A Genetic Algorithm for the Capacitated Arc Routing Problem and Its Extensions

Philippe Lacomme, Christian Prins, and Wahiba Ramdane-Chérif

University of Technology of Troyes, Laboratory for Industrial Systems Optimization
12, Rue Marie Curie, BP 2060, F-10010 Troyes Cedex (France)
{Lacomme, Prins, Ramdane}@univ-troyes.fr

Abstract. The NP-hard Capacitated Arc Routing Problem (CARP) allows to model urban waste collection or road gritting, for instance. Exact algorithms are still limited to small problems and metaheuristics are required for large scale instances. The paper presents the first genetic algorithm (GA) published for the CARP. This hybrid GA tackles realistic extensions like mixed graphs or prohibited turns. It displays excellent results and outperforms the best metaheuristics published when applied to two standard sets of benchmarks: the average deviations to lower bounds are 0.24 % and 0.69 % respectively, a majority of instances are solved to optimality, and eight best known solutions are improved.

1 Introduction

The *Capacitated Arc Routing Problem* (*CARP*) is defined in the literature on an undirected network $G = (V,E)$ with a set V of n nodes and a set E of m edges. A fleet of identical vehicles of capacity Q is based at a depot node s. A subset R of edges require service by a vehicle. All edges can be traversed any number of times. Each edge (i, j) has a traversal cost $c_{ij} \geq 0$ and a demand $r_{ij} \geq 0$. The CARP consists of determining a set of vehicle trips of minimum total cost, such that each trip starts and ends at the depot, each required edge is serviced by one single trip, and the total demand handled by any vehicle does not exceed Q. The cost of a trip comprises the costs of its serviced edges and of its intermediate connecting paths.

Many applications occur in road networks: urban waste collection, snow plowing, sweeping, gritting, etc. Demands are amounts to be collected along the streets (urban waste) or delivered (salt for ice clearance). Costs are distances or travel times. The undirected version concern roads that can be serviced during one traversal and in any direction, which happens in low-traffic suburban areas. In the directed version of the CARP, each arc represents one street (or one side of street) with a mandatory direction for service. Both versions are NP-hard, even in the special case where one trip is sufficient (*Rural Postman Problem*).

E.J.W. Boers et al. (Eds.) EvoWorkshop 2001, LNCS 2037, pp. 473-483, 2001.

Golden and Wong [1], Benavent *et al.* [2] and Belenguer and Benavent [3] have investigated integer linear programming formulations and lower bounds for the CARP. Since exact algorithms like the branch-and-bound method of Hirabayashi *et al.* [4] are still limited to small instances (30 edges), larger instances must be tackled in practice by heuristics. Good greedy heuristics include Path-Scanning from Golden and Wong [1], Construct-Strike as improved by Pearn [5], Augment-Insert from Pearn [6], Augment-Merge from Golden *et al*. [7] and Ulusoy's tour splitting method [8]. Concerning metaheuristics, Li [9] applied simulated annealing and tabu search to a road gritting problem, and Eglese [10] designed a simulated annealing approach for a multi-depot gritting problem with side constraints. The most efficient metaheuristic published so far is a sophisticated tabu search method from Hertz *et al.* [11].

The context of municipal waste collection is adopted in the following, for making the problem more concrete and talkative. We first present in section 2 our extension of the basic CARP with its data structures. Section 3 describes the basic components of our GA, which is tested in section 4 on two standard sets of benchmarks.

2 Extended CARP and Data Structures

Compared with the basic CARP in introduction, we already tackle three extensions. First, we add a set U of directed arcs to get a mixed graph $G = (V,U,E)$ combining edges and arcs. m becomes the number of *links* (arcs or edges). We call *tasks* the t links in R (usually defined by a non-zero amount of waste). Second, we define for each task (i, j) a collecting cost $w_{ij} \geq 0$ distinct from the true traversal cost c_{ij} (without collecting). Third, we handle prohibited turns, *e.g.* undesirable U-turns.

