Computers Opns Res. Vol. 16, No. 6, pp. 589–600, 1989
Printed in Great Britain.

0305-0548/89 $3.00 + 0.00

APPROXIMATE SOLUTIONS FOR THE CAPACITATED ARC ROUTING PROBLEM

WEN LEA PEARN*
AT&T Bell Laboratories, Holmdel, NJ 07733, U.S.A.

(*Received October* 1987; *revised January* 1989)

Scope and Purpose—The capacitated arc routing problem (CARP), a capacitated variation of the arc routing problems such as the Chinese postman problem (CPP), and the rural postman problem (RPP), is a very practical distribution management problem which has many real world applications. Since the CARP involves capacity constraints, many vehicle tours are required in order to serve all the demand arcs, and it is very difficult to solve the problem optimally even on some approximate, restricted versions of the problem. For this reason, many heuristic procedures have been proposed to solve the CARP approximately. The purpose of this paper is to review some of the existing heuristic algorithms and to present two modifications of the existing methods to obtain near-optimal solutions for the CARP.

Abstract—The capacitated arc routing problem (CARP), is a capacitated variation of the arc routing problems in which there is a capacity constraint associated with each vehicle. Due to the computational complexity of the problem, recent research has focussed on developing and testing heuristic algorithms which solve the CARP approximately. In this paper, we review some of the existing solution procedures, analyze their complexity, and present two modifications of the existing methods to obtain near-optimal solutions for the CARP. Extensive computational results are presented and analyzed.

INTRODUCTION

The capacitated arc routing problem (CARP), a generalization of the arc routing problems such as the Chinese postman problem (CPP), or the rural postman problem (RPP), is a very practical distribution management problem which has many real world applications.

The CARP, first introduced by Golden and Wong [1], can be briefly defined as follows: given an undirected network with nonnegative arc demands and arc costs, find a set of minimal cost cycles, each passes through the depot node which traverse all the arcs and that the total demand services by each vehicle does not exceed the vehicle capacity W. Thus, the vehicle routes must be designed so that:

- each positive-demand arc is serviced by exactly one vehicle;
- each cycle begins and ends at a distinguished node called the central depot;
- the total arc demand serviced by each vehicle does not exceed the vehicle capacity W;
- the total routing cost is minimized.

The solution to the CPP may be obtained efficiently using a polynomial-time bounded matching algorithm due to Edmonds and Johnson [2]. However, the CARP, which involves capacity constrains, is very difficult to solve exactly. In fact, Golden and Wong [1] show that even the 0.5-approximate CCPP (capacitated Chinese postman problem) which attempts to find a solution with a cost less than 1.5 times the value of the optimal solution, is NP-hard. Since the CCPP can be viewed as a particular case of the CARP, the 0.5-approximate CARP is also NP-hard.

Due to the computational complexity of the problem, recent research has focussed on developing and testing heuristic algorithms, such as Christofides [3], Stern and Dror [4], Golden *et al.* [5], Chapleau *et al.* [6] and Pearn [7], which solve the CARP approximately. In this paper, we will present two modifications of the existing algorithms to provide near-optimal solutions to the problem.

*Wen Lea Pearn received his PhD degree from University of Maryland. He was an Assistant Professor of Operations Research at California State University, Fresno, and is now with AT&T Bell Laboratories. His area of interest includes network optimization and probabilistic modeling. He has contributed articles to various journals in those areas.

Applications of the CARP include routing of school buses, street sweepers, snow plows, household refuse collection vehicles, spraying of roads with salt, inspection of electric power lines, or oil or gas pipelines, and the reading of electric meters.

REVIEW OF EXISTING ALGORITHMS

Christofides [3] presented an algorithm for the CCPP based on an optimal algorithm for the CPP which obtained near-optimal solutions to the CCPP and was referred to as the construct-strike algorithm. Golden *et al.* [5] presented two other algorithms called the path-scanning algorithm and the augment-merge algorithm. In Stern and Dror [4], an application of the CARP on routing electric meter readers problem, was considered. Since working hour time restriction was imposed in this case, open tours (instead of closed tours) may be desired, and tours may start and end at intermediate points of an arc. In the school bus routing system developed by Chapleau *et al.* [6], besides minimizing the total routing cost incurred, a secondary objective which is of providing "good quality service", was also considered. These two situations may be viewed as CARP variants and thus will not be discussed here.

Construct-strike algorithm

The basic idea behind this algorithm is to construct feasible cycles (having total arc demand no larger than the vehicle capacity W) which, when removed, do not separate the remaining graph into disconnected components. When the feasible cycles are constructed, we remove all the arcs in the cycles from the graph. This cycle construct-then-remove procedure is repeated until no more (feasible) cycles can be found. We then move to the second stage of the algorithm.

In the second stage, the minimal cost 1-matching algorithm is applied to match the odd-degree nodes in the remaining graph (with constructed cycles removed) so that an Euler cycle may be generated by adding the matching solution to the remaining graph. The algorithm then returns to the first stage and search for feasible cycles. The feasible cycle construct-then-remove procedure and the matching procedure are repeated until all the demand arcs in the graph have been covered.

