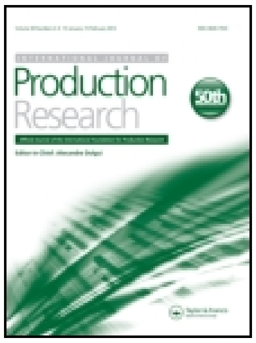

International Journal of Production Research

ISSN: 0020-7543 (Print) 1366-588X (Online) Journal homepage: http://www.tandfonline.com/loi/tprs20

A memetic algorithm with iterated local search for the capacitated arc routing problem

Tiantang Liu , Zhibin Jiang & Na Geng

To cite this article: Tiantang Liu , Zhibin Jiang & Na Geng (2013) A memetic algorithm with iterated local search for the capacitated arc routing problem, International Journal of Production Research, 51:10, 3075-3084, DOI: 10.1080/00207543.2012.753165

To link to this article: http://dx.doi.org/10.1080/00207543.2012.753165

International Journal of Production Research, 2013
Vol. 51, No. 10, 3075–3084, http://dx.doi.org/10.1080/00207543.2012.753165

A memetic algorithm with iterated local search for the capacitated arc routing problem

Tiantang Liu, Zhibin Jiang* and Na Geng

Department of Industrial Engineering and Logistics Management, School of Mechanical Engineering, Shanghai Jiao Tong University, 800 Dong Chuan Road, Shanghai 200240, P. R. China

The capacitated arc routing problem (CARP) is a difficult vehicle routing problem where, given an undirected graph, the objective is to minimise the total cost of all vehicle tours that serve all required edges under vehicle capacity constraints. In this paper, a memetic algorithm with iterated local search (MAILS) is proposed to solve this problem. The proposed MAILS incorporates a new crossover operator, i.e., the longest common substring crossover (LCSX), an iterated local search (ILS), and a perturbation mechanism into the framework of the memetic algorithm (MA). The proposed MAILS is evaluated on the CARP benchmark instances and computational results show that the MAILS is very competitive.

Keywords: capacitated arc routing problem; memetic algorithm; iterated local search; longest common substring crossover

1. Introduction

Vehicle routing problems are widespread in logistics and distribution. To be specific, there are two classes of vehicle routing problems, one is node routing problems, such as the capacitated vehicle routing problem (VRP) and the travelling salesman problem (TSP), and the other is arc routing problems, such as the capacitated arc routing problem (CARP), the rural postman problem (RPP), and the Chinese postman problem (CPP). In real environments, the number of vehicles is often more than one and the capacity of vehicles is limited. Therefore, capacitated cases like the CARP are more realistic to model applications such as winter gritting, street sweeping, garbage collection, mail delivery, meter reading, school bus routing, and other pickup or delivery problems along the streets of a road network.

In this paper, we focus on the CARP, which is introduced by Golden and Wong (1981) and can be described as follows: given an undirected graph $G = (V, E)$ with a set V of n nodes, an undirected edge set E, where $c_{ij} = c_{ji}$ (≥ 0) is the cost (length) of an edge $(i,j) \in E$, and $E_R \subseteq E$ is the set of t required edges (tasks). A fleet of homogeneous vehicles with capacity Q are stationed at the depot $D \subseteq V$. The CARP is to determine a set of routes in such a way that: (1) each vehicle route starts and ends at the depot; (2) each required edge is served exactly once; (3) the total demand of each route served by that vehicle does not exceed vehicle capacity Q; and (4) the total routing cost is minimised. The number of vehicles is a decision variable.