In memory, G is converted into an entirely directed internal graph $H = (V, A)$. A contains $nia = |U|+2.|E|+1$ *internal arcs*: one per arc of U, two opposite arcs per edge of E, and one dummy loop on the depot. We drop the nodes to use arc indexes only. Each internal arc u is defined by a demand $r(u)$, a traversal cost $c(u)$, a collecting cost $w(u)$ if u is required, a pointer $inv(u)$ to the inverse arc when u codes one edge, and a list $Succ(u)$ of successor arcs to which it is allowed to turn. Turn prohibitions become transparent with this data structure. Shortest path costs are pre-computed in a matrix D, $nia \times nia$. For any pair of arcs (u,v), $D(u,v)$ is the traversal cost of a shortest path from u to v (*not included* to ease arc insertions or deletions in trips), taking turn prohibitions into account. A modified Dijsktra's algorithm with arc labels can compute D in $O(m^3)$, or in $O(m^2.\log m)$ with a heap structure. Since $m \approx 4n$ in real road networks, these complexities reduce to $O(n^3)$ and $O(n^2.\log n)$ in practice.

In any solution, each trip is stored as a sequence of tasks (internal arc numbers, in the order of service) with a total cost and a total load. Shortest paths between tasks are assumed. They are not stored, since their cost can be read in $O(1)$ from D. The cost of a trip cumulates the collecting costs of its tasks and the traversal costs of its intermediate paths. The total cost of a solution is the sum of all trip costs.

3 Genetic Algorithms for the Extended CARP

This section describes the basic components of the hybrid GA: chromosomes and fitness, population, reproduction step (selection of parents and crossover), mutation by local search, replacement method, and stopping criteria.

3.1 Chromosomes and Fitness

Most GAs for the travelling salesman problem (TSP) use *permutation chromosomes*. For the CARP, such a chromosome could be viewed as the order in which a vehicle must collect all the t tasks, assuming the same vehicle performs all trips in turn. This encoding is appealing because *there always exists one optimal sequence*. However, a given sequence may be split into trips in many ways, and trip delimiters seem inevitable to avoid ambiguous chromosomes. This raises several problems, for instance defining trip limits in the children generated by a crossover.

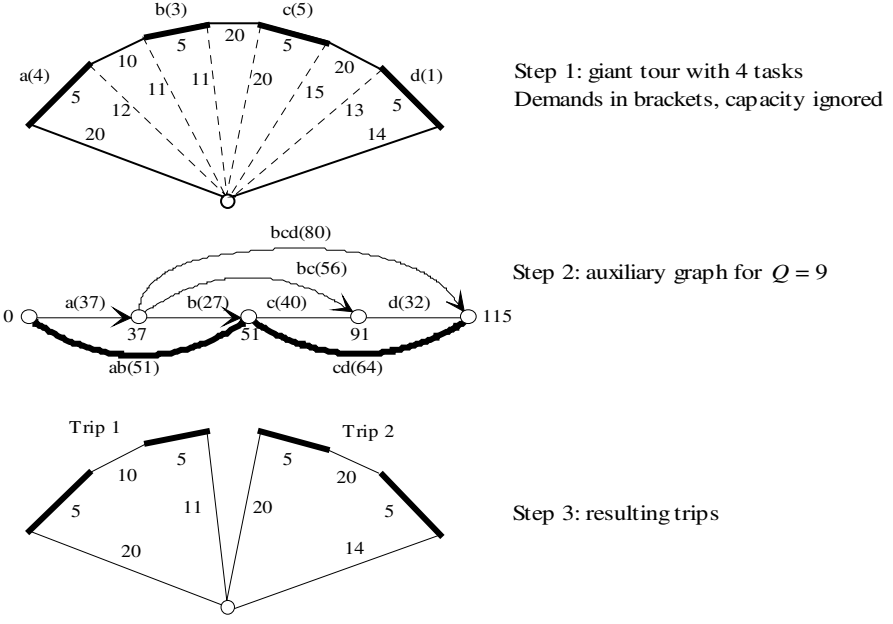

Fig. 1. Main steps of Ulusoy's algorithm

Ulusoy's algorithm provides an elegant solution for keeping simple permutation chromosomes. Usually, this algorithm is a CARP heuristic better explained by figure 1. First, capacity is ignored to build one giant tour T covering all tasks (a, b, c and d in the figure). Second, an auxiliary graph is built in which each arc denotes a subsequence of T that can be done by one trip. Each arc is weighted by the cost of this trip. A shortest path in this graph shows where to split T into trips and gives the cost of