Computational complexity. The bottleneck of the construct-strike algorithm is the minimal cost 1-matching segment for generating Euler cycles, and the all-pair shortest path segment for checking the connectness of the remaining graph. Each requires $O(n^3)$ computations. Since the matching procedure will be iteratively repeated for no more than m times, the complexity of this algorithm is $O(mn^3)$ where n = number of nodes, and m = number of arcs in the graph.

Path-scanning algorithm

The path-scanning procedure is based on constructing one cycle at a time based on a certain myopic optimization criterion. In forming each cycle, a path is extended by adding the arc that looks most promising until the vehicle capacity is exhausted, then the shortest return path to the depot is followed to form a complete cycle. In work by Golden *et al.* [5], five path scanning optimization criteria are considered in the arc selection procedure. These are:

- the cost $c(i, j)$, per unit, remaining demand is minimized;
- the cost $c(i, j)$, per unit, remaining demand is maximized;
- the cost from node j back to the depot is minimized;
- the cost from node j back to the depot is maximized;
- if the remaining vehicle capacity is less than half-full, maximize the cost from node j back to the depot; otherwise, minimize the cost.

Each of these criteria is used to generate a complete solution, and the final solution from this approach is the best of these five.

Computational complexity. In extending the tour path by adding demand arcs using one of the five path scanning optimization criteria, cost (length) of the shortest path between the demand arcs and the current tour path must be computed. This requires an application of the all-pair shortest path algorithm which is of $O(n^3)$. Thus, the complexity of this algorithm is $O(n^3)$.

Augment-merge algorithm

The basic procedure of this algorithm is outlined below (as appeared in the original paper):

- INITIALIZE – all demand arcs are serviced by a separate cycle;
- AUGMENT – starting with the longest cycle available, see if a demand arc on a shorter cycle can be serviced on a longer cycle;
- MERGE – subject to capacity constraints, evaluate the merging of any two cycles (possibly subject to additional restrictions). Merge the two cycles which yield the largest savings;
- ITERATE – repeat Step 3 until finished.

Several experiments and modifications of the basic augment-merge approach are considered in the work by Golden *et al.* [5]. These include minimal-cardinality shortest path, maximal-cardinality shortest path, two saving functions, and many other techniques. The number of augment-merge program executions were limited to 24. The final solution of this approach is the best of the 24 runs.

Computational complexity. In merging two cycles, cost (length) of the shortest path between the endpoints of the two cycles, must be computed so that the savings may be evaluated. This requires an application of the all-pair shortest path algorithm which is of $O(n^3)$. Therefore, the complexity of this algorithm is $O(n^3)$.

PATHOLOGICAL EXAMPLES

In the construct-strike algorithm, feasible cycles are constructed in a way so that when they are removed, they do not separate the remaining graph into disconnected components. While this algorithm works well in general, in certain instances, this restriction may cause the algorithm to perform very badly, as shown in the following example.

Example 1. Consider the network depicted in Fig. 1(a) with all arc costs $c(i, j)$ and demands $q(i, j)$ indicated as $(c(i, j), q(i, j))$. Let the vehicle capacity $W = 6$. Since the removal of the feasible cycle (1, 2, 3, 1) would separate the remaining graph into two disjoint components, and odd-degree nodes exist, one must apply the minimal cost 1-matching algorithm [which yields the artificial arc (3, 6)], so that an Euler cycle may be generated. However, since the removal of cycle (1, 2, 3, 1) will once again separate the remaining graph into disjoint components, one must add two shortest paths of artificial arcs from the depot to its nearest node 3, which allows one to construct and remove the cycle (1, 3, 6, 3, 1). The remaining graph is shown in Fig. 1(b), which has odd degree nodes 1 and 3. Proceeding with the algorithm to the remaining graph which yields an artificial arc (1, 3) and allows the feasible cycle (1, 2, 3, 1) to constructed, the remaining graph is shown in Fig. 1(c). Since there is no odd-degree node in the remaining graph, two shortest paths of artificial arcs are added to the remaining graph from the depot to its nearest node 2, so that an Euler cycle is generated, which allows the final cycle (1, 2, 4, 5, 2, 1) to be generated and the value of this heuristic solution is 2A + 2A + 2A = 6A. Compare this with the optimal solution consisting of the following cycles, (1, 2, 3, 1) and (1, 2, 4, 5, 2, 3, 6, 3, 1), having a total cost of 2A + 2A = 4A, we have (heuristic solution)/(optimal solution) = 6A/4A = 3/2. That is, the value of the heuristic solution exceeds the value of the optimal solution by 50%. (It should be noted that if the shortest paths were added to node 2 instead of node 3 in the first pass, same result will be generated.)