The CARP is an NP-hard problem; therefore, heuristics and meta-heuristics have been the mainstream approach to solve the CARP. For example, augment-merge (Golden and Wong 1981), path-scanning (Golden, Dearmon, and Baker 1983), parallel-insert (Chapleau et al. 1984), construct-strike (Pearn 1989), augment-insert (Pearn 1991) are simple constructive heuristics; and route-first, cluster-second (Ulusoy 1985) and cluster-first, route-second (Benavent et al. 1990) are two-phase constructive heuristics. In the last decade, some meta-heuristics have been proposed, such as simulated annealing (Eglese 1994), tabu search (Hertz, Laporte, and Mittaz 2000; Brandão and Eglese 2008), variable neighborhood search (Hertz and Mittaz 2001; Polacek et al. 2008), guided local search (Beullens et al. 2003), memetic algorithms (Lacomme, Prins, and Ramdane-Cherif 2001; Lacomme, Prins, and Ramdane-Cherif 2004) and ant colony optimization (Santos, Coutinho-Rodrigues, and Current 2010) and greedy randomised adaptive search procedure with evolutionary path-relinking (Usberti, França and França 2012), etc. A recent review can be found in Liu et al. (2008) and Corberán and Prins (2010).

Our proposed approach in this paper is a memetic algorithm with iterated local search (MAILS), which embeds a new crossover, a perturbation mechanism and an iterated local search (ILS) into the framework of memetic algorithm (MA). In combinatorial optimization, the traditional genetic algorithm (GA) is not so competitive compared to other metaheuristics like tabu search, while recently it has been shown that MA (i.e., GA enhanced with local search) is an

*Corresponding author. Email: zbjiang@sjtu.edu.cn efficient metaheusitic for many classical optimization problems, such as scheduling problems (Franca, Tin, and Buriol 2006, Chiang and Fu 2008, Naderi, Zandieh, and Ghomi 2009) and routing problems (Fung, Liu, and Jiang 2013; Lacomme, Prins, and Ramdane-Cherif 2004; Prins 2004; Mendoza et al. 2010; Nagata, Bräysy, and Dullaert 2010).

The structure of the remainder of this paper is as follows: Section 2 gives the general framework and key components of the proposed MAILS. Computational experiments on benchmark data are presented in Section 3, and finally, conclusions are given in Section 4.

2. Memetic algorithm with iterated local search

The memetic algorithm introduced by Moscato (1989), also known as hybrid genetic algorithm or genetic local search, is a combination of a population-based global search GA and an individual-based local search. Several characters of our MAILS are: (1) a novel crossover, longest common substring (LCSX), is proposed and two crossover operators, order crossover (OX) and LCSX, are used simultaneously to keep the balance of exploration and exploitation abilities; (2) infeasible solutions are allowed during the local search process; (3) after the child is improved by local search with a fixed probability p_m, it is further improved by iterated local search if it's cost is close enough to that of the best parent solution in the parent population; (4) one chromosome P_r is selected in the parent population using binary tournament replacement, described in Section 2.6, and it is replaced by one child C if the child C is not a clone of any other chromosome than P_r in the parent population, else a double swap perturbation is implemented on the child C to diversify the population.

The proposed MAILS includes two phases: a main phase and a restart phase. The general framework of the MAILS can be summarised in Figure 1. Several main components are described in detail in Section 2.1–2.6.

2.1 *Chromosomes structure and evaluation*

To describe the tasks clearly, each required edge is identified by being marked a task number instead of one pair of nodes. Each edge $u \in E$ has a tail (start node) $a(u)$, a head (end node) $b(u)$, and a traversing (deadheading) cost $tc(u)$. Each required edge (task) $u \in E_R$ has a demand $d(u)$, a serving cost $sc(u)$, and an inverse mark $inv(u)$. Task $inv(u)$ and u have the same traversing demand, and serving costs. Note that each edge task $u \in E_R$ can be served in either direction, i.e., only one of task u and $inv(u)$ is served.

Inspired by Lacomme, Prins, and Ramdane-Cherif (2004), our chromosome T is a permutation of t required edges (tasks), without route delimiters. Implicit shortest paths are between consecutive tasks. they can be viewed as a RPP or a giant tour. This kind of chromosome representation is adopted because: (1) the encoding is simple, and classical GA operators for the travelling salesman problem can be reused; (2) the solution can be kept feasible in the reproduction process; (3) the search space of the proposed MAILS is a smaller RPP tour than the CARP solution space; (4) an optimal possible CARP solution can be extracted from a chromosome by the partition procedure described below; (5) the indirect encoding is so flexible that it is suitable for extended problems including new constraints and objectives.

