An Ant Colony Optimization for the Capacitated Arc Routing Problem

Han-Shiuan Tsai
Department of Industrial Engineering & Management
Yuan Ze University, Taoyaun, Taiwan
Tel: (+886) 3-463-8800 ext. 2516, Email: mjes09@gmail.com

Ching-Jung Ting†
Department of Industrial Engineering & Management
Yuan Ze University, Taoyaun, Taiwan
Tel: (+886) 3-463-8800 ext. 2526, Email: ietingcj@saturn.yzu.edu.tw

Abstract. The capacitated arc routing problem (CARP) is a difficult combinatorial optimization problem which has wide applicability in real world logistics problems. The CARP is to find a set of routes with minimum costs for a set of demand arcs with vehicle capacity limitations. Due to its NP-hard property, CARP cannot be solved within reasonable time by an exact algorithm. In recent years, metaheuristics algorithms have been developed to solve the CAPR. In this research, we propose an ant colony optimization (ACO) algorithm to solve the problem. The proposed ACO was tested with three sets of benchmark instances from the literature for its effectiveness and compared with the existing best performing metaheuristics. The computational results show that the ACO is competitive with the compared heuristic algorithms.

Keywords: capacitated arc routing problem, ant colony optimization, logistics

1. INTRODUCTION

The capacitated arc routing problem (CARP) has been the subject by many researchers during last decades due to its wide range of real world applications. The CARP can be informally described as follows. We are given a graph with a set of nodes and edges, where contains required edges with nonnegative demand. A traversal cost is associated with each edge. The objective is to determine a set of vehicle routes of minimum total cost, such that each route starts and ends at the depot, each required edge is served by one single vehicle, and the total demand serviced on a route of a vehicle must not exceed the vehicle capacity.

Many applications occur for the CARP, such as household waste collection, snow plowing, winter gritting, postal deliveries, and street sweeping, among others (Dror, 2000). CARP is a NP-hard problem, classical exact methods are only applicable for relative small size instances. Heuristic and metaheuristics are more efficient for solving medium to large size CARP. In this work, we investigate a population-based algorithm for the CARP. Ant colony optimization (ACO) algorithms have been proved to be very effective for solving a large number of difficult combinatorial optimization problems (Chandra Mohan and Baskaran, 2012; Ting and Chen, 2013), including CARP (Lacomme et al., 2004a; Santos et al., 2010). Based on our previous experiences on ACO applied to various combinatorial problems, we provided an effective ACO to compare with the current state-of-the-art CARP methods.

The remainder of this paper is organized as follows. In section 2, we provided related work of capacitated routing problem. We proposes ant colony optimization algorithm (ACO) for the CARP in section 3. Section 4 tests the effectiveness of the proposed ACO by three sets of benchmark instances and compares the results with other leading algorithms. Section 5 concludes this research and suggests future research.

2. RELATED WORK

The CARP was first defined in 1981 by Golden and Wong, who proved its NP-hardness. To the best of our knowledge, Dror (2000), Wøhlk (2008), Corberán and Prins (2010), and Corberán and Laporte (2014) provided good surveys on the CARP. Dror (2000) collected the arc routing problem related articles in theory and applications before 2000. Wøhlk (2008) provided an overview of developments in the CARP between 2000 and 2007. More than 30 articles for the period 2000 to 2010 dealing with CARP variants were discussed by Corberán and Prins (2010). Corberán and Laporte (2014) provided up-to-date collections of researches in theory and applications. We refer interested reader to these excellent survey papers and books.

Due to the CARP complexity, many real world large-scale instances are intractable for exact algorithms, heuristics and metaheuristics were developed to provide high-quality solutions on real world applications. Examples of heuristics for the CARP include path-scanning (Golden et al., 1983), augment-insert (Pearn, 1991), augment-merge (Golden and Wong, 1981), and Ulusoy's tour splitting method (Ulusoy, 1985).

Among the metaheuristic methods, local-search based approaches are popular on earlier publications. Eglese (1994) and Wøhlk (2005) proposed simulated annealing for the CARP. Hertz et al. (2000), Brandão and Eglese (2008), and Mei et al. (2009) developed tabu search (TS) algorithms for the CARP. Hertz and Mittaz (2001) and Polacek et al. (2008) proposed variable neighborhood search (VNS) to solve the CARP. Beullens et al. (2003) designed an effective guided local search (GLS) while Usberti et al. (2003) developed GRASP with evolutionary path-relinking for the CARP.

