A TABU SEARCH HEURISTIC FOR THE CAPACITATED ARC ROUTING PROBLEM

ALAIN HERTZ

Département de Mathématiques, École Polytechnique Fédérale de Lausanne, CH-1015 Lausanne, Switzerland

GILBERT LAPORTE

École des Hautes Etudes Commerciales de Montreal, 3000 chemin de la Côte-Sainte-Catherine, Montréal, Canada H3T 2A7
gilbert@crt.umontreal.ca

MICHEL MITTAZ

Département de Mathématiques, École Polytechnique Fédérale de Lausanne, CH-1015 Lausanne, Switzerland

(Received January 1997; revisions received December 1997, June 1998; accepted July 1998)

The Capacitated Arc Routing Problem arises in several contexts where streets or roads must be traversed for maintenance purposes or for the delivery of services. A tabu search is proposed for this difficult problem. On benchmark instances, it outperforms all known heuristics and often produces a proven optimum.

The purpose of this article is to describe an efficient search heuristic for the undirected Capacitated Arc Routing Problem (CARP) described as follows. Let $G=(V,E)$ be an undirected graph, where $V=\{v_0,v_1,\ldots,v_n\}$ is a vertex set and $E=\{(v_i,v_j):v_i,v_j\in V, i<j\}$ is an edge set. Vertex v_0 represents a depot at which are based m identical vehicles of capacity Q. The number of vehicles can be a decision variable, as in this article, or a fixed parameter. A subset R of edges are said to be *required*, i.e., they must be *serviced* by a vehicle. Any edge of E can be *traversed* any number of times. Each edge (v_i,v_j) of E has a nonnegative cost or length c_{ij}, and in addition, each edge of R has a nonnegative weight or demand q_{ij}. The CARP consists of designing a set of vehicle routes of least total cost, such that each route starts and ends at the depot, each required edge appears in at least one route and is serviced by exactly one vehicle, and the total weight of all edges serviced by any vehicle does not exceed Q. The CARP is NP-hard because it includes as a special case the Rural Postman Problem (RPP) shown to be NP-hard by Lenstra and Rinnooy Kan (1976). Applications of the CARP arise naturally in several contexts where streets or roads must be traversed for the purpose of garbage collection, snow removal, sweeping, gritting, mail delivery, meter reading, school bus routing, etc. (Eiselt et al. 1995).

The undirected CARP can be formulated as an integer linear program (Golden and Wong 1981, Belenguer 1990, Belenguer and Benavent 1991) and can be solved by branch-and-cut. A branch-and-bound procedure was also developed by Hirabayashi et al. (1992), but only relatively small instances can be solved using these approaches. Various lower bounds have been developed by Golden and Wong (1981), Assad et al. (1987), Win (1987), Pearn (1988), Benavent et al. (1992), and Belenguer and Benavent (1997). Heuristics have been proposed by Christofides (1973), Golden et al. (1983). Chapleau et al. (1984), Ulusoy (1985), Win (1987), and Pearn (1989, 1991). The best methods appear to be the modified construct-strike approach (Pearn 1989) for dense graphs and the augment-insert heuristic (Pearn 1991) for sparse graphs. In recent years, several metaheuristics have proved highly efficient for the solution of combinatorial optimization problems. In this spirit, Li (1992) has applied with limited success simulated annealing and tabu search methods to a road gritting problem, and Eglese (1994) has developed a simulated annealing approach for a multidepot gritting problem containing several side constraints. Greistorfer (1994) describes a tabu search algorithm for a version of the CARP with $R=E$ but presents very limited computational results.

Our aim is to develop efficient procedures and to embed these within a tabu search algorithm for the CARP. This algorithm uses a neighbourhood structure similar to that of TABUROUTE, a tabu search heuristic for the Vehicle Routing Problem (VRP) (Gendreau et al. 1994) and some of the routines developed in an undirected RPP context (Hertz et al. 1996). We have also developed some new procedures. The remainder of this paper is organized as follows. In §1 we present a number of basic procedures used in the search process. We then proceed in §2 with the description of the tabu search algorithm, called CARPET. Computational results are presented in §3.

1. BASIC PROCEDURES

Of the seven basic procedures presented in this section, the first three have already been described in detail in Hertz

Subject classifications: Transportation: arc routing problems.
Area of review: TRANSPORTATION.

et al. (1999) in the context of the RPP. Each of these three procedures applies to one vehicle route at a time. We will therefore summarize them and refer the reader to the original reference for technical details. The last four procedures are called PASTE, CUT, SWITCH, and POSTOPT and are new. CUT and SWITCH consider one route at a time, while PASTE and POSTOPT operate on several routes. These procedures will be described at length. Each procedure is specified by a vector (Input; Output).