Step 1 The initial population is constructed using two heuristics and random generation.

Step 2 Each iteration selects two parents P1 and P2 randomly and then two children C1 and C2 are obtained using either OX or LCSX, and only one of them is randomly selected as child *C*.

Step 3 The child *C* undergoes the local search in a given probability p_m and is further improved by the iterated local search if it's cost is close enough to the best parent chromosome. During the local search process, infeasible solutions are allowed.

Step 4 Two chromosomes are selected from the parent population and the worse one P_r is replaced by the child *C* if the child *C* is not a clone of the other chromosomes in the parent population, else a double swap perturbation is implemented on the child *C* to diversify the population.

Step 5 The main phase stops after a maximum number of iterations (*ni*). After that, the restart procedure is implemented *nr* times, where the first, third, fifth… and (*ps*-1)th chromosomes are kept, and others are replaced by new, randomly generated chromosomes, and then the main phase is repeated but with a higher local search probability p_r. That is to say, the total number of iterations is $(1+nr) \times ni$.

Figure 1. General framework of MAILS.

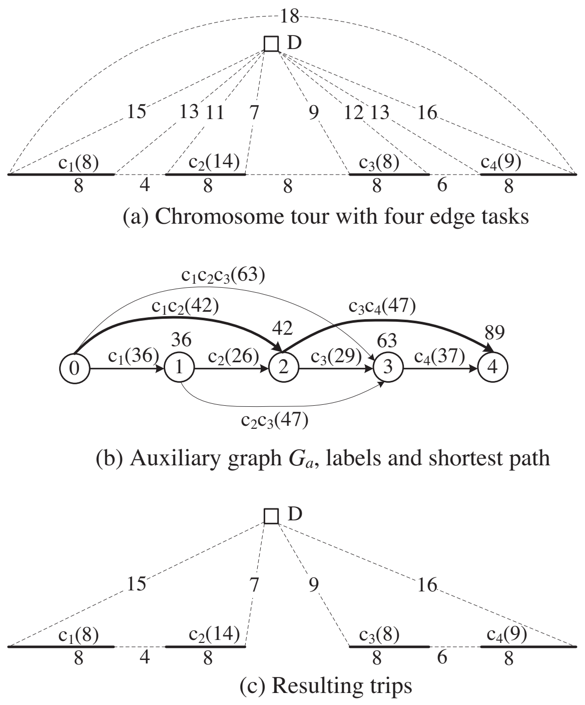

Figure 2. Example of Partition.

Under this kind of chromosomes structure, the chromosome must be converted into a CARP solution by a partition (Ulusoy 1985) procedure which corresponds to chromosome decoding and can evaluate the performance of each chromosome. The fitness is the total cost of this solution.

Given a chromosome $T = (c_1, c_2, ..., c_t)$ where t corresponds to the number of tasks, the partition procedure works on an auxiliary directed acyclic graph $G_a = (X, Y, Z)$, where X is a set of $t + 1$ vertex indices from a dummy node 0 to t. Y is a set of arcs where one arc $(i,j) \in Y$ means that a trip serving tasks subsequence $(c_{i+1}, c_{i+2}, ..., c_j)$ is feasible in terms of capacity, i.e., $\text{load}(i + 1, j) \leq Q$ where $\text{load}(i + 1, j)$ is the load of the trip. Z is the set of the weight of arcs where one weight z_{ij} corresponds to the total cost of one vehicle to serve task subsequence $(c_{i+1}, c_{i+2}, ..., c_j)$. The optimal partitioning of the chromosome T corresponds to a shortest path from node 0 to node t in G_a. Thus, this problem is a shortest path problem (SPP), which can be solved in pseudo-polynomial time based on Bellman's algorithm.

