 0$ et $w(u) = c(u)$.

Ce codage du réseau facilite de nombreux traitements. Par exemple, quand deux arcs internes u et v codent la même arête, le champ inv permet de marquer u et v

comme étant collectés dès qu'une tournée collecte l'arête, que ce soit dans le sens de u ou dans celui de v . Quant à la liste d'arcs successeurs $Succ(u)$, elle rend transparentes les interdictions de tourner : s'il n'est pas permis de tourner de u sur v , l'arc v ne sera pas visible pour un algorithme de tournées parvenu en u . $P(u, v)$ représente la pénalité pour tourner de u vers v .

3.2 Codage et calcul du distancier

A cause des interdictions de tourner, le chemin optimal entre deux nœuds i et j dépend des arcs traversés pour parvenir en i et pour partir de j . Il est donc utile d'utiliser des *distances entre arcs*, précalculées dans un *distancier* D , $nai \times nai$. Pour tout couple d'arcs internes (u, v) , $D(u, v)$ donne le coût du plus court chemin HLP de u à v (*non inclus*), tenant compte des interdictions de tourner et des pénalités éventuelles pour tourner d'un arc au suivant. u et v sont non inclus pour faciliter l'insertion ou l'enlèvement d'une tâche dans une tournée. Par exemple, si on insère une tâche z entre deux tâches u et v , la variation de coût vaut $D(u, z) + w(z) + D(z, v)$. Une matrice $Pred$, $nai \times nai$ est également calculée pour stocker de manière compacte un plus court chemin entre tout couple d'arcs, ce qui permet de détailler les chemins lors de l'impression des tournées. $Pred(u, v)$ est le prédécesseur de v sur le chemin optimal de u à v .

Les coûts positifs de traversée des arcs suggèrent d'utiliser l'algorithme de plus court chemin de Dijkstra. Cet algorithme doit être refondu pour les interdictions de tourner, car il utilise en chaque nœud une étiquette insuffisante pour distinguer de quel arc on vient. La figure 1 donne une version *ad hoc* qui calcule dans le graphe H les plus courts chemins partant de l'arc u , c'est à dire la ligne u de D . Elle utilise pour tout arc v un label $L(v)$ désignant le coût des chemins optimaux de u (non inclus) à v (inclus), et un booléen $F(v)$ vrai si et seulement si v a son label définitif. H est supposé fortement connexe, comme tout réseau routier réel.

```

L(u) := 0, L(v) := ∞ pour tout arc v ≠ u.
F(v) := Faux pour tout arc v
//Boucle fixant le label d'un arc par itération
Pour k := 1 à nai
    Chercher l'arc v non fixé de label minimal
    //L(v) ≠ ∞ car H est fortement connexe
    F(v) := Vrai //Le label de v est définitif
    Pour tout arc z ∈ Succ(v) avec L(v) + c(z) < L(z)
        L(z) := L(v) + c(z) + P(v, z)
        Pred(u, z) := v
    FinPour
FinPour
//Chargement de la ligne u de D
Pour v := 1 à nai
    D(u, v) := L(v) - c(v) - P(Pred(u, v), v)
FinPour

```

Figure 1. Algorithme de Dijkstra modifié

Cette version modifiée de l'algorithme de Dijkstra, appliquée à chaque arc de départ possible u , permet de calculer les matrices D et P en $O(m^3)$, ou en $O(m^2 \cdot \log m)$

si on utilise une structure de données appelée *tas* (Cormen *et al.*, 1990). Pour un réseau routier, on a en général $m \approx 4n$, la complexité des deux implémentations chute alors respectivement à $O(n^3)$ et $O(n^2 \cdot \log n)$. Notez que le distancier n'est pas forcément symétrique, à cause des rues en sens unique ou des interdictions de tourner.

3.3 Codage des solutions

Une solution consiste en un ensemble de $NTrips$ tournées. Pour $i = 1, 2, \dots, NTrips$, la tournée $Trip(i)$ est définie par un coût total $Cost(i)$, une charge totale $Load(i) \leq Q$, et une suite $Seq(i)$ de $NTasks(i)$ tâches (numéros d'AI, dans l'ordre de collecte par la tournée). Les plus courts chemins entre tâches ne sont pas stockés, car leurs coûts sont consultables en $O(1)$ dans D . $Cost(i)$ est la somme des coûts de collecte des tâches de $Seq(i)$ et des coûts des chemins intermédiaires. Le coût total $TotCost$ est la somme des coûts des tournées.

```

u := Seq(i,1)
Cost(i) := D(nai,u)+w(u)
Pour k := 2 à NTasks(i)
    v := Seq(i,k)
    Cost(i) := Cost(i)+D(u,v)+w(v)
    u := v
FinPour
Cost(i) := Cost(i)+D(u,nai).