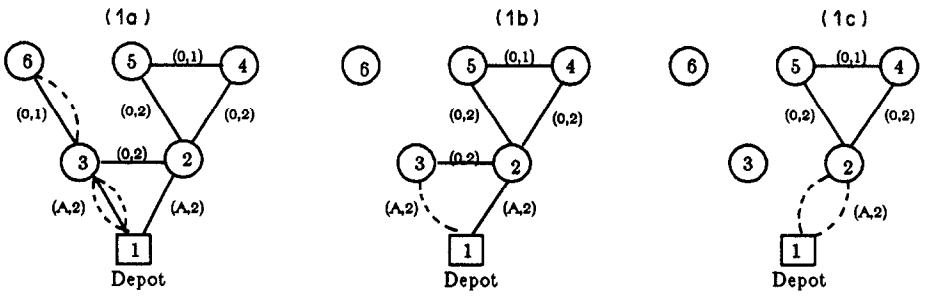

Fig. 1(a)–(c)

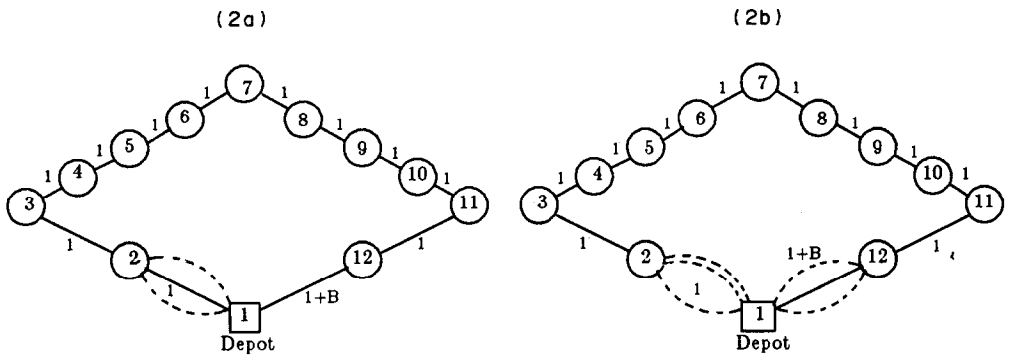

Fig. 2(a)–(b)

When feasible cycles are constructed, one must remove the cycles from the graph. If the remaining graph has no odd-degree nodes, two shortest paths of artificial arcs must be added to the remaining graph from the depot node to its nearest node, second nearest node, etc., until a feasible cycle is found. However, we feel that this successive adding is unnecessary as it may generate many additional cycles which collectively increase the value of the solution, as shown in the following example.

Example 2. Consider the network depicted in Fig. 2(a) with all arc costs indicated. Let all arc demands be 1, and the vehicle capacity W be 6. Since no feasible cycles exist and all nodes are of even-degree, one must add two shortest paths of artificial arcs to the graph from the depot to its nearest node 2, which allows the feasible cycle (1, 2, 1) to be constructed. The remaining graph is shown in Fig. 2(b) in which node 1 and 2 are of odd-degree. The minimal cost 1-matching algorithm is applied which yields the artificial arc (1, 2). Since there exists no feasible cycles, one must add two shortest paths of artificial arcs from the depot to its nearest node 2. However, there still exists no feasible cycles. Two shortest paths of artificial arcs from the depot to its second nearest node 12 (assume $0 < B < 1$) are added, which allows one to construct the feasible cycle (1, 12, 1). Continuing this constructing procedure, the following set of feasible cycles will be generated: (1, 2, 1), (1, 12, 1), (1, 2, 3, 2, 1), (1, 12, 11, 12, 1), (1, 2, 3, 4, 3, 2, 1), (1, 12, 11, 10, 11, 12, 1), (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1). The total cost, therefore, is $36 + 7B$. Compare this heuristic solution with the optimal solution consisting of the cycles (1, 2, 3, 4, 5, 6, 7, 6, 5, 4, 3, 2, 1) and (1, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1), having a total cost of $24 + B$, then we have (heuristic solution)/(optimal solution) $= (36 + 7B)/(24 + B) > 36/24 = 3/2$. That is, the value of the heuristic solution exceeds the value of the optimal solution over 50%.

NEW ALGORITHMS

In the construct-strike algorithm, feasible cycles are constructed and removed only if the remaining graph is not separated into disconnected components. This restriction seems unnecessary, as examples show that this restriction may cause the algorithm to perform very poorly. In example 1 we observed that if we remove the cycle (1, 2, 3, 1) and apply the minimal spanning tree algorithm to render the remaining graph connected by treating each component as a single node, and then proceed with the original algorithm, the optimal solution consisting of cycles (1, 2, 3, 1) and (1, 2, 4, 5, 2, 3, 6, 3, 1), having a total cost of 4A, can be obtained. When feasible cycles are constructed and removed, if the remaining graph forms an Euler cycle, one adds two shortest paths of artificial arcs to the remaining graph from the depot to its nearest node, second nearest node, etc., until a feasible cycle is found. However, we feel that the successive adding of shortest paths is unnecessary as examples show that it could cause the algorithm to perform poorly. When an Euler cycle exists, one could form a cycle by extending a path by joining arbitrary demand arcs until the vehicle capacity is exhausted, then the shortest return path to the depot is followed. Applying the original algorithm along with this modification to the problem in example 2, the optimal solution consisting of cycles (1, 2, 3, 4, 5, 6, 7, 6, 5, 4, 3, 2, 1) and (1, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1), having a total cost of 24, can be obtained. These observations motivate the following algorithm which is based on the original construct-strike algorithm.