Consider one example of vehicle capacity $Q = 30$ and four edge tasks with their respective demands being 8, 14, 8, and 9. Figure 2(a) shows the chromosome tour $T = (c_1, c_2, c_3, c_4)$ with demands in brackets. Thin dotted lines represent shortest paths between any two nodes, the numbers under $t = 4$ tasks are the serving costs, and D represents the depot. The partition procedure builds an auxiliary graph G_a with $t + 1$ nodes indexed from 0 to t, as shown in Figure 2(b). Arc (0, 1) represents the trip (0, c_1, 0), where the first 0 and the last 0 correspond to the depot. A shortest path from node 0 to node t in G_a (bold) indicates the optimal partitioning of T: two trips with total cost of 89. The resulting CARP solution is the trip (0, c_1, c_2, 0) with a cost of 42 the trip (0, c_3, c_4, 0) with a cost of 47, as shown in Figure 2(c).

2.2 *Initial population*

The population P is composed of *ps* chromosomes. The initial population consists of two good (low-cost) chromosomes and *ps*-2 random chromosomes. To be specific, two good chromosomes (RPP tours) P_1 and P_2 are constructed by using Frederickson heuristics (Frederickson 1979) and path-scanning (PS) (Golden, Dearmon, and Baker 1983), respectively.

PS generates a CARP solution which is concatenated into a permutation of tasks (giant RPP tour). The other methods generate directly *ps*-1 giant tours. All *ps* tours are then converted into *ps* solutions by the Partition procedure, and the cost of each solution is obtained. Then each solution is concatenated into one chromosome where route delimiters (depots) are removed, and all chromosomes are stored using an array in increasing cost order.

P1:	**1**	**4**	5	**7**	**9**	10	12	3	6	8
P2:	13	2	10	8	6	15	**7**	**9**	**1**	**4**
C1:	**1**	**4**	13	**7**	**9**	2	10	8	6	15
C2:	5	10	12	3	6	8	**7**	**9**	**1**	**4**

Figure 3. Example of LCSX.

2.3 *Selection and crossover*

Two parents P1 and P2 are selected randomly, and then two crossover operators are used to produce next better offspring: one is a classic crossover operator, order crossover (OX), and the other is a novel crossover, longest common substring crossover (LCSX).

A good chromosome often has a small sum of deadheading costs (shortest paths) between consecutive tasks, i.e. in a good chromosome, the tasks with least deadheading costs in between should be adjacent. Likewise, in two good parent chromosomes, there should be some common substrings with least deadheading costs between consecutive tasks. Therefore, the proposed LCSX preserves only the longest common substrings in the parents. For two parent chromosomes, the longest common substring can be found using dynamic programming in $O(t^2)$, where t is the length of the chromosomes, i.e. the number of tasks.

Figure 3 gives one sample of LCSX, where $t = 10$ tasks are undirected, and each edge task f is converted into two opposite arc tasks with $inv(f) = 10 + f$, i.e., $inv(1)$=11, $inv(2)$=12, etc. The chromosomes (task sequences) of two parents P1 and P2 are (1, 4, 5, 7, 9, 10, 12, 3, 6, 8) and (13, 2, 10, 8, 6, 15, 7, 9, 1, 4), respectively. Then, the longest common substrings are (1, 4) and (7, 9), and so, task substrings (1, 4) and (7, 9) in P1 and P2 are copied into two child chromosomes C1 and C2, respectively and without changing positions. Then, P2 is scanned from 1 onwards to t, to fill C1 in parallel with the tasks missed in C1. The C2 can be obtained by exchanging the roles of P1 and P2. Finally, only one child C, randomly selected between C1 and C2, is kept.

2.4 *Local search*

A local search (LS) is adopted with a fixed probability p_m in our MAILS, to produce a better offspring after each crossover. The LS works on a CARP solution obtained by implementing the partition procedure on the child C, because if it operates directly on the chromosome C without route delimiters, a large amount of time will be spent to evaluate each move of it.

