4e Conférence Francophone de MOdélisation et SIMulation
"Organisation et Conduite d'Activités dans l'Industrie et les Services"
MOSIM'03 – du 23 au 25 avril 2003 - Toulouse (France)

OPTIMISATION PAR COLONIES DE FOURMIS POUR LES TOURNEES SUR ARCS

Philippe LACOMME

Université Blaise Pascal, LIMOS
(CNRS UMR 6158), BP 10125,
63173 Aubière Cedex
Mél : lacomme@sp.isima.fr

Christian PRINS

Université de Technologie de
Troyes, LOSI (JE 2304), BP 2060,
10010 Troyes Cedex
Mél : prins@utt.fr

Alain TANGUY

Université Blaise Pascal, LIMOS
(CNRS UMR 6158), BP 10125,
63173 Aubière Cedex
Mél : tanguy@sp.isima.fr

RESUME : *Le CARP (Capacitated Arc Routing Problem) est un problème de tournées NP-difficile modélisant par exemple la collecte des déchets ménagers. Les instances de grande taille doivent être traitées par des heuristiques ou des métaheuristiques. Cet article propose le premier algorithme à colonies de fourmis pour le CARP. Il inclut une recherche locale accélérant fortement le schéma de base des méthodes à fourmis. Sans être aussi efficace que l'algorithme génétique de Lacomme et al. (2001), il s'avère compétitif avec la meilleure méthode tabou publiée (Hertz et al., 2000). Ces performances sont actuellement exceptionnelles pour un algorithme à fourmis.*

MOTS-CLES : *transport, tournée sur arcs, CARP, métaheuristique, colonie de fourmis*

1. INTRODUCTION

Le CARP (*Capacitated Arc Routing Problem*) est un problème de tournées NP-difficile défini sur un graphe non orienté $G = (V,E)$ où V est un ensemble de n nœuds et E un ensemble de m arêtes. Une flotte homogène de véhicules de capacité Q est basée en un nœud-dépôt s. Chaque arête i possède un coût de traversée $c_i \geq 0$ et une demande $q_i \geq 0$. Les r arêtes de demandes non nulle doivent obligatoirement être traversées pour être traitées. On peut traverser une arête plusieurs fois mais son traitement doit être effectué par un seul véhicule et lors d'une seule traversée. Le CARP consiste à déterminer un ensemble de tournées (cycles passant par le dépôt) tel que :

- chaque arête à traiter est effectivement traitée, par une seule tournée ;
- la demande totale traitée par chaque tournée ne dépasse pas la capacité Q du véhicule ;
- le coût total des tournées (somme des coûts des arêtes traversées) est minimisé.

Les applications du CARP concernent surtout les réseaux routiers. Les demandes sont alors des quantités à collecter (déchets ménagers) ou à distribuer (sablage de routes en cas de verglas). Les coûts sont le plus souvent des distances ou des temps de parcours.

Les formulations par programmes linéaires en nombres entiers sont encore inutilisables pour des cas réels mais elles peuvent fournir d'excellentes bornes inférieures (Belenguer et Benavent, 1997). Les instances de grande taille doivent donc être traitées en pratique par des heuristiques. Parmi les heuristiques simples (constructives), on peut citer Path-Scanning (Golden et Wong, 1981), Augment-Merge (Golden *et al*., 1983) et la méthode d'Ulusoy (1985). Ces méthodes présentent le double intérêt de donner une bonne solution avec un temps de calcul raisonnable et de servir de point de départ pour des métaheuristiques. Ces dernières incluent la méthode de recuit simulé d'Eglese (1994), la méthode tabou de Hertz *et al*. (2000) et l'algorithme génétique de Lacomme, Prins et Ramdane-Chérif (2001), qui est actuellement le meilleur algorithme publié. L'objectif de cette publication est de prouver que les algorithmes de fourmis sont capables de concurrencer ces méthodes.