the corresponding solution. Third, the solution is built with one trip per arc of the path. This algorithm is heuristic, because the giant tour giving an optimal CARP solution is difficult to obtain. However, *it splits any given tour optimally*. Our GA use step 2 to evaluate the chromosomes, considered as giant tours. The trips are never expressed, except to output solutions when the GA stops. The fitness is simply the total cost of the underlying CARP solution, as returned by step 2.

3.2 Population Structure and Initial Population

The population is an array P of nc chromosomes, *sorted in decreasing cost order*. It is initialized with random solutions and three good heuristic solutions. Random ones are computed by a sequential heuristic that randomly selects at each iteration one task not yet collected, plus its collecting direction if it is an edge. The current trip ends when one more task would violate truck capacity. The good solutions are given by Ulusoy's heuristic (see 3.1), Path-Scanning and Augment-Merge. All solutions are converted into chromosomes by concatenating the lists of tasks of their trips and evaluated by step 2 of Ulusoy's algorithm. This significantly improves random solutions but seldom the good heuristic solutions.

Path-Scanning builds trips one by one, starting from the depot, and extends each trip task per task. The extension step at a node i considers two cases: if no free task is incident to i, the trip moves to the nearest node with free tasks incident to it, else the next task (i, j) to be collected is selected using five criteria: 1) maximize distance from j to the depot, 2) minimize this distance, 3) maximize waste density r_{ij} / c_{ij}, 4) minimize waste density and 5) use rule 1 if truck load $\leq Q / 2$, else use rule 2. Five solutions are computed (one for each criterion); the final result is the best solution.

Augment-Merge builds one trip per task and sorts these t trips in decreasing cost order. An *Augment phase* compares each trip T_i, i=1,2,...,t-1, with each smaller trip T_j, j=i+1,i+2,...,t. If T_i traverses the unique task u of T_j, can collect u, and if this improves total cost, then T_i collects u and T_j is discarded. In the *Merge phase*, pairs of trips are merged in descending order of the saving produced, subject to the capacity constraints. This process stops when no further positive saving remains.

Clones (identical solutions) are forbidden in the population, to have a better dispersion of solutions and to diminish the risk of premature convergence. As clone detection is time consuming, we adopt a more restrictive but faster policy in which all solutions must have distinct costs. When building the initial population, we then check that each inserted solution has a new cost. We try up to mnt times (50 for instance) to generate each random solution. If all attempts generate a duplicate cost, the population P is truncated, but this occurs only when nc is too large.

3.3 Reproduction Step and Extended OX Crossover

Parents are chosen by *binary tournament selection*. We first randomly select two solutions from the population. We then select from these two the least cost solution to be the first parent P1. This procedure is repeated to get the second parent P2. The two parents undergo an extended version of the classical *order crossover* (OX). The reproduction step ends by randomly keeping only one child C and by discarding the other. This policy works slightly better than keeping two children or the best one.

For two parents P1 and P2 of length t, the classical OX crossover draws two random subscripts p and q with $1 \le p \le q \le t$. To build child C1, it copies the string P1[p]-P1[q] into C1[p]-C1[q]. Finally, it scans P2 in a circular way from q+1 (mod t) and copies each element not yet taken to fill C1 circularly, starting from q+1 (mod t) too. The roles of P1 and P2 are exchanged to get the other child C2. OX must be extended for the CARP and our data structure. Each chromosome contains all t tasks, but an edge can appear as one internal arc u or as its inverse $inv(u)$. Therefore, when copying u from a parent, we must check whether u and $inv(u)$ are not already taken. As explained in 3.1, the children are evaluated using step 2 of Ulusoy's algorithm.