Let tasks i and j be served after tasks f and g in their respective routes. All move types are described below.

- M1: move task f after task g.
- M2: move two consecutive tasks (f, i) after task g.
- M3: swap task f and g.
- M4: swap task f and (g, j).
- M5: swap task (f, i) and (g, j).
- M6: 2-opt moves.

The LS scans each pair of tasks (f, g) in $o(t^2)$, and each iteration of the LS implements M1–M6 and stops when it finds the first improving move, and then the solution is updated and the next iteration is continued until all pairs of tasks are scanned. The whole M1–M6 process for each pair of tasks are repeated as long as the solutions can be further improved.

There are several points to be noted in the LS. First, each type of move is implemented in the same route or in two different routes. Second, in M1–M5, if a task f is moved to another position, it can appear in either as f or $inv(f)$. Third, in M1 and M2, g can be the start depot of its route. Finally, some routes are removed if they become empty.

To increase the diversification, our LS is allowed to explore capacity infeasible solutions. A penalised fitness function instead of the total cost of one solution is used to evaluate each solution generated from moves of LS. It is defined as follows:

$$\text{fit}(s) = tcost(s) + \beta * tvio(s) \quad (1)$$

where *tcost*(*s*) and *tvio*(*s*) are the total cost and the total capacity violation of the solution *s*, respectively, and β is the penalty coefficient. Initially β is set to 5, and it will be divided by 2 if solutions of consecutive 10 steps are capacity feasible, otherwise it will be multiplied by 2 if solutions of consecutive 10 steps are capacity infeasible. This kind of dynamic adjustment can make the LS generate more feasible solutions.

Note that during the LS, the resulting solution is set as the best feasible solution, i.e. the solution with the lowest *tcost* (*s*) and *tvio*(*s*) = 0, instead of the final solution with the lowest fitness value. The final best feasible solution of LS is converted into a chromosome by concatenating these routes and excluding route delimiters (depots). Then, the chromosome is converted into a solution by the optimal partition which sometimes can bring a better solution for the same chromosome.

2.5 *Iterated local search*

The ILS is formally described in Lourenço Martin, and Stützle (2003) and our ILS can be summarised as Algorithm 1. Given the child chromosome (tour) *C*, which corresponds a solution s_c, if the cost of s_c is close enough to (less than α = 1.005 times) that of the best chromosome of the parent population, a random perturbation is implemented on the child *C* to generated a new chromosome *C*', which is partitioned into solution s_c' and improved by local search of Section 2.4 to produce a good solution *s*", if the cost of *s*" is less than the one of s_c, then the solution *s*" is converted into a chromosome tour *C*", and *C* and s_c are updated to *C*" and s_c", respectively, and the perturbation is repeated *np* times. To avoid high computational cost, the *np* is set as 3 after preliminary experiments. In the ILS, the perturbation mechanism is the single swap and the double swap where the former randomly swaps one task with anothers, and the latter randomly swaps two consecutive tasks with another two.

Algorithm 1. The ILS procedure

```
if($s_c$.cost< $\alpha^*$ bestsoln.cost)
  for $i$: 1 to $np$ do
    $C'_1$ =SingleSwapPerturbation($C$)
    $s'_{c_1}$ = Partition($C'_1$)
    $s''_{c_1}$ = LocalSearch($s'_{c_1}$)
    if($s''_{c_1}$ cost< = $s_c$.cost) then $s_c$.cost =$s''_{c_1}$.cost; $C$=$C''_1$; endif
    $C'_2$ =DoubleSwapPerturbation($C$)
    $s'_{c_2}$ =Partition($C'_2$)
    $s''_{c_1}$=LocalSearch($s'_{c_2}$)
    if($s''_{c_1}$.cost<= $s_c$.cost) then $s_c$.cost =$s''_{c_1}$.cost; $C$=$C''_1$; endif
  endfor
endif
```

2.6 *Replacement and restart*

Inspired by binary tournament selection, we propose a new replacement method which can be called binary tournament replacement, i.e., two chromosomes are selected from the parent population and the worse one P_r is replaced by the child *C* if the child *C* is not identical as any other chromosomes of the parent population. After replacement, the *ps* chromosomes are stored in increasing cost order again.