In what follows, given a vehicle route $S = (v_0 = v_{i_1}, \dots, v_{i_r}, \dots, v_{i_t} = v_0)$, all vertices v_{i_r} and all edges $(v_{i_r}, v_{i_{r+1}})$ are said to *appear* in S. We always allow for the case where some of the intermediate vertices $v_{i_2}, \dots, v_{i_{t-1}}$ also correspond to the depot. Given an edge set $R' \subseteq R$, a solution x is said to be *feasible* for R' if each edge of R' appears in at least one route of x, is serviced by exactly one vehicle, and the total weight of all edges serviced by any vehicle does not exceed Q. If a solution is feasible for R, then it is simply called feasible. Some of these procedures implicitly assume that the length of a shortest chain has previously been computed for every vertex pair.

1.1. Procedure SHORTEN($S; S'$)

Given a circular route $S = (v_0 = v_{i_1}, \dots, v_{i_r}, \dots, v_{i_t} = v_0)$ servicing a subset R' of required edges, this procedure attempts to identify a shorter route S' including v_0 and servicing the same required edges, but not necessarily in the same order. We introduce an artificial depot $v_{0'}$ connected only to v_0 and an additional required edge $(v_0, v_{0'})$ (with zero cost and zero demand) in R' so that the depot necessarily appears in S'. Starting with $r = 1$, SHORTEN considers vertex v_{i_r} as a starting point for the route. It then attempts to construct an equivalent route S' by moving the edges of R' as far as possible in the route. If v_{i_s} is the beginning of the first edge of R' in S', then the chain $(v_{i_r}, \dots, v_{i_s})$ in S' is replaced by a minimum length chain in G. If a cost improvement is obtained, all vertices of the shortened route are relabeled from $v_{i_1} = v_0$ to $v_{i_t} = v_0$ and r is reset equal to 1. If no improvement has been obtained, r is incremented by 1. The process is then repeated as long as $r < t$.

1.2. Procedure DROP($S, (v_j, v_k) S'$)

Given a route S servicing a subset R' of required edges, and given an edge $(v_j, v_k) \in R'$, this procedure constructs a route servicing $R' \backslash \{(v_j, v_k)\}$. It does so by first changing the status of (v_j, v_k) from required to nonrequired and then applying SHORTEN($S; S'$).

1.3. Procedure ADD($S, (v_j, v_k) S'$)

Given a route S servicing a subset R' of required edges and an edge $(v_j, v_k) \notin R'$, this procedure constructs a route S' servicing $R' \cup \{(v_j, v_k)\}$. It first includes (v_j, v_k) into the route, then changes the status of (v_j, v_k) from nonrequired to required, and finally calls SHORTEN($S; S'$). To include (v_j, v_k), three cases are possible. If (v_j, v_k) already appears in S, no action is required. If neither v_j nor v_k appears in S, identify a vertex v_{i_r} of S yielding $\min_r\{c_{i_r j} + c_{i_r k}\}$ and replace $S = (v_{i_1}, \dots, v_{i_r}, \dots, v_{i_t})$ by $S = (v_{i_1}, \dots, v_{i_r}, v_j, v_k, v_{i_r}, \dots, v_{i_t})$. If only one of v_j or v_k (say, v_j), or both of those vertices appear in S, but not consecutively, replace $S = (v_{i_1}, \dots, v_j, \dots, v_{i_t})$ by $S = (v_{i_1}, \dots, v_j, v_k, v_j, \dots, v_{i_t})$.

1.4. Procedure PASTE($x; S$)

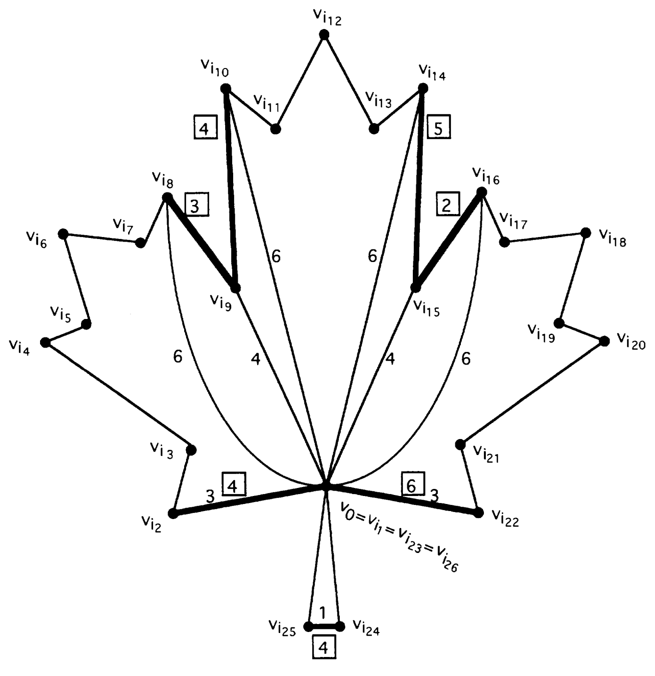
Given a solution x made up of m routes $S_1 = (v_0, \dots, v_{i_{t_1}} = v_0), \dots, S_m = (v_0, \dots, v_{i_{t_m}} = v_0)$, this procedure first creates the single route $S' = (v_0, \dots, v_{i_{t_1}} = v_0, \dots, v_{i_{t_2}} = v_0, \dots, v_{i_{t_m}} = v_0)$, whose total weight may exceed the vehicle capacity. It then calls SHORTEN($S'; S$).