Modified construct-strike algorithm

1. *Construct-strike:* Start with the depot node and construct a feasible cycle. To fully use the vehicle capacity, in forming each cycle, the arc (i, j) added (to extend the tour path) should maximize the total demand on the least-quantity path [the shortest path with respect to the demands $q(i, j)$] from node j back to the depot. Remove the cycle from the graph. Repeat this cycle construct-then-remove procedure until no more feasible cycles can be found. Proceed to step 2.

2. *MST segment:* If the remaining graph is disconnected, apply the minimal spanning tree (MST) algorithm to render the remaining graph connected by treating each component as a single node, where the distance between each pair of components is defined as the length of the shortest path which links the correspondent pair of components. Add the artificial arcs from the MST to the remaining graph, and move to step 3.

3. *Matching segment:* Apply the minimal cost 1-matching algorithm to the graph formed in step 2 so that an Euler cycle may be generated. If feasible cycles can be found, return to step 1. Otherwise, move to step 4.

4. *Construct-strike:* Start with the depot node, construct a cycle by adding arbitrary demand arcs to extend the path until the vehicle capacity is exhausted. Then, the shortest return path to the depot is followed to form a cycle. Remove the cycle and all artificial arcs from the graph. Return to step 2.

5. Repeat steps 1–4 until the whole graph is covered.

Since the removal of the first cycle might affect the final solution, we keep a more flexible algorithm by taking each arc that is incident to the depot as the first arc in constructing the initial cycle in order to generate a complete solution. And the final solution from this approach, is then the best among all these solutions.

Computational complexity. The bottleneck of the modified construct-strike algorithm is the minimal cost 1-matching and the all-pair shortest path segments. Each requires an order of $O(n^3)$ computations. Since the matching and all-pair shortest path procedures will be iteratively repeated for no more than m times, and there are at most $n-1$ demand arcs incident to the depot (and thus there will be at most $n-1$ solutions generated), the complexity of the modified construct-strike algorithm is $O(mn^4)$ where n = number of nodes, and m = number of arcs in the graph.

Example 3. To illustrate this modified approach we work through one of the test problems that is shown in Fig. 3(a), which has node 7 as the depot and the vehicle capacity $W = 10$. Note that in this problem, the value of the lower bound obtained by the matching bounding procedures proposed by Golden and Wong [1], is 275. Suppose we start with the depot and use (7, 2) as the starting arc in the cycle construction. In forming each cycle, one must extend the path by adding the arcs that maximize the total demand on the least-quantity return path to the depot. Therefore, we compute the total demand on each of the least-quantity return paths using one of the arcs (2, 1), (2, 3), (2, 4) and (2, 5), as shown below in (a).

(a)

$(2, j)$	Total demand on the least-quantity path	$\frac{c(i,j)^2}{q(i,j)}$
(2, 1)	4	36
(2, 3)	4	225
(2, 4)	3	162
(2, 5)	4	32

(b)

Current node	Candidate arc	Arc selection	Demand on least quantity path
2	(2, 1) (2, 3) (2, 4) (2, 5)	(2, 3)	4
3	(3, 4) (3, 8) (3, 9)	(3, 8)	3
8	(8, 7) (8, 9) (8, 12)	(8, 9)	4
9	(9, 3) (9, 7) (9, 10)	(9, 3)	4
3	(3, 4)	(3, 4)	3
4	(4, 2) (4, 1) (4, 6) (4, 7) (4, 11)	(4, 7)	0

Since there is a tie, we decided to take the one that has the maximal value of $c(i, j)^2/q(i, j)$, which allows the arc (2, 3) to be selected to join the cycle. Continue this arc selection procedure, the feasible cycle (7, 2, 3, 8, 9, 3, 4, 7), having a total cost of 79, can be constructed, as shown in (b) above.