In the replacement process, if the child *C* is a clone of any chromosome other than P_r of the parent population, one random double swap perturbation described in Section 2.5 is implemented on child *C* to avoid a clone and diversify the population. If the child *C* is still a clone (chromosome with the same cost) of another chromosome of the parent population, then it replaces the clone.

The proposed MAILS includes two phases: a main phase and a restart phase. The main phase stops after a maximum number of iterations (*ni*). After that, the restart procedure is implemented *nr* times, where the first, third, fifth … and (*ps*-1)th chromosomes of a population are kept, and the rest are replaced by new, randomly generated chromosomes. Then, the main phase is repeated but with a higher local search probability p_r. That is to say, the total number of iterations is $(1+nr)\times ni$. It is worth noting that the proposed partial replacement procedure in the restart phase can reserve good chromosomes and increase the diversity of population because any two adjacent chromosomes in the sorted population are often close to clones after many iterations of each phase.

3. Computational experiments

3.1 *Parameters settings*

Our MAILS is implemented in C and executed on an Intel (R) Pentium (R) Dual 1.8-GHz PC under Windows XP.

The three most studied CARP benchmark instances (*gdb*, *val* and *egl* files) are available from http://www.uv.es/≃belengue/carp.html. The *gdb* set is 23 small size instances with 7–27 nodes and 11–55 edges; the *val* set includes 34 medium size instances with 24–50 nodes and 34–97 edges and the *egl* set contains 24 large size instances with 77–140 nodes and 98∼190 edges. In the *gdb* and *val* sets, all the edges are required edges (tasks), and the *egl* set has 51–190 required edges and some non-required edges.

Through preliminary experiments, we determine the parameters of the MAILS as follows: the population size *ps* is 30, the maximum times *try_max* to generate each random non-clone chromosome is 10, the local search probability p_m and p_r in the main phase and restart phase are 0.2 and 0.4, respectively; the stopping criteria of the main phase is the maximum iteration number $ni = 10000$, and the maximum number of restarts *nr* is 10.

3.2 *Computational results*

In this section, we compare our MAILS with the existing four best performing CARP metaheuristics, which are tabu search algorithm (TSA) (Brandão and Eglese 2008), variable neighbourhood search (VNS) (Polacek et al. 2008), ant colony optimisation (ACO) (Santos, Coutinho-Rodrigues, and Current 2010) and GRASP (Usberti, França and França 2012). Except TSA, most of the compared metaheuristics are stochastic search techniques.

The results of the three benchmark sets of CARP problems (*gdb*, *val* and *egl*) are presented in Tables 1–3, where we present our best solutions and average solutions over 10 runs on 81 benchmark test instances. The best solutions from other metaheuristics are also provided in Tables 1–3 but without average solutions due to space limitations. Brandão and Eglese (2008) adopt a deterministic tabu search algorithm (TSA) and provide their best solutions over one run, the best solutions from VNS are over 10 runs on a Pentium IV at 3.6 GHz, the best solutions obtained by ACO are over

Table 1. Results for the *gdb* set.