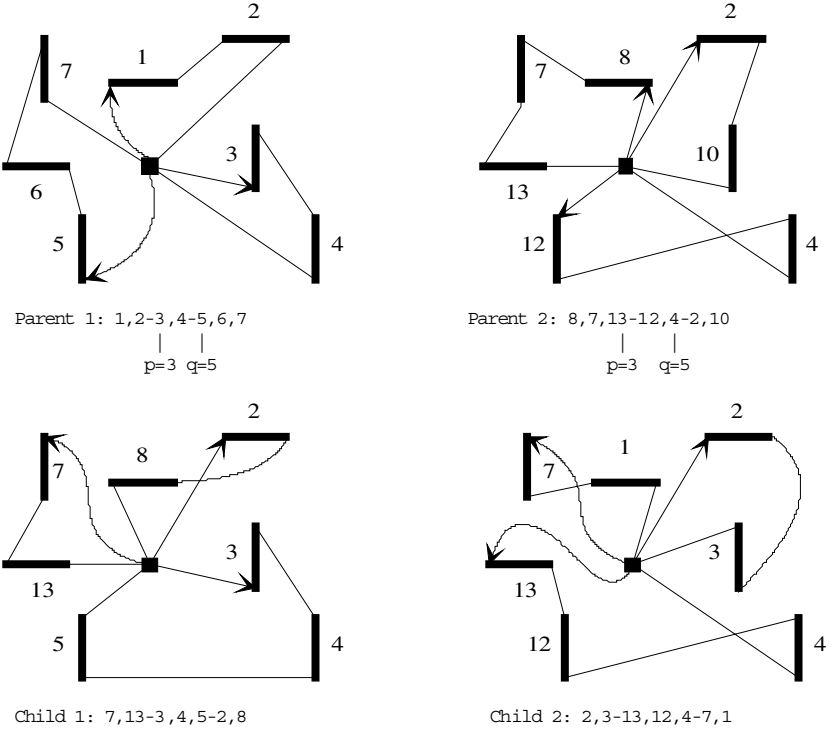

Fig. 2. An example of two parents with children generated by the OX crossover

Figure 1 gives two parent chromosomes with children obtained by the extended OX crossover, for a CARP with 7 edge-tasks. The trip limits are imaginary and result in practice from Ulusoy's algorithm, whose execution cannot be detailed here. They are shown as dashes in the chromosomes only to see the link with the graphical solutions. Recall that such limits are not stored by the GA. Thick lines are required edges. Thin lines are shortest paths in the actual network. Edges as traversed in parent 1 correspond to internal arcs 1-7. For each internal arc $u \leq 7$, the opposite arc is here $inv(u) = u + 7$, *e.g.*, the leftmost edge give internal arcs 6 and 13.

3.4 Local Search as Mutation Operator

We mutate with a fixed rate *pm* the child C produced by the crossover. The mutation operator is a *local search LS*, giving a *hybrid GA*. It is clear nowadays that hybrid GAs are better than Holland's basic GA and can even outperform other metaheuristics. Before applying LS, the child is converted into a set of trips, using steps 2-3 of Ulusoy's algorithm. Each iteration of LS scans in $O(t^2)$ all possible ways of moving a task (in one trip or to another trip) and all possible permutations of two tasks (in one trip or between two distinct trips). The collecting direction of an edge-task u may be inverted during this process, *i.e.* we try to reinsert u or $inv(u)$. We also inspect two kinds of *2-opt moves* by removing two shortest paths (possibly empty) u-v and x-y, in the same trip or in distinct trips (u, v, x and y denote a task or the depot loop). We replace them by shortest paths u-y and v-x or by paths u-x and v-y. 2-opt moves are not always possible in mixed networks because they may invert the trip order.