2. LES ALGORITHMES A COLONIES DE FOURMIS

Les techniques d'optimisation issues des colonies de fourmis (*Ant Colony Optimization* ou *ACO* en anglais) ont été appliquées à divers problèmes d'optimisation comme la coloration de graphes (Costa et Hertz, 1997) et le problème du voyageur de commerce (Dorigo et Gambardella, 1997). Les travaux sur les problèmes de tournées sont plus récents et ne concernent que les tournées sur les noeuds d'un réseau (Bullnheimer *et al*., 1999a, 1999b ; Reimann *et al*., 2002). Gutjahr (2000) a prouvé la convergence en probabilité de certains types d'algorithmes à fourmis, propriété connue pour le recuit simulé (Hajek, 1988) et les algorithmes génétiques (Fleury, 1993 ; Cerf, 1994).

Dans une itération d'algorithme ACO, f agents (fourmis) construisent chacun une solution d'après des décisions basées sur des critères heuristiques et sur des traces de phéromone. Les traces sont mises à jour en examinant les solutions obtenues. Elles sont renforcées pour les décisions ayant donné de meilleures solutions et diminuées pour les autres. Ce mécanisme permet

d'améliorer progressivement les solutions au cours des itérations. En pratique, on construit f solutions initiales, puis on répète l'itération générale suivante jusqu'à la réalisation d'un critère d'arrêt comme un nombre maximum d'itérations ou un écart donné par rapport à une borne inférieure :

- mise à jour des traces de phéromone dans le réseau ;
- génération de f nouvelles solutions par les fourmis, en exploitant les traces de phéromone ;
- application avec une probabilité donnée d'une recherche locale à ces solutions.

3. NOTATIONS UTILISEES DANS LA SUITE

Avant de décrire les étapes de notre algorithme, il nous faut quelques notations. Comme il y a deux sens de traitement possibles par arête, nous remplaçons chaque arête de G par deux arcs opposés. Dans la suite le terme "arc" désigne en fait un sens de traversée pour une arête à traiter et le terme "chemin" désigne toujours un plus court chemin entre deux arcs.

3.1. Constantes

f Nombre de fourmis. On identifie dans la suite les fourmis et leurs solutions, stockées dans une table triée par coût total décroissant. **La meilleure solution est donc la dernière (indice *f*)**.

w_{ij} Distance (coût du plus court chemin dans G) entre l'arc i et l'arc j.

M_d Plus grande distance dans le graphe (maximum des w_{ij})

s_{ij} Mesure de visibilité de l'arc j depuis l'arc i : $s_{ij} = (M_d - c_{ij})/M_d$

r Taux de persistance des traces de phéromone ($0 \le r \le 1$).

F^m Poids constant affecté à la fourmi n° m ("fourmi m" ou "m" pour abréger).

p_a Probabilité de déplacement aveugle, ignorant les traces de phéromone.

k Nombre d'arcs les plus proches considérés lors d'un déplacement aveugle.

P_{LS} Probabilité de lancer une recherche locale sur une solution.

3.2. Variables

t_{ij} Quantité de phéromone sur le chemin de l'arc i à l'arc j.

Δt_{ij}^m Ajout de phéromone par la fourmi m sur le chemin de l'arc i vers l'arc j.

L^m Coût total de la solution trouvée par la fourmi m.

T_m Liste "tabou" de la fourmi m (ensemble des arcs déjà traités par la fourmi).

Ω_i^m Ensemble des k arcs les plus proches de l'arc i (en terme de s_{ij}), non encore traités par m.

Ψ_i^m Ensemble des k meilleurs arcs (en terme de trace t_{ij}), non encore traités par m.

P_{ij}^m Probabilité pour une fourmi m de se déplacer de l'arc i à l'arc j.

4. GENERATION DES SOLUTIONS INITIALES

Les solutions de départ d'un algorithme ACO sont en général aléatoires. Pour le CARP, chaque solution est d'abord construite sous forme d'une tournée unique, en ignorant la capacité des véhicules. Une telle tournée est une séquence aléatoire de m arcs à traiter, implicitement reliés par des plus courts chemins. Chaque arête à traiter y est représentée par un de ses arcs. Une procédure Split décrite par Lacomme *et al*. (2001) découpe ensuite de façon optimale la séquence en tournées respectant la capacité. Les solutions obtenues sont en moyenne médiocres mais bien dispersées dans l'espace de recherche. Une particularité de notre implémentation est d'inclure trois solutions de bonne qualité, calculées par les heuristiques classiques suivantes pour le CARP : Path-Scanning (Golden *et al*., 1983), Augment-Merge (Golden et Wong, 1981) et la méthode d'Ulusoy (1985). Ces solutions initiales permettent de calculer la trace initale de phéromone dans le graphe.