```

Figure 2. Calcul du coût $Cost(i)$ d'une tournée i

La figure 2 montre comment calculer en $O(NTasks(i))$ le coût $Cost(i)$ d'une tournée i . $Seq(i,k)$ désigne la tâche de rang k de la tournée i . La tournée part de l'arc interne d'index nai représentant le dépôt et y revient. La boucle cumule pour chaque tâche v le coût de collecte de v et le coût du chemin optimal depuis la tâche précédente u .

4. ALGORITHMES GENETIQUES POUR LE CARP ETENDU

4.1 Notion d'algorithme génétique

Les *algorithmes génétiques* (GA) proposés par Holland en 1975 ont été depuis utilisés avec un succès croissant en optimisation combinatoire, voir par exemple la synthèse de Reeves et celle en français de Fleurent et Ferland (les deux en 1996). En simplifiant, un GA opère sur une *population* d'individus codés par des chaînes de symboles appelés *chromosomes*. Ces chaînes sont munies d'une évaluation appelée *fitness*, sorte de mesure d'adaptation au milieu.

L'itération de base d'un GA consiste à tirer au sort deux parents selon une distribution favorisant les individus les plus adaptés. Un opérateur de *croisement* combine ensuite les deux chromosomes-parents pour construire un ou deux enfants, pouvant à leur tour être modifiés aléatoirement par un opérateur de *mutation*. Les enfants servent à construire la génération suivante ou remplacent directement des individus de la population. Le processus est répété jusqu'à ce qu'un critère d'arrêt soit vérifié.

Les sections suivantes présentent les caractéristiques essentielles de nos GA : les chromosomes, leur évaluation, les croisements et mutations étudiés. Ces caractéristiques peuvent se décliner en différent GA selon l'implémentation de la population initiale, la gestion des générations, et les critères d'arrêt.

4.2. Type de chromosome et évaluation

Nos chromosomes traduisent directement les solutions et pour une solution réalisable, un chromosome est la concaténation ($Seq(1), Seq(2), \dots, Seq(NTrips)$) des listes de tâches des tournées. Les limites entre tournées successives sont conservées. L'évaluation est simplement le coût total de la solution. Ce coût devant être minimisé dans le cas du CARP, les chromosomes les plus adaptés et donc les plus favorisés pour la reproduction seront ceux de plus faible coût. Le GA initial de Holland utilisait des chromosomes binaires. Comme dans beaucoup de GA publiés pour des problèmes de séquençement, nos chromosomes sont en fait des permutation (des t tâches du réseau).

Le haut de la *figure 3* montre deux parents et leurs chromosomes pour un CARP non orienté avec 8 tâches-arêtes, en trait épais. Les demandes valent toutes 1, et $Q = 3$. Les traits fins sont des plus courts chemins dans le graphe complet, non détaillé ici. Le dépôt s est le point de jonction central. Les flèches indiquent le sens de parcours des tournées. Chaque arête est codée par deux arcs internes. On suppose que les traversées d'arêtes de gauche à droite ou de bas en haut donnent les AI 1 à 8, et que les autres sens donnent les AI 9 à 16. De plus, pour tout AI u entre 1 et 8, on suppose que $inv(u) = u + 8$: par exemple, l'arête la plus haute est codée par les arcs internes 7 et 15. Les tirets dans les chromosomes indiquent les séparations entre tournées.

4.3. Croisements étudiés

Trois croisements opérant sur des permutations sont connus pour les problèmes de séquençement : LOX (*Linear Order Crossover*), OX (*Order Crossover*) et X1 (*croisement à un point de coupure*). Pour deux parents P1 et P2 de longueur t , ils commencent par tirer au sort deux positions p et q avec $p \leq q$. Pour construire l'enfant E1, la portion de P1 entre p et q inclus est copiée dans E1, aux mêmes positions.

Dans LOX, P2 est ensuite balayé de 1 à t et les éléments non déjà présents dans l'enfant remplissent de gauche à droite les positions libres de l'enfant. Dans OX, la seule différence est que le balayage de P2 et le rangement dans l'enfant s'effectuent de façon circulaire, en commençant à l'index $q+1 \pmod{t}$. Pour tous ces croisements, la construction de l'enfant E2 est identique, en permutant les rôles de P1 et P2. Enfin, X1 peut être vu comme un cas particulier de LOX avec $p = 1$.