						Best cost				
Name	|V|	$\|E_R\|$	LB	MAILS_{as}	Sec.	TSA	ACO	GRASP	BKS	MAILS_{bs}
gdb1	12	22	316	316	0.00	**316**	**316**	**316**	**316**	**316**
gdb2	12	26	339	339	0.37	**339**	**339**	**339**	**339**	**339**
gdb3	12	22	275	275	0.10	**275**	**275**	**275**	**275**	**275**
gdb4	11	19	287	287	0.01	**287**	**287**	**287**	**287**	**287**
gdb5	13	26	377	377	0.07	**377**	**377**	**377**	**377**	**377**
gdb6	12	22	298	298	0.08	**298**	**298**	**298**	**298**	**298**
gdb7	12	22	325	325	0.02	**325**	**325**	**325**	**325**	**325**
gdb8	27	46	348	349	27.40	**348**	**348**	**348**	**348**	**348**
gdb9	27	51	303	303	24.02	**303**	**303**	**303**	**303**	**303**
gdb10	12	25	275	275	0.03	**275**	**275**	**275**	**275**	**275**
gdb11	22	45	395	395	0.63	**395**	**395**	**395**	**395**	**395**
gdb12	13	23	458	458	4.86	**458**	**458**	**458**	**458**	**458**
gdb13	10	28	536	536	5.18	540	**536**	**536**	**536**	**536**
gdb14	7	21	100	100	0.10	**100**	**100**	**100**	**100**	**100**
gdb15	7	21	58	58	0.00	**58**	**58**	**58**	**58**	**58**
gdb16	8	28	127	127	0.29	**127**	**127**	**127**	**127**	**127**
gdb17	8	28	91	91	0.01	**91**	**91**	**91**	**91**	**91**
gdb18	9	36	164	164	0.06	**164**	**164**	**164**	**164**	**164**
gdb19	8	11	55	55	0.00	**55**	**55**	**55**	**55**	**55**
gdb20	11	22	121	121	0.52	**121**	**121**	**121**	**121**	**121**
gdb21	11	33	156	156	0.44	**156**	**156**	**156**	**156**	**156**
gdb22	11	44	200	200	1.80	**200**	**200**	**200**	**200**	**200**
gdb23	11	55	233	235	15.06	235	235	**233**	**233**	**233**

Table 2. Results for the *val* set.

						Best cost					
Name	|V|	$\|E_R\|$	LB	$MAILS_{as}$	Sec.	TSA	VNS	ACO	GRASP	BKS	$MAILS_{bs}$
val1A	24	39	173	173	0.06	**173**	**173**	**173**	**173**	**173**	**173**
val1B	24	39	173	173	4.98	**173**	**173**	**173**	**173**	**173**	**173**
val1C	24	39	245	245	1.91	**245**	**245**	**245**	**245**	**245**	**245**
val2A	24	34	227	227	0.07	**227**	**227**	**227**	**227**	**227**	**227**
val2B	24	34	259	259	0.09	**259**	**259**	**259**	**259**	**259**	**259**
val2C	24	34	457	457	4.24	**457**	**457**	**457**	**457**	**457**	**457**
val3A	24	35	81	81	0.03	**81**	**81**	**81**	**81**	**81**	**81**
val3B	24	35	87	87	0.41	**87**	**87**	**87**	**87**	**87**	**87**
val3C	24	35	138	138	2.11	**138**	**138**	**138**	**138**	**138**	**138**
val4A	41	69	400	400	5.65	**400**	**400**	**400**	**400**	**400**	**400**
val4B	41	69	412	412	5.93	**412**	**412**	**412**	**412**	**412**	**412**
val4C	41	69	428	428	17.22	**428**	**428**	**428**	**428**	**428**	**428**
val4D	41	69	526	531	82.41	**530**	**530**	**530**	**530**	**530**	**530**
val5A	34	65	423	423	2.12	**423**	**423**	**423**	**423**	**423**	**423**
val5B	34	65	446	446	1.33	**446**	**446**	**446**	**446**	**446**	**446**
val5C	34	65	473	474	12.96	**474**	**474**	**474**	**474**	**474**	**474**
val5D	34	65	573	583	142.62	583	**575**	577	**581**	**575**	581
val6A	31	50	223	223	0.82	**223**	**223**	**223**	**223**	**223**	**223**
val6B	31	50	233	233	13.62	**233**	**233**	**233**	**233**	**233**	**233**
val6C	31	50	317	317	10.33	**317**	**317**	**317**	**317**	**317**	**317**
val7A	40	66	279	279	1.53	**279**	**279**	**279**	**279**	**279**	**279**
val7B	40	66	283	283	0.36	**283**	**283**	**283**	**283**	**283**	**283**
val7C	40	66	334	334	45.32	**334**	**334**	**334**	**334**	**334**	**334**
val8A	30	63	386	386	1.94	**386**	**386**	**386**	**386**	**386**	**386**
val8B	30	63	395	395	7.14	**395**	**395**	**395**	**395**	**395**	**395**
val8C	30	63	518	527	135.66	529	**521**	**521**	522	**521**	523
val9A	50	92	323	323	38.69	**323**	**323**	**323**	**323**	**323**	**323**
val9B	50	92	326	326	38.32	**326**	**326**	**326**	**326**	**326**	**326**
val9C	50	92	332	332	44.59	**332**	**332**	**332**	**332**	**332**	**332**
val9D	50	92	385	391	164.78	391	**389**	391	391	**389**	391
val10A	50	97	428	428	53.97	**428**	**428**	**428**	**428**	**428**	**428**
val10B	50	97	436	436	47.97	**436**	**436**	**436**	**436**	**436**	**436**
val10C	50	97	446	446	40.28	**446**	**446**	**446**	**446**	**446**	**446**
val10D	50	97	525	531	189.21	530	**526**	**526**	527	**526**	527