Recently, population-based approaches are proposed and generally achieve better performances. Lacomme et al. (2004a), Mei et al. (2009), Tang et al. (2009) proposed memetic algorithms (MA) for the CARP, while Lacomme et al. (2006) developed a genetic algorithm (GA) for the CARP. Both MA and GA improve their solutions based on crossover and mutation operators. Lacomme et al. (2004b) and Santos et al. (2010) designed ant colony optimization algorithm (ACO) for the CARP. Among these methods, the ant colony optimization algorithm provided better results on the classical test instance sets.

3. ANT COLONY OPTIMIZATION

The ant colony optimization (ACO), which is learned from the behavioral of real ant colonies, was first proposed by Dorigo et al. (1996). Subsequently, many variants of ACO have been developed and applied extensively in the combinatorial optimization problems. Dorigo and Stützle (2004) provided descriptions of available ACO algorithms and related literature review. In principle, ACO can be applied to any discrete optimization problem for which some solution construction mechanism can be conceived.

The procedures of our ACO are described as follows and introduced in the following sections.

1. Initialization
   Initialize parameters and value of pheromone matrices.
2. CARP process
   2.1 Select a vehicle to a route.
   2.2 Select the next arc based on state transition rule.
   2.3 If all required arcs are assigned, go to step 3.
   2.4 If the vehicle capacity is exceeded, go to step 2.1; otherwise go to step 2.2.
3. Local pheromone updating
   3.1 Update the pheromone levels.
   3.2 If all ants have solutions, go to step 4; otherwise go to step 2.
4. Local search
   Using swap、2-opt and insertion on the best solution of current iteration. If the iteration best solution is better than the global best solution, update the global best solution.
5. Global pheromone updating
   Update the pheromone by iteration best solution and best solution till now.
6. Terminating condition
   If the terminating criterion (maximum number of iterations in this paper) is met, stop; otherwise repeat steps 2~5.

3.1 Solution Representation

The solution representation for the CARP use the natural encoding approach as used in most of vehicle routing problems. A list of required arcs is connected by implicit shortest paths. In such a way, the encoding and decoding of a solution is easy to compute the total travel cost of all routes. Figure 1 presents a solution of three vehicle routes for the 10 required arcs. The first vehicle will service required arcs 1, 2, 3, and 4, required arcs 5, 6, and7 are serviced by second vehicle, and the required arcs 8 to9 are serviced by third vehicle.

1	2	3	4	0	5	6	7	0	8	9	10

Figure 1: A representation of 10 nodes solution

3.2 Solution Construction

In our ACO, when a vehicle visits a required arc i, ant h movers to a required arc k by the following state transition rule.

$$s = \begin{cases} arg \max_{j \in N_i} \left\{ (\tau_{ij}) \cdot (\eta_{ij})^{\beta} \right\} & , q \leq q_0 \\ S & , q > q_0 \end{cases} \tag{1}$$

$$S:\ P_{ij}^{k}(t)=\begin{cases}\dfrac{(\tau_{ij})\cdot(\eta_{ij})^{\beta}}{\sum_{q\in N_i}(\tau_{iq})\cdot(\eta_{iq})^{\beta}} & ,if\ j\in N_i\\ 0 & ,otherwise\end{cases} \tag{2}$$

where N_i is the set of arcs which are not visited by ant k at arc i, τ_{ij} is the pheromone of the shortest path between arcs i and j, η_{ij} is defined as the reciprocal of the shortest path between arcs i and j. β is the parameter that determines the relative effect of τ_{ij} versus η_{ij} ($\beta > 0$), q is a random variable uniformly distributed in [0, 1], and q_0 is a pre-defined parameter ($0 \le q_0 \le 1$). If $q \le q_0$, then the best arc j for arc i is determined according to eq. (2). On the contrary, it is chosen according to S which is a random variable selected according to the probability distribution given in eq. (3). Hence, the parameter q_0 determines the relative importance of exploitation eq. (2) versus exploration eq. (3).