5. ETAPES DE L'ITERATION GENERALE

5.1. Mise à jour des traces de phéromone

Au début, pour tout couple d'arêtes à traiter (i,j), la quantité de phéromone t_{ij} est nulle. On la met à jour au début de chaque itération selon la formule (1), qui comprend un terme pour l'évaporation et un pour le renforcement.

$$t_{ij} \leftarrow r.t_{ij} + \sum_{m=1}^{f} \Delta t_{ij}^m, \quad \text{avec } \Delta t_{ij}^m = \frac{F^m}{L^m} \qquad (1)$$

Une idée simple est d'affecter des poids égaux $F^m = 1$ aux fourmis, mais alors on ne tient pas compte de la qualité des solutions dans cette pondération supplémentaire des solutions. Les f solutions étant triées par coûts décroissants, une meilleure option est d'utiliser le rang de la fourmi en posant $F^m = m$. Nous utilisons en fait des poids plus fins qui tiennent compte de la distance maximale M_d entre deux arêtes à traiter : la formule (2) définit ainsi une fonction affine du rang m (entre 1 et f), prenant ses valeurs entre 1 et M_d.

$$F^m = \frac{M_d - 1}{f - 1} \times m + \frac{f - M_d}{f - 1} \qquad (2)$$

5.2. Déplacement des fourmis pour construire de nouvelles solutions

Chaque fourmi μ construit une solution à partir du dépôt, en choisissant des arcs à traiter de manière probabiliste. En un arc i, elle choisit le prochain arc j soit de façon aveugle (en ignorant les traces de phéromone), soit en tenant compte des traces. Les choix aveugles sont nécessaires pour explorer de nouvelles solutions. En fait, la fourmi est aveugle avec une probabilité p_a et choisit alors j au hasard parmi les k arcs les plus proches qui lui restent à traiter (formule (3)). Avec la probabilité $1-p_a$, elle tient compte des phéromones et choisit j avec la formule (4). Cette dernière formule est classique et tient compte à la fois des visibilités s_{ij} et des traces t_{ij}, pondérées par des puissances choisies a et b.

$$P_{ij}^m = \begin{cases} 1/k \text{ si } j \in \Omega_i^m \\ 0 \quad \text{sinon} \end{cases} \tag{3}$$

$$P_{ij}^m = \begin{cases} \dfrac{[s_{ij}]^a [t_{ij}]^b}{\sum_{q \in \Psi_i^m} [s_{ij}]^a [t_{ij}]^b} & \text{si } j \in \Psi_i^m \\ 0 \quad \text{sinon} \end{cases} \tag{4}$$

5.3. Recherche locale

Chaque solution calculée par une fourmi subit une procédure d'amélioration (recherche locale) avec une probabilité P_{LS}. La recherche locale employée, est celle proposée dans l'algorithme génétique de Lacomme *et al.* (2001). Elle consiste à examiner les transformations suivantes dans une tournée ou entre deux tournées :

- déplacer une arête à traiter avant/après une autre ;
- déplacer deux arêtes à traiter consécutives ;
- transformations de type 2-opt, bien connues pour le problème du voyageur de commerce.

En déplaçant les arêtes (représentées chacune par deux arcs), on change éventuellement le sens de traversée si cela procure un gain plus important. La recherche locale s'exécute tant qu'elle trouve des transformations améliorant la solution. Comme cela a déjà été observé pour l'algorithme génétique la recherche locale est un élément essentiel car elle permet d'accentuer l'effort de recherche en "tirant" les solutions vers le bas. Le paramètre P_{LS} est donc un paramètre important de l'algorithme. Une valeur trop importante de P_{LS} va entraîner des temps de calcul plus longs sans obtenir d'amélioration significative des solutions et en augmentant le risque de bloquer l'algorithme dans un minimum local. Une valeur trop petite n'intensifiera pas assez la recherche et la convergence sera plus lente.