1.5. Procedure CUT($S, R'; x$)

Given a nonempty subset R' of R and a route $S = (v_0 = v_{i_1}, \dots, v_{i_r}, \dots, v_{i_t} = v_0)$ in which all edges of R' appear, but possibly violating vehicle capacity, this procedure produces a solution x feasible for R'. It does so by first determining the smallest index r $(r > 1)$ such that $v_{i_r} = v_0$, and by setting d equal to the total weight of the chain $(v_{i_1}, \dots, v_{i_r})$. It then identifies a vertex v_{i_p} appearing in S such that the total weight of the chain $(v_{i_1}, \dots v_{i_p})$ does not exceed Q, and the total weight of the chain $(v_{i_p}, \dots, v_{i_r})$ does not exceed Q $(\lceil d/Q \rceil - 1)$. The value of p lies in some interval $[\underline{p}, \bar{p}]$. Vertex v_{i_p} is chosen so as to make good use of the vehicle capacity, but also taking into account the length of the shortest chain between v_{i_p} and the depot. Once v_{i_p} has been determined, the vehicle returns to the depot and collects, in a greedy fashion, the weights associated with the unserviced edges of R' along its return trip, as long as its capacity is not exceeded. The first unserviced edge $(v_{i_h}, v_{i_{h+1}})$ $(h \geqslant p)$ appearing after v_{i_p} in S is then identified and the procedure is reapplied by replacing in S the chain $(v_0 = v_{i_1}, \dots, v_{i_h})$ by a shortest chain between v_0 and v_{i_h}. The detailed implementation of CUT is provided in Hertz et al. (1997). Although the technical details are rather terse, the procedure can easily be understood on a small example.

Consider the route $S := (v_0 = v_{i_1}, \dots, v_{i_{23}} = v_0, v_{i_{24}}, \dots, v_{i_{26}} = v_0)$ depicted in Figure 1 and let $Q = 11$. The value of r is 23 because $v_{i_{23}}$ is the first occurrence of the depot after v_{i_1}. We compute $d = 4 + 3 + 4 + 5 + 2 + 6 = 24$, $\bar{p} = 10$ and $\underline{p} = 2$. Thus p can take any value in $\{2, 9, 10\}$. Let $L(s)$ be the length of a shortest chain from v_{i_s} to the depot and to the beginning of the next required edge in S. We compute $L(2) = 3 + 6 = 9$, $L(9) = 4 + 4 = 8$ and $L(10) = 6 + 6 = 12$, and we choose $p = 9$. A first route $S' = (v_{i_1}, \dots, v_{i_9}, v_0)$ is created, and the procedure is reapplied on $S = (v_0, v_{i_9}, \dots, v_{i_{26}} = v_0)$.

Figure 1. Route S showing edges of R' in bold lines and their demands in the squares. Remaining numbers are edge lengths.



1.6. Procedure SWITCH($S, v; S'$)

Given a route $S = (v_0 = v_{i_1}, \ldots, v_{i_t} = v_0)$ in which a vertex v appears several times, this procedure creates an equivalent route S' in which all minimal subroutes starting and ending at v are traversed in the reverse order. This is done by reversing all chains linking two consecutive copies of v in S, but that does not include v_{i_1} as an intermediate vertex. For example, if $S = (v_0, v_1, v, v_2, v_3, v, v_4, v_5, v_6, v, v_7, v_8, v_0)$, then $S' = (v_0, v_1, v, v_3, v_2, v, v_6, v_5, v_4, v, v_7, v_8, v_0)$. This procedure will help produce a better mix of solutions in CARPET.

1.7. Procedure POSTOPT($x; \tilde{x}$)

Given a solution x, this procedure attempts to identify a better solution $\tilde{x}$ by applying PASTE, SWITCH, CUT, and SHORTEN. (a) Call PASTE ($x; S$) to obtain a single route S, select a vertex v appearing more than once in S, and initialize a counter $\gamma := 0$. (b) Call SWITCH($S, v; S'$) and CUT($S', R; x''$), and apply SHORTEN to each route of x'' to obtain $\bar{x}$. If $\bar{x}$ improves upon x, then set $\tilde{x} := \bar{x}$; otherwise, set $\tilde{x} := x$. (c) If $\gamma := 100$ or if $\tilde{x}$ improves upon x or if v is the only vertex appearing more than once in S', then stop. Otherwise, increment γ by 1, select a vertex $v' \neq v$ repeating itself in S', set $S := S'$ and $v := v'$, and go to (b).