3.3 Pheromone Update

The pheromone updating of a typical ACO includes global and local updating rules. The ants apply a local pheromone update rule immediately after they crossed a shortest path (i, j) during the tour construction. The local pheromone updating rule of our ACO is

$$\tau_{ij}^{new}=\rho\cdot\tau_{ij}^{old}+(1-\rho)\cdot\tau_0,\ \text{if } \{\text{edge}(i,j)\in T_k\} \tag{3}$$

where T_k denotes the routes constructed by ant k, ρ is the pheromone decay parameter in the range of [0, 1] that regulates the reduction of pheromone on the edges. The τ_0 is the initial value of the pheromone matrix for the route construction rule, and is set to be 0.2 in this paper.

In our ACO, the best elitist tours, including the global-best tour (T_b) and the iteration-best tour (T_s) of CARP, are allowed to lay pheromone on the edges that belong to them. The idea here is to balance between exploitation (through emphasizing the global-best tour) as well as exploration (through the emphasis to the iteration-best tour). The global updating rule of ACO for CARP is described as follow.

$$\tau_{ij}^{new}=\rho\cdot\tau_{ij}^{old}+(1-\rho)\cdot\Delta\tau_{ij} \tag{4}$$

where

$$\Delta\tau_{ij}=\begin{cases}1/L_b & \text{if } (i,j)\in T_b\\ 1/L_s & \text{if } (i,j)\in T_s\\ 0 & \text{otherwise}\end{cases} \tag{5}$$

L_b and L_s denote the objective function value of the global-best solution and the iteration-best solution of CARP, respectively; T_b and T_s are the global best solution and iteration-best solution, respectively.

3.4 Local Search

Local search heuristic is a time-consuming procedure but often used to improve the solutions of ACO. To save the computation time, we only apply local search on the iteration-best solution in this paper. In addition, three local search methods are involved in our ACO, including 2-opt, swap and insertion. The local search could be applied within route or between routes. This is because that diverse neighborhood moves can expand the solution searching space. The methods are kept and sorted according to their overall performance on the tested instances. 2-opt move is to delete two linkage of a route or two routes. The broken routes are reconnected by a new linkage. Two customers are exchanged in swap. Insertion is to move one customer from its current position to another position, in the same route or in a different route.

4. COMPUTATIONAL EXPERIMENTS

The ACO is in Microsoft Visual Studio 2010 C++ and implemented on a computer with Intel Core (TM) i5-2400 3.10 GHz processor and 8 GB RAM under Windows 7 operation system. The results are compared with the best methods for the CARP in the literature. These tests are done on three sets of benchmark instances.

The first set *gdb* contains 23 instances from DeArmon (1981) with 7 to 27 nodes and 11 to 55 edges. The second set *val* contains 34 instances from Benavent et al. (1992) with 25 to 50 nodes and 34 to 97 required edges. The third set *egl* contains 24 instances from Eglese (1994) with 77-140 nodes and 98-190 edges that include 51-190 required edges. All the instances were conducted for 10 independent runs.

In preliminary experiments we tried to find a good parameter setting for the proposed ACO algorithm. We consider a set of parameters for the algorithm and then modifying one at a time, while keeping the others fixed. The parameters that were tested include: $\beta \in \{0.5, 0.8, 1\}$, $\rho \in \{0.1, 0.3, 0.5\}$, $q_0 \in \{0.1, 0.5, 0.9\}$, $b = \{10, 15, 20\}$, and $Iter = \{50, 100, 150\}$. We found that for the parameter setting, $\beta = 0.8$, $\rho = 0.1$, $q_0 = 0.9$, $b = 10$, $Iter = 150$ can provide the best average solution. These parameters will be used for all instances for further experiment.

The following is a brief description of the column headings in tables 1, 3, and 5. The column Inst. indicates the instance name. The columns headed |V| and |A| indicate the number of vertices, required serviced arcs numbers. The column headed BKS gives the best know solution from the literature, while the column CPU shows the computational time in second. The columns min, max, avg, represents the best, worst and average are the minimum, maximum and average cost of the solutions obtained among the 10 runs, respectively. The column headed Gap presents the gap of minimum cost from the best known solution.

For evaluate the effectiveness our proposed ACO, we assess ACO in these three benchmark instance sets with current state-of-the-art algorithms: BACO (Lacomme et al., 2004b), MA (Lacomme et al., 2004a), TSA (Brandão and Eglese, 2008), VNS (Polacek et al., 2008), GA (Lacomme et al., 2006), ANT_12 (Santos et al., 2010). For each data set, we first present our ACO results and then the summary of the comparison.