6. STRUCTURE DE L'ALGORITHME

La population des f fourmis comprend en fait f_e fourmis "élitistes" et $f - f_e$ fourmis "non-élitistes". Les élitistes assurent la convergence de l'algorithme, tandis que les non-élitistes explorent l'espace de recherche pour maintenir la diversité des solutions et prévenir une convergence prématurée. Dans le tableau des f solutions, on remplace la dernière solution d'une fourmi élitiste par la nouvelle seulement en cas d'amélioration. Par contre, on remplace toujours la dernière solution d'une fourmi non-élitiste par sa nouvelle solution, qu'il y ait amélioration ou dégradation.

```
Calculer les solutions initiales (§ 4)
Mettre les traces de phéromone et I (nombre d'itérations) à 0
Répéter
   Mise à jour des traces de phéromone (§ 5.1)
   Pour μ ← 1 à f faire
      Répéter
         Choisir le prochain arc j à traiter (§ 5.2)
         Ajouter j à la solution en cours de construction
         Mettre à jour la liste tabou T_m de la fourmi
      Jusqu'à ce que la fourmi μ ait terminé sa tournée
      Evaluer la solution de la fourmi μ avec Split (§ 4)
            Appliquer la recherche locale avec une probabilité donnée P_LS (§ 5.3)
      Si μ est une fourmi non-élitiste alors
         Remplacer la dernière solution de μ par la nouvelle
      Sinon
         si la nouvelle solution est meilleure que la précédente alors
            Remplacer la dernière solution de μ par la nouvelle
         FinSi
      FinSi
   FinPour
   Trier les solutions par coût décroissant
   Si la meilleure solution est inchangée depuis I_s itérations alors
      Mettre à 0 les traces de phéromone
   FinSi
   I ← I + 1
Jusqu'à ce que (une borne inférieure LB soit atteinte) ou ( I = I_max )
```

Figure 1. *Algorithme de principe*

Notons qu'à toute itération les f_e solutions des fourmis élitistes sont les meilleures solutions découvertes depuis le début de l'algorithme. L'algorithme est renforcé par une remise à 0 périodique des traces de phéromone, qui constituent la "mémoire" de l'algorithme. Ce nettoyage est effectué chaque fois que I_s itérations ont été effectuées sans amélioration de la meilleure solution. La *figure 1* résume la structure de notre algorithme ACO pour le CARP.

7. EVALUATION NUMERIQUE

7.1. Paramètres de l'algorithme

f = 30 fourmis	P_{LS} = 0.5 probabilité de recherche locale
a = b = 1, pondérations pour la formule (4)	f_e = 5 fourmis élitistes
r = 0.9, taux de persistance de la phéromone	I_{max} = 200 itérations
k = 10, taille des ensembles Ψ_i^m et Ω_i^m	I_s = 10 itérations sans amélioration avant de remettre à 0 les traces de phéromone
p_a = 0.1, probabilité de déplacement aveugle	

7.2. Jeux d'essai utilisés et format des résultats

Notre algorithme ACO a été programmé en Delphi puis exécuté sur un Pentium III à 800 MHz sur trois ensemble d'instances classiques pour le CARP : les fichiers *gdb*, *val* et *egl*. Les résultats obtenus sont présentés respectivement dans les tables 1, 2 et 3, avec les colonnes suivantes.

LB	meilleure borne inférieure connue (Belenguer et Benavent, 1997)
n,m	nombre de noeuds et nombre d'arêtes
TS	solution de la méthode tabou de Hertz *et al*. (2000) avec un seul réglage de paramètres
BTS	meilleure solution de *TS* (*Best TS*), obtenue avec divers réglages des paramètres
PS, AM, UL	solutions initiales des heuristiques Path-Scanning, Augment-Merge et d'Ulusoy
ACO	solution trouvée par notre algorithme à fourmis
*I**	itération à laquelle la solution présentée dans la colonne *ACO* a été trouvée
Time	temps de calcul en secondes pour la solution d'*ACO*.

Les astérisques indiquent des optima prouvés (*LB* est atteinte). Les valeurs en gras indiquent les cas où *ACO* est meilleure que *TS*. Les 2 dernières lignes indiquent la déviation moyenne à la borne en % (*LB dev*) et le nombre d'optima prouvés (*LB hit*).