At each iteration, the first improving move is executed. The process is repeated until no further saving can be found. Some trips may become empty and are removed at the end. The child C is kept. A new chromosome S is rebuilt from the final trips (by concatenating their tasks) and re-evaluated with step 2 of Ulusoy's algorithm. Quite often, this slightly improves LS by shifting some trip limits. In [11], Hertz use sophisticated improvement procedures for the basic CARP, like one called *Shorten*. Testing shows that adding such procedures in LS has no significant effect on the overall GA. We took simpler neighborhoods for speed and also because they remain valid for our extended CARP model.

3.5 Replacement Method and Stopping Criteria

The replacement is *incremental*. One chromosome $P(k)$ is randomly selected below the median: $k \leq int(nc/2)$. After a mutation by LS, we try to replace $P(k)$ by S if no clone is created. In case of clone, or when LS is not applied, we replace $P(k)$ by C, avoiding cost duplication. If this fails too, the current GA iteration is said *unproductive* and discarded. This method keeps the best solution $P(nc)$ and allows a child to reproduce immediately. The GA stops after a maximum number of iterations

mni, after a maximal number of unproductive iterations *mnui*, or when it reaches a lower bound *LB* known for some instances (in that case *P*(*nc*) is optimal).

3.6 Summary – General Structure

Before starting the algorithm, we fix *nc*, *mnt*, *mni, mnui* and the mutation rate *pm*. A chromosome is encoded in *P* as a record with two fields *Seq* (sequence of tasks) and *Cost* (fitness). This gives the following general structure:

```
//Initial population
Put in P the solutions of Path-Scanning, Augment-Merge and Ulusoy
k := 3
Repeat
  k := k + 1
  Randomly generate P(k) with a new cost (mnt attempts)
Until (k = nc) or (failure)
If failure then nc := k - 1
Sort P in decreasing cost order
//Main iteration
ni, nui := 0
Repeat
   Select P1 and P2 by binary tournament
   Apply extended OX crossover and select one child C
   Evaluate C using Ulusoy's algorithm
   Choose k randomly in [1,int(nc/2)]
   If Random < pm then
      Mutate C by local search giving S
      If S.Cost not in P\{P(k)} then C := S
   EndIf
   If C.Cost not in P\{P(k)} then //Productive iteration
      ni := ni + 1
      If S.Cost < P(nc).Cost then nui := 0 Else nui := nui + 1
      P(k) := S
      Shift P(k) to resort P
   EndIf
Until (ni = mni) or (nui = mnui) or (P(nc).Cost = LB).
```

4 Computational Evaluation

The GA is implemented in Delphi on a 500 MHz PC under Windows 95 and compared with the best method for the CARP, the tabu search Carpet from Hertz *et al.* [11]. These tests are done on two standard sets of undirected instances in which all edges are required. The first set (table 1) contains 23 instances from DeArmon [12] with 7 to 27 nodes and 11 to 55 edges. The second set (table 2) contains 34 harder instances from Belenguer and Benavent [3], with 24 to 50 nodes and 34 to 97 edges. All these files can be obtained at *ftp://matheron.estadi.uv.es/pub/CARP*.

In both tables, *Prob* gives the instance number and *n*,*m* the numbers of nodes and of edges. *LB* is a lower bound from Belenguer *et al.* [3]. *TS* is the result of Carpet with the parameter setting yielding the best results on average (same setting for all

instances). *Best* gives the best solution published, generally obtained by Carpet with various settings of parameters. *GA* is the solution of our hybrid GA, with a unique parameter setting to ease comparison with Carpet. *Dev* is the deviation of GA to Carpet in %. *New best* is the new best solution when running the GA with several settings. Boldface indicates instances for which the GA improves Carpet. New best solutions are in boldface italics. Asterisks denote new optimal solutions. Our results are obtained with a small population of *nc* = 30 solutions. The local search is applied with a rate *pm* = 0.1. The GA performs a first phase stopping after *mni* = 20000 productive crossovers, *mnui* = 6000 crossovers without improving the best solution, or when *LB* is reached. If *LB* is not reached, it restarts for 10 short phases with *mni* = *mnui* = 2000 and *pm* pushed up to 0.2. *nc* is small to avoid losing too much time in unproductive crossovers (the rejection rate is 10 to 20% for *nc* = 30).