Table 1 presents the results for the small size *gdb* data set (23 instances). Instances 8 and 9 were removed because they contained inconsistencies. Our ACO is able to obtain the optimal solutions for all 23 instances. The average computational time is only 1 second. Table 2 summarizes the performance of our ACO and other leading heuristic algorithms. Both GLS and our ACO can obtain optimal solutions in 23 instances. All the comparing algorithms were tested on different computers with a CPU ranging from 500 megahertz to 3.1 gigahertz. To compare the efficiency of the algorithms, we used Dongarra's (2014) tables to get a very rough idea of the relative speeds of different computers. If T_a is the computational time and P_a is the computer power (Mflop/s) for one algorithm a, the scaled time of the algorithm is $(T_a/T_g)*(P_a/P_g)$, with g standing for the ACO. GLS is the fastest algorithm among all compared heuristics for this small size data set. Our ACO outperforms the other two ant colony optimization algorithms in terms of solution accuracy.

Table 3 presents the computational results for the medium size *val* data set (34 instances). Our ACO can find all 34 optimal solutions within 4.1 seconds on average. Table 4 further summarizes the results among all comparing algorithms. Our ACO is the only algorithm that can obtain all 34 optimal solutions. The MA is the fastest algorithm among all compared heuristics. However, MA can only reach 29 optimal solutions. Our ACO is the best among three ant colony optimization algorithms in terms of solution quality.

Table 5 shows the computational results for the large size *egl* instances (24 instances). Our ACO can only reach 16 best known solutions out of 24 instances. The computational time is 33.2 seconds on average. The average gap is much higher than previous smaller instances

Table 1: Results for *gdb* data set.

Inst.	|V|	|A|	BKS	CPU	min	max	avg	Gap
gdb1	12	22	316	0.5	316	316	316	0
gdb2	12	26	339	0.6	339	339	339	0
gdb3	12	22	275	0.4	275	275	275	0
gdb4	11	19	287	0.4	287	287	287	0
gdb5	13	26	377	0.6	377	377	377	0
gdb6	12	22	298	0.5	298	298	298	0
gdb7	12	22	325	0.5	325	325	325	0
gdb10	27	46	348	1.7	348	348	348	0
gdb11	27	51	303	2.2	303	333	326.5	0
gdb12	12	25	275	0.7	275	275	275	0
gdb13	22	45	395	1.2	395	395	395	0
gdb14	13	23	458	0.7	458	458	458	0
gdb15	10	28	536	0.6	536	536	536	0
gdb16	7	21	100	0.9	100	100	100	0
gdb17	7	21	58	0.7	58	58	58	0
gdb18	8	28	127	0.9	127	127	127	0
gdb19	8	28	91	0.8	91	91	91	0
gdb20	9	36	164	1.0	164	164	164	0
gdb21	8	11	55	0.3	55	55	55	0
gdb22	11	22	121	0.7	121	121	121	0
gdb23	11	33	156	1.1	156	156	156	0
gdb24	11	44	200	2.0	200	200	200	0
gdb25	11	55	233	3.0	233	233	233	0
Avg.				1.0	253.8	255.1	254.8	0

Table 2: Comparison of computational results for various algorithms of *gdb* data set.

Method	Reference	*Num*	*Gap*	*CPU*	Computer	Mflop/s	*ST*
BACO	Lacomme et al. (2004b)	19/23	0.30	19.8	Pentium III 800 MHz	138	1.40
MA	Lacomme et al. (2004a)	22/23	0.04	3.2	Pentium III 1.0 GHz	192	0.32
TSA	Brandão and Eglese (2008)	21/23	0.08	2.5	Pentium Mobile 1.4Ghz	352	0.36
GA	Lacomme et al. (2006)	19/23	0.21	13.8	Pentium IV 1.8 GHz	292	2.08
GLS	Beullens et al. (2003)	23/23	0.00	1.7	Pentium II 500 MHz	98	0.09
ANT_12	Santos et al. (2010)	22/23	0.04	3.4	Pentium III 1.0 GHz	192	0.34
ACO	Our	23/23	0.00	1.0	Intel Core i5-2400 3.10 GHz	2426	1.00

Table 3: Results for *val* data set.