7.3. Résultats sur les fichiers Gdb (table 1)

Ces 23 instances ont toutes leurs arêtes à traiter (r = m). Elles sont moins grandes que les val et egl, ce qui explique que 18 optima sont connus. Les heuristiques constructives PS, AM et UL trouvent au plus 2 optima et affichent un écart moyen à la borne d'au moins 6.4%. La table donne les résultats d'ACO pour 3 exécutions, réalisées avec trois germes différents pour le générateur de nombres aléatoires (Exec 1, 2 et 3). La colonne BACO (Best ACO) donne la meilleure solution.

La comparaison avec la meilleure méthode tabou publiée doit être effectuée dans des conditions équivalentes : un seul réglage des paramètres, colonnes TS et ACO. En termes d'optima trouvés, on constate que ACO est aussi efficace que TS. Selon le germe initial, il bat TS une ou deux fois et améliore une meilleure solution connue (gdb9). Il est même un peu plus robuste que TS en termes d'écart moyen à la borne. De plus, les résultats semblent peu sensibles aux valeurs du germe.

La convergence est assez rapide : pour le premier germe par exemple, l'algorithme converge en moyenne en 15 itérations et 22 secondes. Les temps nuls correspondent aux cas où une des heuristiques initiales a trouvé l'optimum.

File	n,m	LB	TS	BTS	PS	AM	UL	Exec 1			Exec 2			Exec 3			BACO
								ACO	I*	Time	ACO	I*	Time	ACO	I*	Time	
gdb1	12,22	316	316*	316*	350	349	330	316*	1	0.16	316*	1	0.16	316*	1	0.17	316*
gdb2	12,26	339	339*	339*	366	370	353	339*	3	0.44	339*	5	0.66	339*	3	0.44	339*
gdb3	12,22	275	275*	275*	293	319	297	275*	2	0.28	275*	1	0.11	275*	1	0.16	275*
gdb4	11,19	287	287*	287*	287*	302	320	287*	0	0.00	287*	0	0.00	287*	0	0.06	287*
gdb5	13,26	377	377*	377*	438	423	407	377*	1	0.16	377*	11	1.42	377*	1	0.16	377*
gdb6	12,22	298	298*	298*	324	340	318	298*	1	0.17	298*	3	0.33	298*	3	0.27	298*
gdb7	12,22	325	325*	325*	363	325*	330	325*	0	0.00	325*	0	0.06	325*	0	0.00	325*
gdb8	27,46	344	352	348	463	393	388	352	104	101.39	**350**	163	71.46	353	28	14.72	**350**
gdb9	27,51	303	317	311	354	352	358	**309**	55	130.72	**311**	46	26.59	**306**	138	76.07	**306**
gdb10	12,25	275	275*	275*	295	300	283	275*	5	0.66	275*	3	0.38	275*	2	0.33	275*
gdb11	22,45	395	395*	395*	447	449	413	395*	5	2.91	395*	17	7.74	395*	9	4.06	395*
gdb12	13,23	448	458	458	581	569	537	458	56	20.76	458	7	0.72	458	2	0.22	458
gdb13	10,28	536	544	544	563	560	552	544	3	31.20	544	4	0.60	544	2	0.33	544
gdb14	7,21	100	100*	100*	114	102	104	100*	1	0.11	100*	2	0.22	100*	3	0.27	100*
gdb15	7,21	58	58*	58*	60	60	58*	58*	0	0.06	58*	0	0.05	58*	0	0.06	58*
gdb16	8,28	127	127*	127*	135	129	132	127*	9	1.31	127*	6	0.83	127*	11	1.48	127*
gdb17	8,28	91	91*	91*	93	91*	93	91*	0	0.00	91*	0	0.05	91*	0	0.05	90*
gdb18	9,36	164	164*	164*	177	174	172	164*	1	0.33	164*	2	0.66	164*	1	0.27	164*
gdb19	8,11	55	55*	55*	57	63	63	55*	1	0.05	55*	1	0.06	55*	1	0.06	55*
gdb20	11,22	121	121*	121*	132	129	125	123	21	1.87	121*	191	6.42	121*	126	10.60	121*
gdb21	11,33	156	156*	156*	176	163	162	156*	35	8.18	156*	24	4.45	156*	49	9.23	156*
gdb22	11,44	200	200*	200*	208	204	207	200*	18	72.83	200*	200	60.03	200*	87	27.41	200*
gdb23	11,55	233	235	233*	251	237	239	235	19	139.80	235	8	4.45	235	3	2.14	235
LB dev			0.50	0.33	10.4	8.4	6.4	0.46			0.39			0.36			0.32
LB hit			18	19	1	2	1	17			18			18			18