Processing with the algorithm by repeating step 1 until no more feasible cycles can be found,

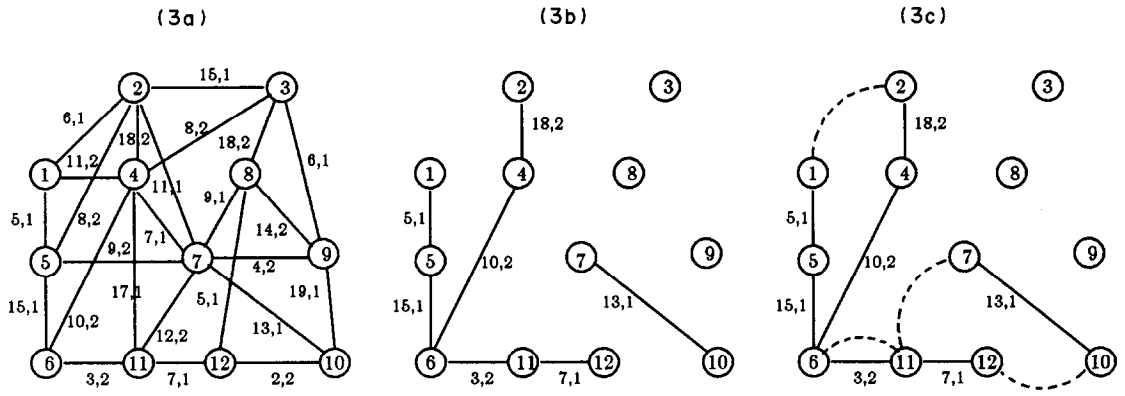

Fig. 3(a)–(c)

the following three cycles (7, 2, 3, 8, 9, 3, 4, 7), (7, 5, 2, 1, 4, 11, 7) and (7, 8, 12, 10, 9, 7) may be constructed. The remaining graph is shown in Fig. 3(b). Now since the remaining graph is disconnected, we treat each component as a single node and apply the minimal spanning tree algorithm to connect the graph, yielding the artificial arc (10, 12). Since odd-degree nodes exist, the minimal cost 1-matching algorithm is applied so that an Euler cycle may be generated. This yields artificial arcs (1, 2), (6, 11) and (7, 11). The resulting graph is shown in Fig. 3(c). And finally, the cycle (7, 10, 12, 11, 6, 5, 1, 2, 4, 6, 11, 7), having a cost of 94, can be constructed. In summary, see (c) below. The total cost is 275.

(c)

Cycle number	Cycle path	Cycle cost
1	(7, 2, 3, 8, 9, 3, 4, 7)	79
2	(7, 5, 2, 1, 4, 11, 7)	63
3	(7, 8, 12, 10, 9, 7)	39
4	(7, 10, 12, 11, 6, 5, 1, 2, 4, 6, 11, 7)	94

(d)

Starting arc	Solution
(7, 2)	275
(7, 4)	300
(7, 5)	283
(7, 8)	300
(7, 9)	309
(7, 10)	300
(7, 11)	300

In this problem, there are seven arcs incident to the depot, each arc is used to generate a complete solution. This procedure generates the solution 275, 283, 300, 309, 300, and 300, with each starting from a different arc that is incident to the depot to generate the initial cycle (as shown in (d) above).

The final solution from this approach is therefore 275, which is optimal. Compare this solution with that generated by the original construct-strike algorithm, which has a value of 364, the improvement is significant.

Modified path-scanning algorithm

In the path-scanning algorithm, five path scanning optimization criteria are considered. Each one is used to generate a complete solution. The final solution from this approach is the best solution of the five.

In the following, we present a variation of the path-scanning algorithm. Instead of using each single criterion to generate a complete solution, we could experiment with the following mixed-type criteria.

- Select one of the five criteria at random to use at each step, and generate a complete solution. The best of k solutions could then be chosen as the final solution from this new approach.
- Select one of the five criteria with varying probabilities (for example, criterion 1 receives probability 0.6 and each of the other four criteria receives probability 0.1) to use at each step, and generate a complete solution. The best of k solutions could then be chosen as the final solution of this new approach.

In testing the modified path-scanning algorithm, we experimented the mixed-type criteria with varying probabilities. These include the probabilities (0.2, 0.2, 0.2, 0.2, 0.2), (0.6, 0.1, 0.1, 0.1, 0.1), (1/3, 1/3, 1/3, 0, 0) and (0.1, 0.1, 0.1, 0.35, 0.35) with each of the ith coordinates representing the probability for selecting the ith criterion. After some experiment, we decided to use the probability (0.2, 0.2, 0.2, 0.2, 0.2) in the modified path-scanning algorithm. That is, randomly select one of the five criteria to use at each step, and generate a complete solution. As the best of k solutions is chosen to be the final solution, the accuracy of the modified path-scanning algorithm can always be improved by increasing the number of solutions. In this paper, however, the number of solutions, k, is set to 30.

Computational complexity. The bottleneck of the modified path-scanning algorithm is the all-pair shortest path segment which requires an order of $O(n^3)$ computations. Therefore, the complexity of the modified path-scanning algorithm is $O(n^3)$.

COMPUTATIONAL RESULTS

In order to compare the new approaches with the existing algorithms, we ran through the set of problems that were originally tested in DeArmon [8] and Golden *et al.* [5]. The size of these test problems ranges from 7 to 27 nodes, 11 to 55 arcs. Some of these networks are dense with small arc demands, and some are sparse with large arc demands. The characteristics of these networks are displayed in Table 1.