2. ALGORITHM CARPET

We first outline the general structure of CARPET, followed by a step-by-step description.

2.1. General Structure

The algorithm works with two different objective functions. The first $F_1(x)$, is simply the total length of solution x. As we allow intermediate infeasible solutions during the search process, we also define an artificial objective $F_2(x) = F_1(x) + \beta E(x)$, where $E(x)$ is the sum over all routes of x of the demand exceeding vehicle capacity, and β is a self-adjusting parameter. Initially, β is set equal to the average length of a shortest path between the depot and the extremity of all required edges. Every λ iterations, β is halved if all previous λ solutions were feasible and doubled if they were all infeasible. In our implementation, $\lambda = 5$.

An initial route S is obtained by solving a Rural Postman Problem on G by means of the heuristic suggested by Frederickson (1979), where R is the set of required edges. Feasibility is regained by calling CUT($S, R; x$). Denote by F_1^* and F_2^* the best known values of F_1 and F_2, respectively. At a general step, if F_1^* and F_2^* have not been improved for at least σ iterations, an attempt to improve the incumbent x^* is made by calling POSTOPT($x^*; \tilde{x}$). If $\tilde{x}$ does not improve upon x^*, it is disregarded and neighbours of the current solution x are generated. If $\tilde{x}$ improves upon x^*, the incumbent is updated and the current solution x is set equal to $\tilde{x}$. To define the neighbourhood $N(x)$ of a solution x, consider a route S of x, an edge (v_j, v_k) serviced in S, and another route S' containing only the depot (the vehicle servicing S' is then idle) or an edge of R with an extremity distant from v_j or v_k by at most δ, where δ is a parameter. A neighbour of x is then obtained by calling DROP($S, (v_j, v_k); \tilde{S}$), and ADD($S', (v_j, v_k); \tilde{S}'$) and by replacing S and S' by $\tilde{S}$ and $\tilde{S}'$ in x. If x is feasible and the best neighbour x' of x does not improve upon x, an attempt is made to improve x by successively removing and reinserting each required edge of x by calling DROP and ADD (unless this was done at the previous iteration). If this does not improve x, the algorithm selects x' as the next solution, even if DROP-ADD produces a better solution than x'. Parameter δ is set equal to the average length of an edge of E. Whenever an edge is removed from route S at a given iteration, it cannot be reintroduced into S for θ iterations, where θ is randomly selected in [5,10].

The search ends when F_1^* is equal to a lower bound value $\underline{F}$, or when μ—the number of consecutive iterations without improvement in F_1^* or F_2^*—reaches a value ρ. In our implementation we have used $\rho = 100$. A truncated version of the algorithm can also be run with a preset time limit. We have set the lower bound $\underline{F}$ equal to the maximum of the three lower bounds CPA, LB2$'$, and NDLB$'$ described in the literature. The first, CPA, was proposed by Belenguer and Benavent (1997) and is based on a cutting plane procedure. The second and third, LB2$'$ and NDLB$'$, are modified versions of LB2 (Benavent et al. 1992) and NDLB (Hirabayashi et al. 1992), respectively. In LB2$'$ and NDLB$'$, a lower bound on the number of vehicles required to serve a subset R of edges is computed by means of the lower bounding procedure LR proposed by Martello and

Table 1. Computational results on the DeArmon (1981) instances.