Inst.	|V|	|A|	BKS	CPU	min	max	avg	Gap
val1a	24	39	173	2.4	173	173	173.0	0
val1b	24	39	173	2.4	173	173	173.0	0
val1c	24	39	245	2.8	245	245	245.0	0
val2a	24	34	227	2.2	227	229	227.3	0
val2b	24	34	259	2.1	259	259	259.0	0
val2c	24	34	457	2.5	457	457	457.0	0
val3a	24	35	81	2.3	81	81	81.0	0
val3b	24	35	87	2.2	87	87	87.0	0
val3c	24	35	138	2.4	138	138	138.0	0
val4a	41	69	400	3.4	400	400	400.0	0
val4b	41	69	412	3.5	412	412	412.0	0
val4c	41	69	428	3.7	428	463	450.9	0
val4d	41	69	530	3.8	530	585	559.1	0
val5a	34	65	423	3.8	423	431	426.4	0
val5b	34	65	446	3.6	446	450	447.3	0
val5c	34	65	474	3.7	474	482	478.6	0
val5d	34	65	575	3.9	575	630	598.9	0
val6a	31	50	223	2.9	223	231	225.3	0
val6b	31	50	233	2.8	233	233	233.0	0
val6c	31	50	317	3.5	317	317	317.0	0
val7a	40	66	279	3.6	279	282	282.4	0
val7b	40	66	283	3.8	283	287	283.3	0
val7c	40	66	334	4.1	334	362	347.0	0
val8a	30	63	386	3.5	386	386	386.0	0
val8b	30	63	395	3.5	395	413	401.4	0
val8c	30	63	521	3.7	521	521	521.0	0
val9a	50	92	323	7.3	323	341	330.1	0
val9b	50	92	326	7.6	326	347	334.1	0
val9c	50	92	332	7.4	332	346	344.5	0
val9d	50	92	389	7.3	389	425	415.1	0
val10a	50	97	428	7.0	428	436	436.2	0
val10b	50	97	436	7.2	436	448	436.8	0
val10c	50	97	446	6.9	446	462	455.8	0
val10d	50	97	525	7.5	525	570	551.1	0
Avg.				4.1	344.4	355.9	350.4	0

Table 4: Comparison of computational results for various algorithms of *val* data set.

Method	Reference	*Num*	*Gap*	*CPU*	Computer	Mflop/s	*ST*
BACO	Lacomme et al. (2004b)	26/34	0.90	276.3	Pentium III 800 Hz	138	4.02
MA	Lacomme et al. (2004a)	29/34	0.23	25.6	Pentium III 1.0 GHz	192	0.49
TSA	Brandão and Eglese (2008)	31/34	0.15	20.2	Pentium Mobile 1.4Ghz	352	0.71
VNS	Polacek et al. (2008)	32/34	0.09	43.9	Pentium IV 3.0GHz	1573	7.30
GA	Lacomme et al. (2006)	23/34	0.50	76.2	Pentium IV 1.8 GHz	292	2.35
GLS	Beullens et al. (2003)	30/34	0.47	81.3	Pentium II 500 Hz	98	0.84
ANT_12	Santos et al. (2010)	26/34	0.03	25.3	Pentium III 1.0 GHz	192	0.51
ACO	Our	34/34	0.00	4.1	Intel Core i5-2400 3.10GHz	2426	1.00

Table 5: Results for *egl* data set.