Table 2 presents the solutions generated by the modified construct-strike algorithm. Each single solution in each problem is a complete solution generated by using one of the arcs incident to the depot, to construct the first feasible cycle. And the final solution is the best among these single solutions. For example, problem 1 has five arcs incident to the depot, each arc is used to generate a complete solution, therefore, five solutions, 337, 365, 323, 359 and 359 are generated, and the final solution for this problem is 323.

Table 3 presents the solutions generated by our new approaches and the existing algorithms on the 23 test problems. The value of the lower bounds were obtained by using both the matching lower bound proposed by Golden and Wong [1], and the node scanning lower bound proposed by Assad *et al.* [9], which is to be used as a convenient reference point for assessing the accuracy of the heuristic solutions. From Table 3, we can see that the modified construct-strike algorithm significantly outperform the original construct-strike algorithm, as all the 23 test problems received

Table 1. Characteristics of the 23 test problems

Problem	N	M	W	Q	Depot	Density (% complete)
1	12	22	5	22	1	33
2	12	26	5	26	1	39
3	12	22	5	22	1	33
4	11	19	5	19	1	35
5	13	26	5	26	1	33
6	12	22	5	22	1	33
7	12	22	5	22	1	33
8	27	46	27	249	2	13
9	27	51	27	258	2	14
10	12	25	10	37	7	38
11	22	45	50	225	7	19
12	13	23	35	212	1	30
13	10	28	41	240	4	63
14	7	21	21	89	1	100
15	7	21	37	112	1	100
16	8	28	24	116	1	100
17	8	28	41	168	1	100
18	9	36	37	153	1	100
19	11	11	27	66	1	20
20	11	22	27	106	1	40
21	11	33	27	154	1	60
22	11	44	27	205	1	80
23	11	55	27	266	1	100

N = number of nodes; M = number of arcs; W = the vehicle capacity; Q = total arc demand from the network.

Table 2. Performance of the modified construct-strike algorithm (underlines indicate the best solutions)

Problem											Best solution
1	337	365	323	359	359						323
2	345	365	345	361	345	361					345
3	275	304	324	330	275	330					275
4	325	317	287	339	339						287
5	389	391	401	386	386						386
6	321	319	315	321	319						315
7	342	338	343	346	338	325					325
8	366	366	376	376	366						366
9	401	346	346	346	367	401	401				346
10	275	300	283	300	309	300	300				275
11	433	432	428	410	418	406	406				406
12	645	645	645	645							645
13	550	568	550	568	550	544	544	544			544
14	104	102	102	102	102	102					102
15	58	58	58	58							58
16	129	131	127	127	127	127	127				127
17	91	91	91	91	91	91	91				91
18	164	164	166	164	164	164	164	164			164
19	63	63	63								63
20	127	123	127	123	127	124					123
21	156	157	158	158	156	164	164	157	158		156
22	200	200	201	200	200	200	200	200	204	202	200
23	235	237	239	237	237	233	233	237	235	235	233

Table 3. Solutions of the 23 test problems

					Modified	
Problem	Lower bound	Construct strike	Path scan	Augment merge	Construct strike	Path scan
1	310	331	316	326	323	316
2	339	418	367	367	345	355
3	275	313	289	316	275	283
4	274	350	320	290	287	292
5	370	475	417	383	386	401
6	295	356	316	324	315	319
7	312	355	357	325	325	325
8	264	407	416	356	366	380
9	253	364	355	339	346	357
10	275	364	302	302	275	281
11	395	501	424	443	406	424
12	424	655	560	573	645	566
13	544	560	592	560	544	551
14	100	112	102	102	102	100
15	58	58	58	58	58	58
16	127	149	131	131	127	131
17	91	91	93	91	91	93
18	164	174	168	170	164	167
19	55	63	57	63	63	55
20	121	125	125	123	123	123
21	156	165	168	158	156	163
22	200	204	207	204	200	202
23	233	237	241	237	233	244

Lower bounds were obtained by using both the matching lower bound and the node scanning lower bound. Underlines indicate the best solutions.

better solutions from the modified approach. In comparing the modified path-scanning algorithm with the original path-scanning algorithm, we observed that 19 out of 23 test problems received better solutions from the modified approach. Once again, the modified approach significantly outperform the original algorithm.

Gain in precision. In Table 4, the performance of all the five algorithms are compared in terms of average deviation from the lower bound, average rank among those 23 test problems, number of problems receiving the best solutions among all algorithms, number of problems receiving optimal solutions, and the worst solution in terms of percentage above the lower bound.