Instance Number	$\lvert V\rvert$	$\lvert E\rvert$	PS	AM	CS	MCS	MPS	F_0	CARPET (T) $T=1$	CARPET (T) $T=10$	CARPET	Best Known	$\underline{F}$	Seconds
1	12	22	**316**	326	331	323	**316**	337	323	323	**316**	316	316	17.1
2	12	26	367	367	418	345	355	345	345	345	**339**	339	339	28.0
3	12	22	289	316	313	**275**	283	289	**275**	**275**	**275**	275	275	0.4
4	11	19	320	290	350	**287**	292	330	**287**	**287**	**287**	287	287	0.5
5	13	26	417	383	475	386	401	414	406	**377**	**377**	377	377	30.3
6	12	22	316	324	356	315	319	323	315	**298**	**298**	298	298	4.6
7	12	22	357	**325**	355	**325**	**325**	**325**	**325**	**325**	**325**	325	325	0.0
10	27	46	416	356	407	366	380	398	382	358	352	348	344	330.6
11	27	51	355	339	364	346	357	350	332	328	317	311	303	292.2
12	12	25	302	302	364	**275**	281	283	283	**275**	**275**	275	275	8.4
13	22	45	424	443	501	406	424	427	417	409	**395**	395	395	12.4
14	13	23	560	573	655	645	566	560	516	506	**458**	458	448	111.8
15	10	28	592	560	560	**544**	551	564	556	548	**544**	544	536	13.1
16	7	21	102	102	112	102	**100**	104	102	**100**	**100**	100	100	2.6
17	7	21	**58**	**58**	**58**	**58**	**58**	**58**	**58**	**58**	**58**	58	58	0.0
18	8	28	131	131	149	**127**	131	131	129	**127**	**127**	127	127	9.2
19	8	28	93	**91**	**91**	**91**	93	**91**	**91**	**91**	**91**	91	91	0.0
20	9	36	168	170	174	**164**	167	174	172	**164**	**164**	164	164	1.5
21	11	11	57	63	63	63	**55**	63	57	**55**	**55**	55	55	1.1
22	11	22	125	123	125	123	123	125	125	123	**121**	121	121	51.5
23	11	33	168	158	165	**156**	163	164	158	**156**	**156**	156	156	6.1
24	11	44	207	204	204	**200**	202	206	204	202	**200**	200	200	18.3
25	11	55	241	237	237	**233**	244	239	239	237	235	233	233	186.3
Average Deviation			7.2%	5.6%	13.9%	3.9%	4.4%	6.6%	3.4%	1.4%	0.2%			
Worst Deviation			22.3%	25.1%	43.0%	40.8%	23.6%	22.3%	12.7%	10.5%	1.9%			
Number of Optima			2	3	1	12	5	3	5	13	18			
Number of Best			2	3	2	12	5	3	5	13	20			

Toth (1990) for the bin packing problem, instead of $\lceil (\sum_{(v_i,v_j)\in R} q_{ij})/Q \rceil$.

2.2. Step-by-Step Description

Step 1 (Initial solution).
Set the iteration counts $\tau := 0$ and $\mu := 0$, and set $\rho := 100$, $\sigma := 10$. Construct an initial solution x as described above. Call POSTOPT(x;x^*). Compute $F_1^* = F_2^* = F_1(x^*)$. Set $x := x^*$. Compute a lower bound $\underline{F}$ on F_1^*. Initially no edge is tabu.

Step 2 (Termination or incumbent improvement).
If $\mu = \rho$, or $F_1^* = \underline{F}$, or a time limit has been reached, then stop with the best known solution x^*. Otherwise, if $\mu \geqslant \sigma$, call POSTOPT (x^*;$\tilde{x}$). If $F_1(\tilde{x}) < F_1^*$, set $x := \tilde{x}$ and go to Step 4.

Step 3 (Next iteration).

Set $\tau := \tau + 1$. Generate all neighbours of x. Any such neighbour x'_{jk} obtained by moving edge (v_j, v_k) is kept for further consideration whenever (v_j, v_k) is not tabu. If (v_j, v_k) is tabu, then x'_{jk} is discarded unless it is feasible and $F_1(x'_{jk}) < F_1^*$ or it is infeasible and $F_2(x'_{jk}) < F_2^*$. Assign to each nondiscarded neighbour x'_{jk} a cost $F(x'_{jk})$ equal to $F_2(x'_{jk})$ if $F_2(x'_{jk}) < F_2(x)$ and to $F_2(x'_{jk}) + \Delta\sqrt{m}\eta\varphi_{jk}$ otherwise, where:

Δ:	the largest observed absolute difference between the values of F_2 observed at two successive iterations;
m:	the number of vehicles used in the solution x'_{jk};
η:	scaling factor equal to 0.01 in our implementation;
φ_{jk}:	the number of times edge (v_j, v_k) has been moved, divided by τ.

Set $x' := \text{argmin}\{F(x'_{jk})\}$ and let j^*, k^* be the two indices yielding x'. If x is feasible, $F_2(x) < F_2(x')$ and Step 3a was not executed at iteration $\tau - 1$, then go to (a); else go to (b).

(a) Successively apply DROP($S, (v_j, v_k); S'$) and ADD($S', (v_j, v_k); S$) to the routes S of x and the serviced edges (v_j, v_k) of S, until no further improvement can be obtained. Let x'' be the solution obtained at the end of this step. If $F_1(x'') < F_1(x)$, set $x := x''$ and go to Step 4.
(b) Set $x := x'$, and declare $(v_{j^*}; v_{k^*})$ tabu for θ iterations.

Step 4 (Increment updates).
If x is feasible and $F_1(x) < F_1^*$, set $F_1^* := F_1(x)$ and $x^* := x$. If $F_2(x) < F_2^*$, set $F_2^* := F_2(x)$. If F_1^* or F_2^* has been modified, reset $\mu := 0$, otherwise increment μ by 1. Go to Step 2.