Inst.	|V|	|A|	BKS	CPU	min	max	avg	Gap
egl-e1-A	77	98	3548	15.8	3548	3595	3563.4	0
egl-e1-B	77	98	4498	15.9	4498	4539	4508.6	0
egl-e1-C	77	98	5595	15.2	5595	5668	5615.3	0
egl-e2-A	77	98	5018	16.7	5018	5032	5023.8	0
egl-e2-B	77	98	6317	17.1	6317	6425	6389.3	0
egl-e2-C	77	98	8335	16.3	8335	8398	8358.2	0
egl-e3-A	77	98	5898	17.0	5898	5986	5943.7	0
egl-e3-B	77	98	7775	17.5	7775	7815	7796.6	0
egl-e3-C	77	98	10292	17.2	10292	10446	10340.4	0
egl-e4-A	77	98	6444	17.2	6444	6464	6461.0	0
egl-e4-B	77	98	8983	18.1	8983	9079	9009.2	0
egl-e4-C	77	98	11596	18.2	11596	11670	11645.8	0
egl-s1-A	140	190	5018	42.8	5018	5065	5035.2	0
egl-s1-B	140	190	6388	43.6	6388	6502	6433.4	0
egl-s1-C	140	190	8518	46.9	8518	8535	8521.5	0
egl-s2-A	140	190	9884	49.4	9936	9997	9974.7	0.5
egl-s2-B	140	190	13100	46.5	13140	13280	13186.4	0.3
egl-s2-C	140	190	16425	48.4	16558	16680	16625.9	0.5
egl-s3-A	140	190	10220	51.3	10249	10323	10306.0	0.3
egl-s3-B	140	190	13682	49.8	13762	13839	13802.4	1.5
egl-s3-C	140	190	17230	49.6	17266	17312	17281.1	0.4
egl-s4-A	140	190	12268	54.2	12325	12638	12513.5	0.5
egl-s4-B	140	190	16321	55.5	16386	16490	16428.2	0.6
egl-s4-C	140	190	20517	56.3	20911	21097	21036.6	1.9
Avg.				33.2	9781.5	9869.8	9825.0	0.3

at 0.3. To give a comparison of the performance of each algorithm, we summarize the results in Table 6. Our ACO can reach 16 best known solutions, which is the most among all compared algorithms, though the average gap performance is not the lowest. The MA is the fastest algorithm for the egl data set, but it only can provide 7 best known solutions. The average gap for MA is much higher than our ACO. Our ACO outperforms BACO in terms of solution quality and computational times. ANT_12 provides lower average gap but needs longer computational times than our ACO.

From tables 2, 4, and 6 we can conclude that our ACO can provide competitive performance against other state-of-the-art algorithms. Comparing to the other two ACO algorithms, BACO and ANT_12, our ACO can provide better solution quality. The CPU time is much shorter than these two ACO algorithms in larger size instances. Though the computational time is not the fastest one, we can obtain best known solutions in 73 out of 81 instances, which is the most among all compared algorithms.

Table 6: Comparison of computational results for various algorithms of *egl* data set.

Method	Reference	*Num*	*Gap*	*CPU*	Computer	Mflop/s	*ST*
BACO	Lacomme et al. (2004b)	1/24	2.14	2341.3	Pentium III 800 Hz	138	4.20
MA	Lacomme et al. (2004a)	7/24	1.74	351.4	Pentium III 1.0 GHz	192	0.84
TSA	Brandão and Eglese (2008)	5/24	1.54	291.4	Pentium Mobile 1.4Ghz	352	1.27
VNS	Polacek et al. (2008)	11/24	0.15	503.2	Pentium IV 3.0GHz	1573	10.30
GA	Lacomme et al. (2006)	2/24	1.73	267.0	Pentium IV 1.8 GHz	292	1.01
ANT_12	Santos et al. (2010)	11/24	0.18	503.0	Pentium III 1.0 GHz	192	1.26
ACO	Our	16/24	0.30	33.2	Intel Core i5-2400 3.10GHz	2426	1.00

5. CONCLUSION

The capacitated arc routing problem (CARP) attracts more attention due to its wide range of real world applications in recent years. We developed an ant colony optimization (ACO) algorithm for effectively solving CARP in this paper. The results presented demonstrate that ACO can provide good performance over three sets of 81 popular CARP benchmark instances. Specifically, ACO with a single parameter setting outperforms the compared algorithms in terms of number of best known solutions that can be found. We believe that our ACO can be adapted to handle other CARP variants with slight modifications of the solution encoding and decoding approaches.

In the future, we would like to incorporate other heuristic, such as path relinking, as a form of intensification solution. Furthermore, we would apply the ACO to real world problems which might need intermediate refill facilities, such as street sweeping and washing.

REFERENCES

Belenguer, J.M. and E. Benavent. (2003) A cutting plane algorithm for the capacitated arc routing problem, *Computers and Operations Research*, 30, 705-728.

Benavent, E., Campos, V., Corberan, E., and Mota, E. (1992) The capacitated arc routing problem: Lower bounds. *Networks*, 22, 669-690.

Beullens, P., Muyldermans, L., Cattrysse, D., and Van Oudheusden, D. (2003) A guided local search heuristic for the capacitated arc routing problem. *European Journal of Operational Research*, 147, 629-643.