Table 1. Results of the GA on De Armon's instances

Prob	n,m	LB	Best	TS	GA	Var %	New best
1	12,22	316	316	316	316	0.00	316
2	12,26	339	339	339	339	0.00	339
3	12,22	275	275	275	275	0.00	275
4	11,19	287	287	287	287	0.00	287
5	13,26	377	377	377	377	0.00	377
6	12,22	298	298	298	298	0.00	298
7	12,22	325	325	325	325	0.00	325
8	27,46	344	348	352	**350**	**-0.57**	348
9	27,51	303	311	317	***303****	***-4.42***	***303****
10	12,25	275	275	275	275	0.00	275
11	22,45	395	395	395	395	0.00	395
12	13,23	448	458	458	458	0.00	458
13	10,28	536	544	544	***540***	***-0.74***	***538***
14	7,21	100	100	100	100	0.00	100
15	7,21	58	58	58	58	0.00	58
16	8,28	127	127	127	127	0.00	127
17	8,28	91	91	91	91	0.00	91
18	9,36	164	164	164	164	0.00	164
19	11,11	55	55	55	55	0.00	55
20	11,22	121	121	121	121	0.00	121
21	11,33	156	156	156	156	0.00	156
22	11,44	200	200	200	200	0.00	200
23	11,55	233	233	235	235	0.00	235

The hybrid GA is *very* efficient: on all instances, it is at least as good as Carpet. On the 23 DeArmon's instances, it outperforms Carpet 3 times, improves 2 best known solutions with one to optimality, and reaches *LB* 19 times. The average and worst deviations to *LB* are roughly divided by 2 compared to Carpet and become respectively 0.24 % and 2.23 %. For the 34 harder instances from Belenguer *et al.*, Carpet is improved upon 16 times and the best solution 5 times, with one new optimal solution. Optimality is proven on 23 instances. Again, the average and worst

deviations to *LB* are halved and become 0.69 % and 4.26 %. *Table 3* summarizes this comparison with Carpet and gives some other indicators. CPU times are difficult to compare: Carpet was tested by Hertz on a Silicon Graphics Indigo 2 at 195 MHz, while we use a Pentium III PC at 500 MHz.

Table 2. Results of the GA on the instances from Belenguer and Benavent

Prob	n,m	LB	Best	TS	GA	Var %	New best
1A	24,39	173	173	173	173	0.00	173
1B	24,39	173	173	173	173	0.00	173
1C	24,39	235	245	245	245	0.00	245
2A	24,34	227	227	227	227	0.00	227
2B	24,34	259	259	260	**259**	**-0.38**	259
2C	24,34	455	457	494	**462**	**-6.48**	457
3A	24,35	81	81	81	81	0.00	81
3B	24,35	87	87	87	87	0.00	87
3C	24,35	137	138	138	138	0.00	138
4A	41,69	400	400	400	400	0.00	400
4B	41,69	412	412	416	**412**	**-0.96**	412
4C	41,69	428	430	453	***428****	***-5.52***	***428****
4D	41,69	520	546	556	***541***	***-0.92***	***530***
5A	34,65	423	423	423	423	0.00	423
5B	34,65	446	446	448	**446**	**-0.45**	446
5C	34,65	469	474	476	**474**	**-0.42**	474
5D	34,65	571	593	607	***581***	***-4.28***	***581***
6A	31,50	223	223	223	223	0.00	223
6B	31,50	231	233	241	**233**	**-3.32**	233
6C	31,50	311	317	329	**317**	**-3.65**	317
7A	40,66	279	279	279	279	0.00	279
7B	40,66	283	283	283	283	0.00	283
7C	40,66	333	334	343	**334**	**-2.62**	334
8A	30,63	386	386	386	386	0.00	386
8B	30,63	395	395	401	**395**	**-1.50**	395
8C	30,63	517	528	533	533	0.00	***527***
9A	50,92	323	323	323	323	0.00	323
9B	50,92	326	326	329	**326**	**-0.91**	326
9C	50,92	332	332	332	332	0.00	332
9D	50,92	382	399	409	***391***	***-4.40***	***391***
10A	50,97	428	428	428	428	0.00	428
10B	50,97	436	436	436	436	0.00	436
10C	50,97	446	446	451	**446**	**-1.11**	446
10D	50,97	524	536	544	***535***	***-1.65***	***530***