3. COMPUTATIONAL RESULTS

CARPET was coded in Pascal and run on a Silicon Graphics Indigo2 machine (195 MHz, IP28 processor). It was first

Table 2. Computational results on the instances supplied by Benavent (1997).

Instance Number	$\|V\|$	$\|E\|$	$\underline{F}$	CARPET	Best Known	Seconds
1.A	24	39	173	**173**	173 a,b	0.1
1.B	24	39	173	**173**	173 a,b	50.2
1.C	24	39	235	**245**	245 a,b	506.1
2.A	24	34	227	**227**	227 a,b	0.9
2.B	24	34	259	260	259 a,b	70.7
2.C	24	34	455	494	457 a	171.6
3.A	24	35	81	**81**	81 a,b	4.2
3.B	24	35	87	**87**	87 a,b	15.1
3.C	24	35	138	**138**	138 a	225.8
4.A	41	69	400	**400**	400 a,b	153.5
4.B	41	69	412	416	412 a,b	410.1
4.C	41	69	428	453	430 a,b	379.7
4.D	41	69	520	556	546 a,b	1265.9
5.A	34	65	423	**423**	423 a,b	20.6
5.B	34	65	446	448	446 a,b	224.3
5.C	34	65	469	476	474 a,b	288.7
5.D	34	65	571	607	593 a	1214.7
6.A	31	50	223	**223**	223 a,b	21.1
6.B	31	50	231	241	233 b	146.0
6.C	31	50	311	329	317 a	461.7
7.A	40	66	279	**279**	279 a,b	35.7
7.B	40	66	283	**283**	283 a,b	0.1
7.C	40	66	333	343	334 a,b	658.2
8.A	30	63	386	**386**	386 a,b	20.8
8.B	30	63	395	401	395 a,b	441.5
8.C	30	63	517	533	528 a	798.9
9.A	50	92	323	**323**	323 a,b	154.5
9.B	50	92	326	329	326 a,b	324.6
9.C	50	92	332	**332**	332 a,b	305.9
9.D	50	92	382	409	399 a,b	1914.8
10.A	50	97	428	**428**	428 a,b	29.9
10.B	50	97	436	**436**	436 a,b	99.9
10.C	50	97	446	451	446 b	506.6
10.D	50	97	524	544	536 a	847.2
			Average Deviation	1.1%		
			Worst Deviation	8.1%		
			Number of Optima	16		
			Number of Best	17		

[a] Best run of CARPET with various parameter values.
[b] Belenguer and Benavent (1997).

tested on the 23 instances used by Pearn (1989) and fully described in DeArmon (1981). The latter reference contains 25 instances, but instances 8 and 9 were removed because they contained inconsistencies. The size of these problems ranges from 7 to 27 vertices and from 11 to 55 edges, all of which are required. In Table 1 we compare the results with a single pass of CARPET to those obtained by other methods. The various headings of Table 1 are as follows:

Instance number:	as in DeArmon (1981);
$\|V\|$:	number of vertices;
$\|E\| = \|R\|$:	number of edges;
PS:	solution value produced with the path-scanning heuristic (Golden et al. 1983);
AM:	solution value produced with the augment-merge heuristic (Golden et al. 1983);
CS:	solution value produced with the construct-strike heuristic (Christofides 1973);
MCS:	solution value produced with the modified construct-strike heuristic (Pearn 1989);
MPS:	solution value produced with the modified path-scanning heuristic (Pearn 1989);
F_0:	value of the initial solution obtained at the end of Step 1 after the call to POSTOPT;
CARPET(T):	solution value obtained by CARPET, with a total running time of T seconds;
CARPET:	solution value produced with the CARPET heuristic:
Best known:	best known solution value;

Table 3. Computational results on larger randomly generated instances.