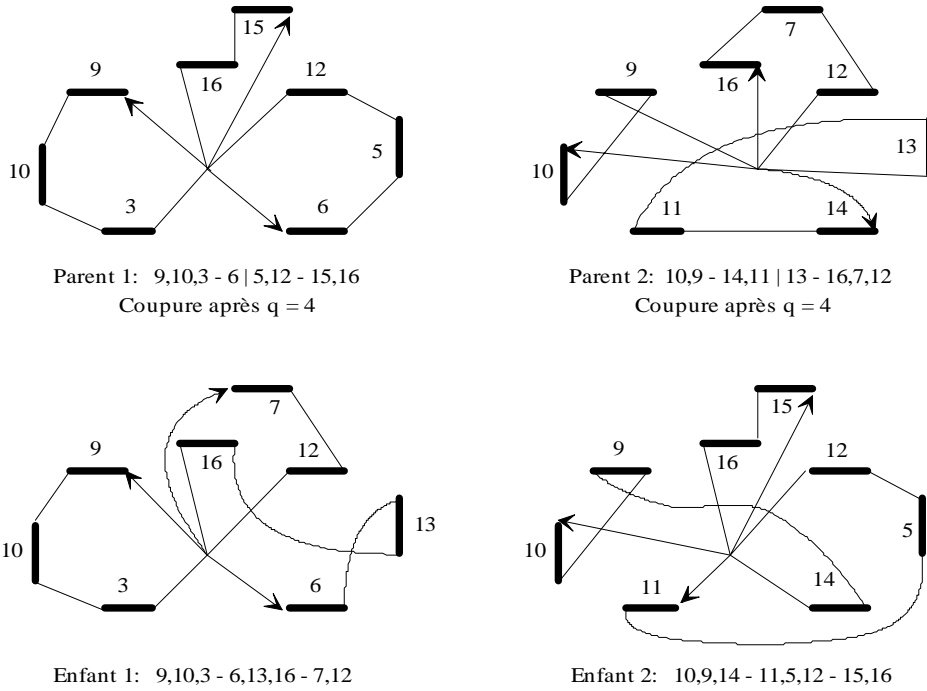


Figure 3. Exemple de parents avec les enfants obtenus avec le croisement X1

Nous adaptons ces croisements au CARP et à notre structure de données. Un chromosome contient bien les t tâches du problème, mais une tâche-arête peut y figurer sous forme d'un *arc interne* u ou de son inverse. En copiant u d'un parent, on vérifie donc que u et $inv(u)$ ne sont pas déjà pris. Un autre problème est le découpage en tournées des enfants. Nous parcourons circulairement chaque enfant, en démarrant une tournée avec la tâche d'indice p . Si la tâche en cours u viole la capacité du camion, la tournée actuelle est terminée et une nouvelle tournée commence avec u . L'évaluation (coût total de la solution) est recalculée durant ce balayage. Tous ces croisements modifiés sont implémentables en $O(t)$. Le bas de la figure 3 donne les fils obtenus pour le croisement X1.

4.4. Mutations étudiées

Traditionnellement, la mutation est appliquée avec un faible taux aux enfants pour empêcher une convergence trop rapide de l'algorithme génétique. Nous avons testé trois types de mutation, nommés *Move*, *Swap*, et *RL*. *Move* tire au sort deux positions p et q du chromosome, avec $1 \leq p \leq t$, $0 \leq q \leq t$, $p \neq q$ et $p \neq q-1$. La tâche u au rang p est déplacée *après* celle de rang q ($q = 0$ signifie une insertion en tête). Puis, pour une tâche-arête, on inverse avec une probabilité de 0.5 le sens de traitement (remplacement de u par $inv(u)$). *Swap* tire au sort deux positions p et q entre 1 et t inclus, permute les tâches situées à ces positions, et inverse chacune d'elles avec une probabilité de 0.5 si ce sont des arêtes. Le chromosome obtenu est balayé pour être redécoupé en tournées et réévalué. Le travail total est en $O(t)$.

RL est une recherche locale consistant à améliorer les enfants avant de les injecter dans la population. Les supporters du GA initial de Holland (qui ne prévoit pas cette technique) affirment que la recherche locale modifie sensiblement la philosophie de la méthode : le croisement paraît réduit à une astuce nécessaire pour échapper aux minima locaux atteints par la recherche locale. On peut cependant interpréter cette phase d'amélioration individuelle comme un apprentissage permettant à l'individu de mieux s'adapter à l'environnement. En tout cas, il est clair aujourd'hui que les GA avec recherches locales (appelés *GA hybrides*) sont bien meilleurs que les GA classiques et peuvent même surpasser les autres métaheuristiques comme le recuit simulé et les méthodes tabou : par exemple, le GA hybride proposé par Prins (2000) pour le problème d'ordonnancement open-shop surpasse trois méthodes taboues publiées.

Notre recherche locale opère sur l'ensemble des tournées d'une solution complète, sans modifier le découpage en tournées. Chaque itération en $O(t^2)$ teste les déplacements de tâche (dans la même tournée ou vers une autre tournée) et les permutations de deux tâches (de la même tournée ou de tournées différentes). Pour chaque tâche u déplacée ou permutée, on teste deux cas : la tâche est réinsérée en conservant son sens de collecte ou bien en inversant ce sens (on réinsère $inv(u)$). La première transformation améliorante détectée est effectuée. Le processus est répété jusqu'à ce qu'on ne trouve plus d'amélioration.

Notez que les déplacements de tâches peuvent vider complètement une tournée et économiser ainsi un camion. Une tournée peut se vider temporairement durant la recherche locale, puis recevoir de nouvelles tâches lors des itérations suivantes. C'est pourquoi de telles tournées vides sont détruites uniquement à la fin de la recherche locale.