Table 1. Résultats sur les fichiers gdb

FILE	n,m	LB	TS	BTS	PS	AM	UL	ACO	I*	Time
val1a	24,39	173	173*	173*	186	190	173*	173*	0	0.06
val1b	24,39	173	173*	173*	209	196	197	173*	179	57.51
val1c	24,39	235	245	245	331	294	280	245	21	6.59
val2a	24,34	227	227*	227*	259	238	255	227*	1	0.38
val2b	24,34	259	260	259*	284	275	281	**259***	2	0.66
val2c	24,34	455	494	457	516	533	515	**463**	66	14.77
val3a	24,35	81	81*	81*	88	86	85	81*	1	0.33
val3b	24,35	87	87*	87*	99	96	99	87*	22	6.37
val3C	24,35	137	138	138	158	152	153	138	118	26.86
val4A	41,69	400	400*	400*	451	443	436	400*	9	13.34
val4B	41,69	412	416	412*	487	487	468	**412***	16	21.97
val4C	41,69	428	453	430	539	483	486	**436**	93	116.89
val4D	41,69	520	556	546	656	631	608	**546**	198	239.14
val5A	34,65	423	423*	423*	476	466	451	423*	39	44.10
val5B	34,65	446	448	446*	508	487	486	**446***	16	17.85
val5C	34,65	469	476	474	544	509	504	**474**	117	124.24
val5D	34,65	571	607	593	720	667	660	**593**	58	58.55
val6A	31,50	223	223*	223*	271	241	243	223*	1	0.77
val6B	31,50	231	241	233	274	247	253	**233**	7	4.18
val6C	31,50	311	329	317	381	365	367	**317**	115	59.37
val7A	40,66	279	279*	279*	326	306	293	279*	6	7.85
val7B	40,66	283	283*	283*	353	314	295	283*	6	7.86
val7C	40,66	333	343	334	394	387	381	**334**	185	210.59
val8A	30,63	386	386*	386*	433	412	432	386*	27	27.02
val8B	30,63	395	401	395*	455	426	439	401	38	38.67
val8C	30,63	517	533	528	596	604	603	550	66	59.54
val9A	50,92	323	323*	323*	358	348	345	324	141	407.88
val9B	50,92	326	329	326*	352	358	350	**327**	64	184.55
val9C	50,92	332	332*	332*	394	368	368	337	111	318.02
val9D	50,92	382	409	399	492	436	462	**402**	49	134.40
va1l0A	50,97	428	428*	428*	453	453	452	428*	113	383.33
val10B	50,97	436	436*	436*	474	460	457	438	89	291.71
val10C	50,97	446	451	446*	503	478	496	**450**	71	233.70
val10D	50,97	524	544	536	614	590	589	544	105	318.13
LB dev			1.90	0.82	16.8	11.4	10.9	1.24		
LB hit			15	21	0	0	1	15		

Table 2. Résultats sur les fichiers val

7.4. Résultats sur les fichiers Val (table 2)

Ces 34 instances ont encore toutes leurs arêtes à traiter ($r = m$). L'algorithme *ACO* bat la méthode tabou 14 fois, fait aussi bien 16 fois et moins bien 4 fois, dont un petit accident sur *val8c* (550 contre 533 pour *TS*). L'écart moyen à la borne est nettement inférieur (1.24% au lieu de 1.90%). Pour l'ensemble fichiers *val*, 23 meilleures solutions connues sont retrouvées mais aucune n'est améliorée. Les fichiers *val* sont en moyenne plus grands que les *gdb*, ce qui se traduit par une convergence plus lente de l'algorithme à fourmis, en moyenne en 63 itérations au lieu de 15 et en 101 secondes au lieu de 22.