On the existing set of test problems, the two proposed modified procedures outperformed the original algorithms (averagely) by approx. 10.3% (modified construct-strike) and 3.0% (modified

Table 4. Comparison of performance of the five algorithms

				Modified	
	Construct strike	Path scan	Augment merge	Construct strike	Path scan
Average % above the lower bound	17.91	11.03	9.16	7.59	8.05
Average rank among the five algorithms	3.52	2.78	2.35	1.43	2.13
Number of problems receiving optimal solutions	2	1	2	10	3
Number of problems receiving best solution among all algorithms	2	3	7	16	6
Worse solution among 23 problems in terms of % above the lower bound	54.4	57.5	35.1	52.1	43.9

path-scanning) of the problem lower bounds respectively. The worst solution obtained in terms of % above the lower bound for the two original algorithms, have been also reduced to 2.3% for the modified construct-strike algorithm and 13.6% for the modified path-scanning algorithm. In terms of number of problems receiving optimal solutions, 10 out of 23 problems solutions (over 43%) obtained by the modified construct-strike algorithm, are optimal. Compare that with 2 out of 23 (less than 9%) by the original construct-strike algorithm, the improvement is significant.

It should be noted that while computing the problem lower bounds for assessing the accuracy of the heuristic solutions, we have also implemented the node scanning lower bounding procedure proposed by Assad *et al.* [9], along with the matching lower bound, to obtain better lower bound values. Consequently, the averaged deviation of the heuristic solutions from the problem lower bounds (shown in Table 4), is decreased, and therefore, it may not be the same as the results presented by Golden *et al.* [5].

Computer run time. Computer run times for the five algorithms vary quite extensively. Roughly, the size of the 23 test problems can be characterized into the following three categories.

- All the 23 test problems are less than 13 nodes except problems 8, 9 and 11.
- Problems 19–23 all have 11 nodes and 11–55 arcs.
- Problems 8 and 9 have 27 nodes and 46–51 arcs. They are the largest problems that were tested.

In our testing, the run times for problems having same size and same vehicle capacity are very much the same on these problems. However, run times may vary for problems having same size but with different vehicle capacity. Therefore, instead of showing the average run time, we display the run times for three test problems (one from each category) in CPU seconds on the UNIVAC 1108 as an indication of the relative efficiency of the five algorithms [see (e) below].

(e)

Problem number	Number of nodes	Number of arcs	Construct strike	Path scan	Augment merge	Modified construct strike	Modified path scan
1	12	22	1.6	1.6	38.4	14.3	4.8
8	27	46	12.3	11.0	91.2	201.5	49.2
23	11	55	2.3	4.6	165.6	52.4	6.9

The run time of the modified construct-strike algorithm for problem 8 seems quite large. This problem has 27 nodes, 46 arcs, and (along with problem 9) is the largest problem that was tested in DeArmon [8] and Golden *et al.* [5]. Each single solution takes approx. 37 CPU seconds. Roughly speaking, we have:

- the construct-strike and the path-scanning algorithms are comparable in running times;
- the modified construct-strike algorithm takes approx. 2–3 times (averagely) the run time of the original construct-strike algorithm for each single execution;
- the modified path-scanning algorithm takes approx. 3–4 times (averagely) the run time of the original path-scanning algorithm;
- the modified construct-strike, and the augment-merge algorithms take considerably more computer time than the other three procedures (construct-strike, path-scanning and modified path-scanning).

We note that the modified construct-strike algorithm performs extremely well on test problems 13–23 (except for problems 14 and 19) as it received best solutions among all algorithms uniformly over these test problems. On 8 out of 10 problems, it received the problem lower bounds; hence the solution obtained must be optimal. These networks are very dense, most are either 100% complete or at least 80% complete networks. It seems that this algorithm does very well on dense networks, so we tested another 20 problems. These problems are all 100% complete networks, range from 11 nodes, 55 arcs, to 17 nodes, 136 arcs.

Table 5 presents the solutions of these 20 problems generated by our modified procedures and the two original algorithms. The performance of the modified construct-strike algorithm is somewhat surprising in that it received best solutions among all algorithms that we tested uniformly over these 20 test problems. On 19 out of 20 problems, it received optimal solutions. Comparisons of the performance among algorithms in terms of number of problems receiving best solutions, and number of problems receiving optimal solutions, is shown in Table 6. The run time of the modified construct-strike algorithm in CPU seconds are very large when the size of the problem is increased to 17 nodes, 136 arcs, as one run takes approx. 60 CPU seconds. However, in each of the 20 problems that we tested, the first single solution have achieved its problem lower bound and hence must be optimal. Therefore, when the network is dense, we might be able to reduce the run time by only taking the first single solution as the final solution.

For further testing, we took additional 15 problems. These problems are all between 70% complete and 90% complete, range from 11 nodes, 45 arcs to 17 nodes, 116 arcs. The solutions of these 15 problems generated by the algorithms are displayed in Table 7, comparisons among the algorithms in terms of the number of problems receiving the best solutions, and number of problems receiving optimal solutions, is shown in Table 8. On this set of 15 test problems, the modified construct-like algorithm, once again, received best solutions among all algorithms that we tested uniformly over all the 15 problems, on 8 out of 15 problems, it received problem optimal solutions.