Brandão, J. and Eglese, R. (2008) A deterministic tabu search algorithm for the capacitated arc routing problem. *Computers & Operations Research*, 35, 1112-1126.

Chandra Mohan, B. and Baskaran, R. (2012) A survey: Ant colony optimization based recent research and implementation on several engineering domain, *Expert Systems with Applications*, 39, 4618-4627.

Corberán, Á. and Prins, C. (2010) Recent results on arc routing problems: An annotated bibliography, *Networks*, 56, 50-69.

Corberán, Á. and Laporte, G. (2014) *Arc routing: Problems, methods, and applications*, MOS-SIAM Series on Optimization, SIAM, Philadelphia, PA, USA.

DeArmon, J.S. (1981) A comparison of heuristics for the capacitated Chinese postman problem, Master's Thesis, The University of Maryland at College Park, MD, USA.

Dongarra, J. Performance of Various Computers Using Standard Linear Equations Software, Report CS-89-85, University of Tennessee, 2014.

Dorigo, M., Maniezzo, V. and Colorni, A. (1996) Ant system: Optimization by a colony of cooperating agents, *IEEE Transactions on Systems, Man and Cybernetics Part B*, 26, 29-41.

Dorigo, M. and Stützle, T. (2004) *Ant Colony Optimization*, Bradford Books, MIT Press, Cambridge, MA.

Dror, M. (2000) *Arc routing: Theory, solutions and applications*, Kluwer Academic Publishers, Boston, MA, USA.

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

Eydi, A. and Javazi, L. (2012) A novel heuristic method to solve the capacitated arc routing problem, *International Journal of Industrial Engineering Computations*, 3, 767-776.

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

Golden, B.L. and Wong, R.T. (1981) Capacitated arc routing problem, *Networks*, 11, 305-315.

Hertz, A., Laporte, G. and Mittaz, M. (2000) A tabu search heuristic for the capacitated arc routing problem, *Operations Research*, 48, 129-135.

Hertz, A. and Mittaz, M. (2001). A variable neighborhood descent algorithm for the undirected capacitated arc routing problem, *Transportation Science*, 35, 425-434 .

Lacomme, P., Prins, C. and Ramdane-Cherif, W. (2004a) Competitive memetic algorithms for arc routing problems, *Annals of Operations Research*, 131, 159-185.

Lacomme, P., Prins, C. and Tanguy, A. (2004b) First competitive ant colony scheme for the CARP. *Research Report LIMOS/RR-04-21*.

Mei, Y., Tang, K. and Yao, X. (2009) A global repair operator for capacitated arc routing problem, *IEEE Transactions on Systems*, *Man*, *and Cybernetics*, *Part B: Cybernetics*, 39, 723-734.

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

Polacek, M., Doerner, K. F., Hartl, R.F. and Maniezzo, V. (2008) A variable neighborhood search for the capacitated arc routing problem with intermediate facilities, *Journal of Heuristics*, 14, 405-423 .

Santos, L., Coutinho-Rodrigues, J. and Current, J.R. (2010) An improved ant colony optimization based algorithm for the capacitated arc routing problem, *Transportation Research Part B*, 44, 246-266.

Tang, K., Mei, Y. and Yao, X. (2009) Memetic algorithm with extended neighborhood search for capacitated arc routing problems, *IEEE Transactions on Evolutionary Computation*, 13, 1151-1166.

Ting, C.J. and Chen, C.H. (2013) A multiple ant colony optimization algorithm for the capacitated location routing problem, *International Journal of Production Economics*, 141, 34-44.

Ulusoy, G. (1985) The fleet size and mix problem for capacitated arc routing, *European Journal of Operational Research*, 22(3): 329-337.

Usberti, F.L., França, P.M., and França, A.L.M. (2013) GRASP with evolutionary path-relinking for the capacitated arc routing problem, *Computers & Operations Research*, 40, 3206-3217.

Wøhlk, S. (2005) Contributions to arc routing, Ph.D. dissertation, University of Southern Denmark, Arhus, Demark.

Wøhlk, S. (2008) A decade of capacitated arc routing, In *The vehicle routing problem: Latest advances and new challenges*, Golden, B., Raghavan, S. and Wasil, E. (eds.), Springer, New York, 29-48.