7.5. Résultats sur les fichiers Egl (table 3)

Ces 24 instances de grande taille (jusqu'à 190 arêtes) sont extraites du réseau routier réel du comté de Lancaster (UK) et sont beaucoup plus difficiles que les précédentes. Les heuristiques constructives s'effondrent. Les écarts des métaheuristiques aux bornes augmentent et aucune borne n'est atteinte (ce qui ne veut pas dire qu'aucune solution n'est optimale!). La colonne BTS n'est pas fournie car nous ne connaissons les résultats de la méthode tabou que pour son réglage standard de paramètres. La colonne TS donne donc les meilleures solutions connues. L'algorithme ACO bat la méthode tabou 8 fois et fait moins bien 16 fois. Mais il affiche le même écart moyen aux bornes. Il nécessite en moyenne 91 itérations et 21 minutes pour trouver les solutions mentionnées.

FILE	n,m,r	LB	TS	PS	AM	UL	ACO	I*	Time
egl-e1-A	77,98,51	3515	3625	4115	4605	3952	**3548**	23	57.46
egl-e1-B	77,98,51	4436	4532	5228	5494	5054	4539	32	77.55
egl-e1-C	77,98,51	5453	5663	7240	6799	6166	5687	49	119.79
egl-e2-A	77,98,72	4994	5233	6458	6253	5716	**5018**	16	48.12
egl-e2-B	77,98,72	6249	6422	7964	7923	7080	**6401**	111	337.29
egl-e2-C	77,98,72	8114	8603	10313	10453	9338	**8563**	166	493.50
egl-e3-A	77,98,87	5869	5907	7454	7350	6723	5986	160	600.39
egl-e3-B	77,98,87	7646	7921	9900	9244	8713	7959	90	339.05
egl-e3-C	77,98,87	10019	10805	12672	12556	11641	**10430**	90	319.94
egl-e4-A	77,98,98	6372	6489	7527	7798	7231	6579	48	200.36
egl-e4-B	77,98,98	8809	9216	10946	10543	10223	9248	33	136.38
egl-e4-C	77,98,98	11276	11824	13828	13623	13165	11958	154	626.26
egl-s1-A	140,190,75	4992	5149	6382	6143	5636	**5081**	41	634.82
egl-s1-B	140,190,75	6201	6641	8631	7992	7086	**6602**	48	743.85
egl-s1-C	149,190,75	8310	8687	10259	10338	9572	**8651**	196	3034.69
egl-s2-A	140,190,147	9780	10373	12344	11672	11475	10390	67	1496.28
egl-s2-B	140,190,147	12886	13495	16386	15178	14845	13650	186	4286.39
egl-s2-C	140,190,147	16221	17121	20520	19673	19290	17139	34	901.77
egl-s3-A	140,190,159	10025	10541	13041	11957	11956	10778	155	4170.00
egl-s3-B	140,190,159	13554	14291	17377	15891	15663	14438	39	1052.15
egl-s3-C	140,190,159	16969	17789	21071	19971	20064	18055	27	797.41
egl-s4-A	140,190,190	12027	13036	15321	14741	13978	13065	90	2852.56
egl-s4-B	140,190,190	15933	16924	19860	19172	18612	17251	166	3942.99
egl-s4-C	140,190,190	20179	21486	25921	24175	23727	21654	152	3701.65
LB dev			4.74	26.4	22.8	15.4	4.80		
LB hit			0	0	0	0	0		

Table 3. Résultats sur les fichiers egl

8. CONCLUSION

Cet article présente le premier algorithme à fourmis pour le CARP. Ses points remarquables sont l'utilisation des heuristiques constructives pour fournir trois bonnes solutions initiales et l'amélioration des solutions par recherche locale à la fin de chaque itération. De manière consistante, l'évaluation numérique sur trois bibliothèques d'instances classiques montre que l'algorithme est aussi efficace que la meilleure méthode tabou publiée et qu'il améliore même 9 meilleures solutions connues.

Cette première tentative est très encourageante car un nombre assez faible d'itérations a été accordé à l'algorithme. Le processus de convergence semble être correctement géré puisque sur certaines instances l'algorithme améliore encore sa meilleure solution vers la dernière itération (200). Cela montre que l'algorithme ne converge pas prématurément vers des minima locaux dont il ne pourrait plus s'extraire. Ce bon comportement est probablement favorisé par la remise à 0 périodique des traces de phéromone.