4.5. Structure de la population - Population initiale

La population stocke un nombre fixe *MaxPopu* de chromosomes dans un tableau *Popu*. Ces solutions sont construites au départ avec une méthode séquentielle choisissant aléatoirement à chaque itération une tâche non encore traitée, ainsi que son orientation s'il s'agit d'une arête. La tournée en cours est terminée quand la tâche suivante viole la capacité du camion. De telles solutions aléatoires étant en moyenne très mauvaises, nous incluons dans la population la solution obtenue par la très bonne heuristique Augment-Merge (Golden *et al.*, 1983) que nous rappelons en 4.6.

La population peut être gérée en autorisant ou en interdisant l'apparition de solutions identiques (*clones*). La détection des clones étant coûteuse dans le second cas, nous nous contentons d'individus de coûts tous distincts. On vérifie alors au début que les individus placés un par un dans *Popu(1)*, *Popu(2)*... ont bien un nouveau coût. En cas de conflit, on tente *MaxTries* fois (50 par exemple) de générer l'individu courant *Popu(k)*. En cas d'échec, la population est tronquée à *k-1* individus, ce qui ne se produit que quand on se fixe un effectif *MaxPopu* trop élevé. Le test d'unicité d'un coût se fait en $O(1)$ avec un tableau de booléens indicé par les valeurs des coûts.

4.6. Heuristique Augment-Merge

Cette heuristique ressemble à la fameuse méthode de la marguerite pour le VRP, exposée par exemple dans Christofides *et al.* (1979). Voici son principe pour le CARP de base, non orienté. On part de tournées triviales collectant chacune une tâche. Dans une première phase, dite *d'augmentation*, on cherche à éliminer les petites tournées dont l'unique tâche est traversée par une tournée plus grande : si la capacité résiduelle du camion le permet, la grande tournée peut collecter cette tâche en passant, absorbant ainsi la petite tournée. Dans la seconde phase, dite de *fusion*, on examine les paires de tournées pouvant être enchaînées par un seul camion : si on enchaîne une tournée finissant par une tâche *u* et une autre tournée débutant par une tâche *v*, le gain en coût (*saving*) est $D(u,nai)+D(nai,v)-D(u,v)$ car on évite un aller-retour au dépôt (codé par le pseudo-arc *nai*). Si ce gain est positif, on peut remplacer les deux tournées par une seule. Ce processus de fusion est appliqué tant qu'on peut diminuer le coût total.

La figure 4 précise l'ordre des opérations dans les phases Augment et Merge. Cet ordre est critique pour avoir de bons résultats en moyenne. L'heuristique peut être implémentée en $O(t^3)$. $A(i)$ est un indicateur valant Vrai si et seulement si la tournée initiale $n^{\circ} i$ a été absorbée par une autre.

```
//Initialisations
Construire une tournée pour chaque tâche
Trier ces t tournées par coût décroissant
A(i) := Faux pour chaque tournée Trip(i)
//Phase d'augmentation (Augment)
Pour i := 1 à t-1 avec A(i)=Faux
  Pour j := i+1 à t avec A(j)=Faux
    Si Trip(i) traverse la tâche unique u
collectée par Trip(j) et si la capacité
restante de Trip(i) permet de collecter u
    alors
      Collecter u par Trip(i)
      A(j) := Vrai //Trip(j) est absorbée
    FinSi
  FinPour
FinPour
Supprimer les tournées absorbées

//Phase de fusion (Merge)
Pour chaque Trip(i)
  Pour chaque Trip(j)
    Si la capacité de Trip(i) permet de
collecter les tâches de Trip(j) alors
      Rechercher la concaténation qui donne
un gain positif et minimal
      Si cette concaténation existe alors
        Trip(i) et Trip(j) sont concaténés
dans Trip(i).
        Supprimer Trip(j)
      FinSi
    FinSi
  FinPour
FinPour
```

Figure 4. Heuristique Augment-Merge

Pour le CARP de base, la phase Augment ne change pas le coût de la tournée absorbante parce qu'il n'y a pas de différence entre coût de collecte et coût de passage sans collecte. Pour notre extension du CARP, il faut en fait vérifier si le traitement de *u* par *Trip(i)* au lieu de *Trip(j)* diminue le coût total de la solution, et absorber *u* seulement dans ce cas.

4.7 Gestion des générations et critères d'arrêt

Nous utilisons la technique de sélection de Reeves (1995). La population est triée *en ordre décroissant des coûts*. Le rang *k* du 1^{er} parent est choisi avec une probabilité de $2k / (MaxPopu(MaxPopu+1))$, par une procédure *FitnessChoice*. Ainsi, la probabilité de tirer un individu de coût médian est d'environ $1 / MaxPopu$, celle de tirer le dernier (le meilleur) est quasiment double : $2 / (MaxPopu+1)$. Le rang du second parent est choisi avec une probabilité $1 / MaxPopu$ par une procédure *UniformChoice*. Le croisement choisi au départ du GA (X1, LOX ou OX) est ensuite appliqué. On ne conserve qu'un des deux fils, par tirage au sort.