			CARPET (60)			CARPET (600)		
Number of Vertices	Edge Density	Proportion of Required Edges	Average Deviation	Worst Deviation	Number of Proven Optima	Average Deviation	Worst Deviation	Number of Proven Optima
20	[0.1, 0.3]	[0.1, 0.3]	1.009	1.071	8	1.009	1.071	8
		[0.4, 0.6]	1.022	1.055	3	1.021	1.055	3
		[0.8, 1.0]	1.015	1.090	5	1.012	1.068	5
	[0.4, 0.6]	[0.1, 0.3]	1.029	1.110	4	1.015	1.044	4
		[0.4, 0.6]	1.025	1.113	5	1.006	1.054	8
		[0.8, 1.0]	1.013	1.079	5	1.004	1.033	7
	[0.7, 0.9]	[0.1, 0.3]	1.020	1.047	3	1.018	1.047	4
		[0.4, 0.6]	1.089	1.303	0	1.016	1.053	3
		[0.8, 1.0]	1.020	1.096	1	1.008	1.024	1
40	[0.1, 0.3]	[0.1, 0.3]	1.037	1.078	1	1.022	1.078	1
		[0.4, 0.6]	1.067	1.151	0	1.027	1.091	1
		[0.8, 1.0]	1.006	1.021	4	1.003	1.016	4
	[0.4, 0.6]	[0.1, 0.3]	1.114	1.337	0	1.027	1.173	2
		[0.4, 0.6]	1.074	1.201	0	1.013	1.058	0
		[0.8, 1.0]	1.018	1.056	0	1.006	1.017	1
	[0.7, 0.9]	[0.1, 0.3]	1.123	1.260	0	1.020	1.066	2
		[0.4, 0.6]	1.073	1.137	0	1.007	1.029	0
		[0.8, 1.0]	1.024	1.056	1	1.006	1.014	2
60	[0.1, 0.3]	[0.1, 0.3]	1.080	1.138	0	1.013	1.035	1
		[0.4, 0.6]	1.104	1.271	0	1.035	1.017	0
		[0.8, 1.0]	1.036	1.076	0	1.009	1.022	0
	[0.4, 0.6]	[0.1, 0.3]	1.122	1.245	0	1.017	1.071	0
		[0.4, 0.6]	1.078	1.139	0	1.022	1.049	0
		[0.8, 1.0]	1.015	1.030	0	1.007	1.017	0
	[0.7, 0.9]	[0.1, 0.3]	1.156	1.240	0	1.025	1.080	0
		[0.4, 0.6]	1.110	1.187	0	1.052	1.108	0
		[0.8, 1.0]	1.014	1.035	1	1.009	1.022	2
		Average results	5.5 %	13.4 %		1.6 %	5.2 %	

$\underline{F}$:	lower bound on the optimum value F^*, computed as $\max\{\text{CPA}, \text{LB2}', \text{NDLB}'\}$, (see §2.8);
Seconds:	computational time in seconds on a Silicon Graphics Indigo2 (195 MHz, IP 28 Processor);
Average deviation:	average ratio (in %) of the heuristic solution value over the best known solution value;
Worst deviation:	largest ratio (in %) of the heuristic solution value over the best known solution value;
Number of optima:	number of optimal solutions produced by the heuristic;
Number of best:	number of best known solutions produced by the heuristic (indicated by bold numbers).

These results indicate that CARPET is a highly efficient heuristic. On these 23 instances, it generates a proved optimum 18 times and a best known solution 20 times. It always produces the best known solution of the table, except for instance 25, where MCS is slightly better. It should be noted that on instances 10 and 11, we have obtained solution values of 348 and 311, using different versions of CARPET. The solutions generated with CARPET are on average only 0.17% away from the best known and this deviation is never more then 1.93%. This compares favourably with similar statistics relative to other heuristics. For example, the "worst deviation" obtained with CARPET is better than the "average deviation" obtained with any of the previous heuristics. To investigate whether CARPET can quickly produce good quality results, we have run three truncated versions of this algorithm. In the first (F_0), only Step 1 was applied (no tabu search was performed). In the other two (CARPET(1) and CARPET(10)), the computing time was limited to 1 and 10 seconds, respectively. In all cases, the "worst deviations" produced by these truncated algorithms are better than those obtained with any of the previous heuristics. The average deviation from optimality of CARPET(1) and CARPET(10) is also better than that of all heuristics used in the comparison. The numbers of optimal and best solutions obtained with CARPET(10) are also larger than for previous heuristics, and CARPET(1) is surpassed only by MCS with respect to these two criteria.

CARPET was then applied to 34 instances supplied to the authors by Benavent (1997). These instances range in size from 24 to 50 vertices, and from 34 to 97 edges, all of which are required. The solution values obtained by CARPET were then compared to best known solutions. These were obtained in 6 cases by running CARPET with a variety of parameters, and in two cases by Belenguer and Benavent (1997). In the remaining 26 cases, the best known solution is obtained by both methods. Using CARPET with the parameter values give in §2, we have obtained 17 best known solutions. The average deviation from the best known solution was equal to 1.13%, and the worst deviation was 8.10%. The average deviation from $\underline{F}$ was equal to 1.94%, and 16 proven optima were obtained. These results are shown in Table 2.

Finally, we have analyzed the performance of CARPET on 270 larger instances with $|V| = 20, 40$, and 60 generated as follows. The depot was selected as the median of these vertices. A first set of edges were generated by constructing a random Hamiltonian cycle on these vertices. Additional edges were then randomly generated until a prescribed density ε was attained, where ε was randomly selected in [0.1,0.3], [0.4,0.6], and [0.7,0.9], and required edges generated with probability ω in [0.1,0.3], [0.4,0.6], and [0.8,1.0]. The number of required edges in these graphs can be in excess of 1,500, a size representative of several real-life situations. For each of the 27 combinations of these parameters, 10 instances were generated and solved with time limits of 60 and 600 seconds. We report in Table 3 average computational results for these instances.