6 Conclusion

This paper presents the first GA ever published for the CARP. On two standard sets of benchmarks, it improves eight best known solutions and its average deviation to the lower bound is very small, less than 0.7 %. On all tested instances, despite simple

neighborhoods in its local search, this hybrid GA is as least as good as the best algorithm published, a powerful tabu search. This shows that, contrary to a still widespread opinion, hybrid GAs can outperform other metaheuristics. Moreover, thanks to our data structure, our GA can already handle an extended CARP, with a mixed graph, collecting and traversal costs, and prohibited turns or turn penalties. Our goal is now to design large instances of that type and to investigate new useful extensions, like heterogeneous fleets, multitrips, selective collection of several kinds of waste, several dumping sites, periodic problems over several days or weeks, sectors with different priorities, and time windows on the arcs.

Table 3. Overall comparison between Carpet and the GA on the two data sets

	DeArmon instances		Belenguer's instances	
Algorithm	Carpet	GA	Carpet	GA
Avg. dev. to LB%	0.5	0.24	1.9	0.69
Worst dev. to LB %	4.62	2.23	8.57	4.26
Avg. dev. to Best %	0.2	-0.08	1.1	-0.10
Proven optima	18	19	16	23
Solutions < Carpet	0	3	0	16
Solutions = Carpet	23	20	34	18
Solutions > Carpet	0	0	0	0
Solutions < Best	0	2	0	5
Solutions = Best	20	19	17	27
Solutions > Best	3	2	17	2
Avg. CPU time (s)	49	21	346	120

References

1. B.L. Golden, R.T.Wong, Capacitated arc routing problems, *Networks*, 11, 305-315, 1981.
2. E. Benavent, V. Campos, A. Corberan, E. Mota, The capacitated arc routing problem: lower bounds, *Networks*, 22, 669-690, 1992.
3. J.M. Belenguer, E. Benavent, *A cutting plane algorithm for the capacitated arc routing problem*, Research Report, Dept. of Statistics and OR, Univ. of Valencia (Spain), 1997.
4. R. Hirabayashi, Y. Saruwatari, N. Nishida, Tour construction algorithm for the capacitated arc routing problem, *Asia-Pacific Journal of Oper. Res.*, 9, 155-175, 1992.
5. W.L. Pearn, Approximate solutions for the capacitated arc routing problem, *Computers and Operations Research*, 16(6), 589-600, 1989.
6. W.L. Pearn, Augment-insert algorithms for the capacitated arc routing problem, *Computers and Operations Research*, 18(2), 189-198, 1991.
7. B.L. Golden, J.S DeArmon, E.K. Baker, Computational experiments with algorithms for a class of routing problems, *Computers and Operation Research*, 10(1), 47-59, 1983.
8. G. Ulusoy, The Fleet Size and Mixed Problem for Capacitated Arc Routing, *European Journal of Operational Research*, 22, 329-337, 1985.
9. L.Y.O. Li, Vehicle routing for winter gritting. Ph.D. dissertation, Department of OR and OM, Lancaster University, Lancaster, UK, 1992.

10. R.W. Eglese, Routing winter gritting vehicles, *DAM*, 48(3), 231-244, 1994.
11. A. Hertz, G. Laporte, M. Mittaz, A Tabu Search Heuristic for the Capacitated Arc Routing Problem, *Operations Research*, 48(1), 129-135, 2000.
12. J.S. DeArmon, *A Comparison of Heuristics for the Capacitated Chinese Postman Problem*, Master's Thesis, The University of Maryland at College Park, MD, USA, 1981.