Le remplacement est *incrémental* : le fils écrase un chromosome choisi uniformément sous la médiane (rang 1 à $MaxPopu / 2$) par une procédure *WhoIsKilled*. Cette technique préserve la meilleure solution et est connue pour sa bonne convergence, due au fait qu'un enfant peut immédiatement se reproduire. L'autre remplacement classique, moins efficace ici, consiste à ranger les enfants dans un autre tableau représentant la génération suivante. Quand ce tableau est plein, il remplace en bloc la population actuelle. Les GA sont stoppés après un nombre maximum de croisements (*MaxIt*), ou après un nombre maximum de croisements sans amélioration de la meilleure solution (*MaxItStable*). Les GA s'arrêtent aussi en atteignant une borne inférieure *LB* du coût minimal, fournie avec certaines instances.

```

01.//Initialisations
02.Calculer le distancier avec l'algorithme
   de Dijkstra modifié
03.Calculer dans Popu(1) la solution
   de Augment-Merge
04.Rank := 1
05.Répéter
06. Rank := Rank + 1
07. Tries := 0;
08. Répéter
09.   Tries := Tries + 1
10.   Générer aléatoirement une solution S
11.   Jusqu'à(Tries = MaxTries) ou (CloneOK(S))
12.   Si CloneOK(S) alors Popu(Rank) := S
13. Jusqu'à(Rank = MaxPopu) ou (non CloneOK(S))
14. Si Rank < MaxPopu alors MaxPopu := Rank - 1
15. Trier Popu par coût décroissant
16.
17.//Iteration principale
18.It := 0

```

4.8. Résumé – Structure générale

Au début de l'algorithme, on choisit *MaxPopu*, *MaxTries*, *MaxItStable*, l'opérateur de croisement, celui de mutation, le taux de mutation *PMut* et le mode de gestion avec ou sans clones. *Crossover(P1,P2,S)* croise les chromosomes *Popu(P1)* et *Popu(P2)*, choisit au hasard un des deux fils et le renvoie dans le chromosome *S*. *Mutate(S)* mute sur place le chromosome *S*. *CloneOK(S)* est une fonction booléenne renvoyant vrai si le coût de *S* n'est pas déjà présent dans la population. Elle renvoie toujours Vrai si les clones sont permis. *Mutate* et *Crossover* calculent aussi le coût de *S*, *TotCost(S)*. On obtient finalement la structure générale de la figure 5.

```

19.ItStable := 0
20.Répéter
21.  FitnessChoice (P1)
22.  UniformChoice (P2)
23.  Crossover (P1,P2,S)
24.  WhoIsKilled (Rank)
25.  Si Random < PMut alors Mutate(S)
26.  Si CloneOK(S) alors
27.    It := It + 1
28.    Si TotCost(S) < TotCost(Popu(1)) alors
29.      ItStable := 0
30.    Si non
31.      ItStable := ItStable + 1
32.    FinSi
29.    Popu(Rank) := S
30.    Remettre Popu(k) en position triée
31.  FinSi
32. Jusqu'à (It=MaxIt)ou(ItStable=MaxItStable)
   ou (TotCost(Popu(Maxpopu))=LB).

```

Figure 5. Structure générale de l'algorithme génétique

5. ÉVALUATION NUMÉRIQUE

Les GA ont été programmés en Delphi 5 (successeur 32 bits de Turbo Pascal) et exécutés sur un PC à 350 MHz sous Windows 95. Après plusieurs tests, les meilleurs résultats ont été obtenus avec de petites populations ($MaxPopu = 35$) sans clones et avec le croisement OX. Le GA hybride, avec la recherche locale RL comme opérateur de mutation, est nettement plus performant que les versions avec Move ou Swap.

L'avantage des populations sans clones est une meilleure dispersion des solutions, ce qui favorise une bonne exploration de l'espace de recherche tout en évitant une convergence prématurée. Du coup, on peut se permettre d'appliquer comme mutation la recherche locale avec des taux relativement élevés (10 à 30 %), sans que la convergence du GA soit trop rapide. Le seul inconvénient est que le GA perd un certain temps dans des croisements improductifs, dont les enfants sont rejetés à cause d'un coût déjà présent dans la population. C'est cette constatation qui nous a conduit à prendre de petites populations : pour $MaxPopu \leq 35$, le taux de rejets est raisonnable, de l'ordre de 10 à 20% selon les instances. Pour des populations plus grandes, ce taux augmente vite, et le GA consacre l'essentiel de

son temps à des itérations infructueuses. De plus, la population initiale, qui elle aussi est sans clones, devient difficile à générer.