Table 5. Solutions of the 20 test problems (100% complete) (underlines indicate the best solution)

				Modified	
Problem	Lower bound	Construct strike	Path scan	Construct strike	Path scan
1	225	229	243	227	235
2	261	271	277	261	275
3	219	229	226	219	225
4	208	210	218	208	216
5	234	238	249	234	246
6	320	336	324	320	332
7	397	401	406	397	409
8	307	307	315	307	313
9	272	280	286	272	277
10	288	296	292	288	300
11	404	410	411	404	412
12	314	326	319	314	320
13	248	256	252	248	251
14	316	320	323	316	323
15	322	330	331	322	329
16	408	410	414	408	412
17	318	320	324	318	323
18	246	250	253	246	252
19	316	330	322	316	324
20	322	328	333	322	328

Table 6

			Modified	
	Construct strike	Path scan	Construct strike	Path scan
Number of problems receiving best solutions among algorithms	1	0	20	0
Number of problems receiving optimal solutions	1	0	19	0

Table 7. Solutions of the 15 test problems (70–90% complete) (underlines indicate the best solutions)

Problem	Lower bound	Construct strike	Path scan	Modified Construct strike	Modified Path scan
1	223	233	231	223	237
2	174	178	182	176	178
3	171	171	174	171	173
4	205	221	212	209	212
5	282	286	294	284	290
6	349	366	365	362	362
7	269	281	281	269	277
8	242	246	254	242	251
9	242	252	255	242	248
10	363	367	371	363	369
11	277	279	281	277	285
12	216	225	221	217	221
13	297	309	303	299	303
14	290	298	297	290	295
15	216	246	230	220	233

Table 8

	Construct strike	Path scan	Modified Construct strike	Modified Path scan
Number of problems receiving best solutions among algorithms	1	0	15	1
Number of problems receiving optimal solutions	1	0	8	0

CONCLUSIONS

In this paper, we presented two new algorithms, the modified construct-strike algorithm, and the modified path-scanning algorithm. We have also automated the node scanning lower bound proposed by Assad *et al.* [9], along with the matching lower bound, to be used for assessing the accuracy of the heuristic solutions.

In our testing, these two modified procedures significantly outperform the three existing algorithms, i.e. the construct-strike algorithm, the path-scanning algorithm and the augment-merge algorithm. As on the 23 problems that were originally tested in DeArmon [8] and Golden *et al.* [5], both of the new procedures which we presented, received better problem solutions than that by the three existing algorithms. And the solutions obtained from our modified procedures, were approx. 7.6% and 8.0% above the problem lower bounds. Compare that with 17.9% and 11.0% (above the problem lower bounds) obtained by the two original algorithms, the improvement (with 10.3% and 3.0%) is significant. Further test results also show that the modified construct-strike algorithm works very well on dense networks, as 27 out of 35 problems that we tested (all at least 70% complete), received problem optimal solutions.

While we made no claims that the modified procedures always do better than the existing algorithms, we do believe, however, that the relative performance of these heuristic procedures on the sample problems tested (58 problems total) in this paper, is indicative of their relative performance in general; in particular, when the underlined networks are dense with small arc demands. This is certainly the situation of many real-world applications, such as routing of mail-delivering vehicles, street sweepers, or household refuse collection trucks in urban district.

If the computer run time is not primary concern, the modified procedures that we presented in this paper are ones that should probably be applied. Also, since we found none of the existing algorithms as well as the modified procedures worked well on sparse networks with large arc demand, research in handling such problems should be encouraged.

Acknowledgements—The author would like to thank the editor Dr Samuel J. Raff for his time and patience. The author also wants to thank the two anonymous referees for a careful reading of the paper and several valuable suggestions for improvements.

REFERENCES

1. B. Golden and R. Wong, Capacitated arc routing problems, *Networks* **11**, 305–318 (1981).
2. J. Edmonds and E. Johnson, Matching, Euler tours and the Chinese postman problem. *Math. Prog.* **4**, 88–124 (1973).
3. N. Christofides, The optimal traversal of a graph. *Omega* **1**, 719–732 (1973).
4. H. Stern and M. Dror, Routing electric meter readers. *Computers Opns Res.* **6**, 209–223 (1979).
5. B. Golden, J. DeArmon and E. K. Baker, Computational experiments with algorithms for a class of routing problems. *Computers Opns Res.* **10**, 47–69 (1983).
6. L. Chapleau, J. Ferland, G. Lapalme and J. M. Rousseau, A parallel insert method for the capacitated arc routing problem. *Opns Res. Lett.* **3**, 95–99 (1984).
7. W. L. Pearn, The capacitated Chinese postman problem. Ph.D. Thesis, University of Maryland at College Park (1984).
8. J. DeArmon, A comparison of heuristics for the capacitated Chinese postman problem, Master's Thesis, University of Maryland at College Park (1981).
9. A. Assad, W. L. Pearn and B. Golden, The capacitated Chinese postman problem: lower bounds and solvable cases. *Am. J. Mathemat. Mgmt Sci.* **7**, 63–88 (1987).