Results shown in Table 3 indicate that CARPET consistently produces high-quality solutions on the larger randomly generated graphs. Several proven optimal solutions are obtained for the smaller graphs, and the average percentage deviation from $\underline{F}$ varies between 0.3% and 5.2%. A part of these gaps is of course a result of the difference between the lower bound $\underline{F}$ and the global optimum value.

ACKNOWLEDGMENTS

This work was partly supported by the Canadian Natural Sciences and Engineering Council under grant OPG0039682 and by the Swiss National Scientific Research Council under grant 21-45574.95. This support is gratefully acknowledged. The authors thank Enrique Benavent, who kindly supplied the test problems of Table 2 and a computer code for the determination of the CPA lower bound, and to three anonymous referees for their valuable comments.

REFERENCES

Assad, A. A., W.-L. Pearn, B.L. Golden. 1987. The capacitated Chinese postman problem: Lower bounds and solvable cases. *Amer. J. Math. and Management Sci.* **7** 63–88.

Belenguer, J. M. 1990. El poliedro del problema de rutas por arcos con capacidades. Ph.D. Thesis, Universidad de Valencia, Spain.

——, E. Benavent. 1991. Polyhedral results on the capacitated arc routing problem. Working Paper, Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Spain.

——, ——. 1997. A cutting plane algorithm for the capacitated arc routing problem. Unpublished manuscript.

Benavent, E. 1997. ftp://indurain.estadi.uv.es/pub/CARP.

——, V. Campos, A. Corberán, E. Mota. 1992. The capacitated arc routing problem: lower bounds. *Networks* **22** 669–690.

Chapleau, L., J. A. Ferland, G. Lapalme, J.-M. Rousseau. 1984. A parallel insert method for the capacitated arc routing problem. *Oper. Res. Letters* **3** 95–99.

Christofides, N. 1973. The optimum traversal of a graph. *Omega* **1** 719–732.

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

Eglese, R. W. 1994. Routing winter gritting vehicles. *Discrete Appl. Math.* **48** 231–244.

Eiselt, H. A., M. Gendreau, G. Laporte. 1995. Arc routing problems, part II: the rural postman problem. *Oper. Res.* **43** 399–414.

Frederickson, G. N. 1979. Approximation algorithms for some postman problems. *J. ACM* **26** 538–554.

Gendreau, M., A. Hertz, G. Laporte. 1994. A tabu search heuristic for the vehicle routing problem. *Management Sci.* **40** 1276–1290.

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

——, J. S. DeArmon, E. K. Baker. 1983. Computational experiments with algorithms for a class of routing problems. *Comput. Oper. Res.* **10** 47–59.

Greistorfer, P. 1994. Computational experiments with heuristics for a capacitated arc routing problem. Working Paper 32, Department of Business, University of Graz, Graz, Austria.

Hertz, A., G. Laporte, M. Mittaz. 1997. A tabu search heuristic for the capacitated arc routing problem. Publication CRT-97-03, Centre for Research on Transportation, Montreal, Canada.

——, ——, P. Nanchen-Hugo. 1999. Improvement procedures for the undirected rural postman problem. *INFORMS J. on Computing* **11** 53–62.

Hirabayashi, R., Y. Saruwatari, N. Nishida. 1992. Tour construction algorithm for the capacitated arc routing problem. *Asia-Pacific J. Oper. Res.* **9** 155–175.

Lenstra, J. K., A. H. G. Rinnooy Kan. 1976. On general routing problems. *Networks* **6** 273–280.

Li, L. Y. O. 1992. Vehicle Routing for Winter Gritting. Ph.D. Dissertation, Department of OR & OM, Lancaster University, Lancaster, UK.

Martello, S., P. Toth. 1990. Lower bounds and reduction procedures for the bin packing problem. *Discrete Appl. Math.* **28** 59–70.

Pearn, W.-L. 1988. New lower bounds for the capacitated arc routing problem. *Networks* **18** 181–191.

——. 1989. Approximate solutions for the capacitated arc routing problem. *Comput. Oper. Res.* **16** 589–600.

——. 1991. Augment-insert algorithms for the capacitated arc routing problem. *Comput. Oper. Res.* **18** 189–198.

Ulusoy, G. 1985. The fleet size and mix problem for capacitated arc routing. *Euro. J. Oper. Res.* **22** 329–337.

Win, Z. 1987. Contribution to Routing Problems. Ph.D. Dissertation, University of Augsburg, Augsburg, Germany.

——. 1989. On the windy postman problem on Eulerian graphs. *Math. Programming* **44** 97–112.