Les tableaux comparent ce GA hybride sur des instances de la littérature, par rapport au meilleur algorithme connu pour le CARP, la méthode tabou *Carpet* de Hertz (2000). Ces instances sont disponibles à l'adresse donnée sous (Belenguer, 1997). Il s'agit de CARP non orientés, avec toutes les arêtes à collecter, c'est-à-dire en fait des problèmes de postier chinois avec capacités.

La colonne *Pb* donne la référence du problème. Celle *n,m* indique les nombres de nœuds et d'arêtes. *LB* donne la borne inférieure de Belenguer et Benavent (1997), fournie avec les instances. concernant les autres colonnes, Hertz (2000) souligne que trop d'articles sur les métaheuristiques montrent pour chaque instance un résultat avantageux, obtenu après avoir testé divers réglages de paramètres. Le meilleur réglage pour l'instance est rarement mentionné, ce qui fait qu'un lecteur reprogrammant la méthode avec un réglage fixe a peu de chances de retrouver d'aussi bons résultats. Hertz distingue donc pour sa méthode tabou *Carpet* une colonne *Best* donnant le meilleur résultat obtenu après

différents réglages des paramètres, et une colonne *Tabu* pour les résultats avec un réglage unique qui donne les meilleurs résultats *en moyenne* (ce réglage est fixe pour toutes les instances d'un tableau). Nous reprenons ces colonnes dans nos tableaux.

Pour faciliter la comparaison avec Carpet, la colonne *GA* indique les résultats de notre GA hybride avec un réglage unique des paramètres. La dernière colonne *Ecart* fournit l'écart entre le GA et Carpet, en pourcents. Les nombres en gras indiquent les cas où le GA fait aussi bien que Carpet, ceux en italique signalent les améliorations par rapport à Carpet.

Le *tableau 1* concerne les 23 instances de DeArmon (1981). Les instances 8-9 sont omises par tous les auteurs car elles contiennent des erreurs. Ces problèmes

ont de 7 à 27 nœuds et 11 à 55 arêtes. Le taux de mutation avec RL vaut 20 %. Le GA stoppe quand *LB* est atteint ou après *MaxIt* = 100000 croisements productifs (non rejetés pour clonage, soit environ 80 % des croisements). Le test d'arrêt sur *MaxItStable* n'a pas été utilisé.

Notre GA hybride est très efficace : sur les 23 instances, il fait mieux que Carpet sur un cas, aussi bien sur 20, et un peu moins bien sur 2. Il atteint 20 fois la meilleure solution connue et l'optimum est prouvé par atteinte de *LB* dans 17 cas. L'écart par rapport à Carpet vaut 0.03 % en moyenne et 1.65 % au maximum. L'écart moyen par rapport à *LB* est 0.53 %. En utilisant plusieurs réglages, le GA retrouve toutes les meilleures solutions connues. Le temps de calcul moyen par instance est de trois minutes.

| Pb | n,m | LB | Best | Tabu | GA | Ecart |
|----|-------|-----|------|------|------------|--------------|
| 1 | 12,22 | 316 | 316 | 316 | 316 | 0.00 |
| 2 | 12,26 | 339 | 339 | 339 | 339 | 0.00 |
| 3 | 12,22 | 275 | 275 | 275 | 275 | 0.00 |
| 4 | 11,19 | 287 | 287 | 287 | 287 | 0.00 |
| 5 | 13,26 | 377 | 377 | 377 | 377 | 0.00 |
| 6 | 12,22 | 298 | 298 | 298 | 298 | 0.00 |
| 7 | 12,22 | 325 | 325 | 325 | 325 | 0.00 |
| 10 | 27,46 | 344 | 348 | 352 | 356 | 1.13 |
| 11 | 27,51 | 303 | 311 | 317 | 311 | -1.89 |
| 12 | 12,25 | 275 | 275 | 275 | 275 | 0.00 |
| 13 | 22,45 | 395 | 395 | 395 | 395 | 0.00 |
| 14 | 13,23 | 448 | 458 | 458 | 458 | 0.00 |

| Pb | n,m | LB | Best | Tabu | GA | Ecart |
|----|-------|-----|------|------|------------|-------------|
| 15 | 10,28 | 536 | 544 | 544 | 544 | 0.00 |
| 16 | 7,21 | 100 | 100 | 100 | 100 | 0.00 |
| 17 | 7,21 | 58 | 58 | 58 | 58 | 0.00 |
| 18 | 8,28 | 127 | 127 | 127 | 127 | 0.00 |
| 19 | 8,28 | 91 | 91 | 91 | 91 | 0.00 |
| 20 | 9,36 | 164 | 164 | 164 | 164 | 0.00 |
| 21 | 11,11 | 55 | 55 | 55 | 55 | 0.00 |
| 22 | 11,22 | 121 | 121 | 121 | 123 | 1.65 |
| 23 | 11,33 | 156 | 156 | 156 | 156 | 0.00 |
| 24 | 11,44 | 200 | 200 | 200 | 200 | 0.00 |
| 25 | 11,55 | 233 | 233 | 235 | 235 | 0.00 |