Les temps de calcul sont très raisonnables par rapport à ceux habituellement rapportés dans la littérature pour des algorithmes à fourmis. Ils se dégradent assez vite quand la taille des instances augmente (gros fichiers *egl*), mais l'implémentation actuelle n'est pas spécialement optimisée et elle pourrait certainement être accélérée. Bien que notre méthode n'égale pas encore l'algorithme génétique hybride de Lacomme, Prins et Ramdane-Chérif (2001), qui utilise des croisements très rapides, elle affiche des performances exceptionnelles pour un algorithme à colonie de fourmis.

Les algorithmes à colonies de fourmis fournissent de très bons résultats pour les problèmes de TSP avec plusieurs centaines de nœuds, mais ne concurrencent pas les meilleures méthodes publiées dans le domaine. Les résultats obtenus sur le CARP montrent que ces algorithmes fournissent des résultats comparables en qualités à ceux obtenus avec des métaheuristiques issues du recuit simulé. Néanmoins, le principal inconvénient de l'approche réside dans le coût relativement élevé de la génération des solutions. Les temps de calcul pour une itération de l'algorithme à fourmis nous ont conduit à réaliser un très faible nombre d'itérations par rapport au nombre d'itérations réalisées par l'algorithme génétique. Cette grande différence dans le nombre d'itérations explique partiellement la dominance de l'algorithme génétique par rapport à l'algorithme de fourmis. D'autres éléments devraient aussi être considérés tels que le schéma d'optimisation lui-même mais les conclusions sur ce domaine sont particulièrement délicates.

REFERENCES

Belenguer J.M. et E. Benavent. 1997. A cutting plane algorithm for the capacitated arc routing problem. Rapport de Recherche, Dép. de Statistiques et RO, Université de Valencia, Espagne.

Bullnheimer B., R.F. Hartl et C. Strauss. 1999a. Applying the ant systems to the vehicle routing problem. Dans S. Voss, S. Martello, I.H. Osman et C. Roucairol (éd.), Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer.

Bullnheimer B., R.F. Hartl et C. Strauss. 1999b. An improved ant system algorithm for the Vehicle Routing Problem, Annals of Operations Research, 89, pp. 319-328.

Cerf R. 1994. Une théorie asymptotique des algorithmes génétiques. Thèse de l'Université de Montpellier II.

Costa D. et A. Hertz. 1997. Ants can colour graphs, JORS, 48(3), pp. 295-305.

Dorigo M. et L.M. Gambardella. 1997. Ant Colony System: a cooperative leaning approach to the Travelling Salesman Problem, IEEE Transactions on Evolutionary Computation, 1(1), pp. 53-66.

Eglese R.W. 1994. Routing winter gritting vehicles, DAM, 48(3), pp. 231-244.

Fleury G. 1993. Méthodes stochastiques et déterministes pour les problèmes NP-difficiles. Thèse de l'Université Blaise Pascal.

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

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

Gutjahr W.J. 2000. A graph-based Ant System and its convergence, Future Generation Computing Systems, 16, pp. 873-888.

Hajek B. 1988. Cooling schedules for optimal annealing, Maths Oper. Res., 13 (2), pp. 311-329.

Hertz A., G. Laporte et M. Mittaz. 2000. A tabu search heuristic for the Capacitated Arc Routing Problem, Operations Research, 48(1), pp. 129-135.

Lacomme P., C. Prins et W. Ramdane-Chérif. 2001. Competitive genetic algorithms for the Capacitated Arc Routing Problem and its extensions, dans E.J.W. Boers *et al.* (éd.), Applications of evolutionary computing, pp. 473-483, Lecture Notes in Computer Science 2037, Springer.

Reimann M., M. Stummer et K. Doerner. 2002. A savings based Ant System for the Vehicle Routing Problem, dans Langdon *et al*. (éd.): *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002)*, Morgan Kaufmann, New York, pp. 1317-1325.

Ulusoy G. 1985. The fleet size and mix problem for Capacitated Arc Routing, European Journal of Operational Research, 22, pp. 329-337.