Tableau 1. Résultats du GA sur les exemples de De Armon (1981).

| Pb | n,m | LB | Best | Tabu | GA | Ecart |
|----|-------|-----|------|------|------------|--------------|
| 1A | 24,39 | 173 | 173 | 173 | 173 | 0.00 |
| 1B | 24,39 | 173 | 173 | 173 | 179 | 3.46 |
| 1C | 24,39 | 235 | 245 | 245 | 245 | 0.00 |
| 2A | 24,34 | 227 | 227 | 227 | 227 | 0.00 |
| 2B | 24,34 | 259 | 259 | 260 | 259 | -0.38 |
| 2C | 24,34 | 455 | 457 | 494 | 457 | -7.49 |
| 3A | 24,35 | 81 | 81 | 81 | 81 | 0.00 |
| 3B | 24,35 | 87 | 87 | 87 | 87 | 0.00 |
| 3C | 24,35 | 137 | 138 | 138 | 138 | 0.00 |
| 4A | 41,69 | 400 | 400 | 400 | 408 | 2.00 |
| 4B | 41,69 | 412 | 412 | 416 | 420 | 0.96 |
| 4C | 41,69 | 428 | 430 | 453 | 434 | -4.19 |
| 4D | 41,69 | 520 | 546 | 556 | 536 | -3.59 |
| 5A | 34,65 | 423 | 423 | 423 | 423 | 0.00 |
| 5B | 34,65 | 446 | 446 | 448 | 446 | -0.44 |
| 5C | 34,65 | 469 | 474 | 476 | 481 | 1.05 |
| 5D | 34,65 | 571 | 593 | 607 | 589 | -2.96 |

| Pb | n,m | LB | Best | Tabu | GA | Ecart |
|-----|-------|-----|------|------|------------|--------------|
| 6A | 31,50 | 223 | 223 | 223 | 223 | 0.00 |
| 6B | 31,50 | 231 | 233 | 241 | 233 | -3.31 |
| 6C | 31,50 | 311 | 317 | 329 | 317 | -3.64 |
| 7A | 40,66 | 279 | 279 | 279 | 283 | 1.43 |
| 7B | 40,66 | 283 | 283 | 283 | 283 | 0.00 |
| 7C | 40,66 | 333 | 334 | 343 | 336 | -2.04 |
| 8A | 30,63 | 386 | 386 | 386 | 386 | 0.00 |
| 8B | 30,63 | 395 | 395 | 401 | 401 | 0.00 |
| 8C | 30,63 | 517 | 528 | 533 | 538 | 0.93 |
| 9A | 50,92 | 323 | 323 | 323 | 327 | 1.23 |
| 9B | 50,92 | 326 | 326 | 329 | 336 | 2.12 |
| 9C | 50,92 | 332 | 332 | 332 | 336 | 1.20 |
| 9D | 50,92 | 382 | 399 | 409 | 410 | 0.24 |
| 10A | 50,97 | 428 | 428 | 428 | 433 | 1.16 |
| 10B | 50,97 | 436 | 436 | 436 | 446 | 2.29 |
| 10C | 50,97 | 446 | 446 | 451 | 456 | 1.10 |
| 10D | 50,97 | 524 | 536 | 544 | 549 | 0.91 |

Tableau 2. Résultats du GA sur les instances de Belenguer *et al.* (1997)

Le tableau 2 concerne 34 instances plus dures de Belenguer et Benavent (1997), avec 24 à 50 nœuds et 34 à 97 arêtes. On a choisi *Pmut* = 30 % pour RL et *MaxIt* = 100000. Après 10000 croisements sans amélioration, le GA est redémarré en conservant la meilleure solution et en remplissant le reste du tableau Popu avec de nouvelles solutions aléatoires. On permet au maximum deux redémarrages de ce type. La déviation

moyenne par rapport à Carpet est de -0,23 %, soit un écart de 1,65 % par rapport à *LB*. La pire déviation par rapport à CARPET est 3,46 %. Le GA améliore Carpet 9 fois et fait aussi bien 11 fois. Il atteint la meilleure solution connue 15 fois et la borne inférieure 10 fois. Le temps de calcul moyen par instance passe à environ 10 minutes.

Le tableau 3 résume les performances de notre GA. Il donne le nombre d'instances pour lesquelles le GA est aussi bon que Carpet (= Carpet) ou meilleur que Carpet (< Carpet). Il indique aussi le nombre de fois où on atteint la meilleure solution connue (= Best) ou un

optimum prouvé (= LB). Les dernières colonnes indiquent respectivement la moyenne et le pire des écarts avec Carpet, ainsi que la déviation moyenne par rapport à l'optimum.

| Instances | < Carpet | = Carpet | = Best | = LB | Ecart moyen à Carpet | Pire écart à Carpet | Ecart moyen à LB |
|---------------|----------|----------|--------|------|----------------------|---------------------|------------------|
| DeArmon(23) | 1 | 18 | 17 | 15 | 0.14 % | 1.65 % | 0.57 % |
| Belenguer(34) | 9 | 11 | 15 | 10 | -0.23 % | 3.46 % | 1.65 % |

Tableau 3. Synthèse des résultats obtenus pour les deux types d'instances.

6. CONCLUSION

Cet article présente des résultats préliminaires sur les premiers GA publiés pour le CARP. Notre meilleur GA hybride est déjà compétitif avec la méthode tabou de Hertz sur les instances de DeArmon, avec un écart moyen de 0.14% par rapport à Carpet. Il est un peu meilleur sur celles de Belenguer, avec un écart moyen de -0.18%. Ces travaux confirment que, contrairement à une opinion répandue, les algorithmes génétiques peuvent rivaliser avec des métaheuristiques plus réputées en optimisation combinatoire, comme le recuit simulé et la méthode tabou. Cette efficacité requiert cependant l'introduction de connaissances spécifiques au problème à traiter, par exemple par le biais de la recherche locale utilisée comme opérateur de mutation.

Rappelons que les instances que nous avons utilisées dans nos comparaisons sont des cas particuliers du CARP de base (postier chinois non orienté, avec capacités). En fait, grâce à notre structure de donnée, nos GA ont déjà l'avantage de traiter un CARP étendu, avec un graphe mixte, des coûts de collecte distincts des coûts de traversée, et des interdictions de tourner. Nous avons dû nous contenter pour les tests des instances précédentes, car aucune méthode publiée n'est disponible pour des comparaisons avec nos GA sur le CARP étendu.

Nous prévoyons d'améliorer les performances tout en gérant plus de contraintes réelles, comme des flottes hétérogènes de véhicules, différents types de déchets, des secteurs géographiques avec priorités, ou des fenêtres de temps sur les arcs.

REFERENCES

Belenguer J.M. et E. Benavent, 1997. *A cutting plane algorithm for the capacitated arc routing problem*. Rapport de Recherche, Département de Statistiques et de Recherche Opérationnelle, Université de Valencia (Espagne), à paraître dans *Computer and Operations Research*. Instances disponibles à : <ftp://matheron.estadi.uv.es/pub/CARP>.

Benavent E., V. Campos et A. Corberan, 1992. The capacitated arc routing problem : lower bounds, *Networks*, 22, p. 669-690.

Christofides N., A. Mingozzi et P. Toth, 1979. Chapitre *The vehicle routing problem*, p. 315-338 dans *Combinatorial optimization*, coordonné par ces mêmes auteurs et C. Sandi, Wiley.

Cormen T.H., C.L. Leiserson et M.L. Rivest, 1990. *Introduction to algorithms*, The MIT Press.

DeArmon J.S., 1981. *A Comparison of Heuristics for the Capacitated Chinese Postman Problem*. Master's Thesis, The University of Maryland at College Park, MD, USA.

Eglese R.W., 1994. Routing winter gritting vehicles, *Discrete Applied Math.*, 48(3), p. 231-244.

Fleurent C, J.A. Ferland, 1996. Algorithmes génétiques hybrides pour l'optimisation combinatoire, *RAIRO-Operations Research*, 30(4), 373-398.

Golden B.L. et R.T. Wong, 1981. Capacitated arc routing problems, *Networks*, 11, p. 305-315.

Golden B.L., J.S. DeArmon et E.K. Baker, 1983. Computational experiments with algorithms for a class of routing problems, *Computers and Operation Research*, 10(1), p. 47-59.

Hertz A., G. Laporte et M. Mittaz, 2000. A Tabu Search Heuristic for the Capacitated Arc Routing Problem, *Operations Research*, 48(1), p. 129-135.

Hirabayashi R., Y. Saruwatari et N. Nishida, 1992. Tour construction algorithm for the capacitated arc routing problem, *Asia-Pacific Journal of Operational Research*, 9, p. 155-175.

Holland J., 1975. *Adaptation in natural and artificial systems*, University of Michigan Press.

Pearn W.L., 1989. Approximate solutions for the capacitated arc routing problem, *Computers and Operations Research*, 16(6), p. 589-600.

Pearn W.L., 1991. Augment-insert algorithms for the capacitated arc routing problem, *Computers and Operations Research*, 18(2), p. 189-198.

Prins C., 2000. Competitive genetic algorithms for the open-shop scheduling problem, *Mathematical Methods of Operations Research*, 51, p. 540-564.

Reeves C.R., 1995. A genetic algorithm for flowshop sequencing, *Computers and Operations Research*, 22(1), p. 5-13.

Reeves C.R., 1996. *Modern heuristic techniques*, Blackwell Scientific, Oxford, UK.

Ulusoy G., 1985. The Fleet Size and Mixed Problem for Capacitated Arc Routing, *European Journal of Operational Research*, 22, p. 329-337.