15 runs on a Pentium III at 1.0 GHz, and Usberti, França and França (2012) present the best solutions by running their GRASP 15 times on an Intel Core 2 Quad at 3.0 GHz.

For each instance in Tables 1–3, the columns headed |V| and $|E_R|$ contain the number of vertices and required edges, respectively, the lower bound (LB) are given in column 4, and the average solutions ($MAILS_{as}$) and computing times of the MAILS are respectively shown in columns 5–6, The columns headed TSA, VNS, ACO, GRASP, and $MAILS_{bs}$ indicate the best solutions of each metaheuristic, and the column headed BKS lists the best known solutions reported in recent publications. Note that the VNS has not been applied to *gdb* set. In Tables 1–3, if the best solution of each metaheuristic is equivalent or superior to that of the BKS then it is shown in bold.

Since our MAILS and most of the above metaheuristics are stochastic algorithms, we present the comparison of these algorithms over the quality of both the best solutions and average solutions in Tables 4–6. Table 4 presents the number of the best solutions found by each metaheuristic. The results show that the MAILS is competitive with existing metaheuristics. The MAILS finds all the optimal solutions on 23 small-size instances in *gdb* set, and the best solutions on 30 out of 34 median size instances in *val* set and 17 out of 24 large size instances in *egl* set. Moreover, the MAILS finds two new best solutions which are underlined in Table 3.

Tables 5 and 6 provide the average percentage deviation of each algorithm's best solutions and average solutions from the best known solutions on each benchmark set, respectively. TSA is excluded from Table 6 because it's a deterministic algorithm. We can see that, on average, our MAILS performs better than TSA, VNS, ACO, and slightly worse than GRASP.

Table 7 gives the computing times (in seconds) of each metaheuristic on the three benchmark sets. To make fair comparison on the computing times, the original times were multiplied by the scaling factor which is the central

Table 3. Results for the *egl* set.

						Best cost					
Name	|V|	$\|E_R\|$	LB	MAILS_{as}	Sec.	TSA	VNS	ACO	GRASP	BKS	MAILS_{bs}
egl-e1-A	77	51	3548	3548	1.80	**3548**	**3548**	**3548**	**3548**	**3548**	**3548**
egl-e1-B	77	51	4498	4511	33.44	4533	**4498**	**4498**	**4498**	**4498**	**4498**
egl-e1-C	77	51	5542	5595	66.53	**5595**	**5595**	**5595**	**5595**	**5595**	**5595**
egl-e2-A	77	72	5011	5018	100.26	**5018**	**5018**	**5018**	**5018**	**5018**	**5018**
egl-e2-B	77	72	6280	6344	121.15	6343	6321	**6317**	**6317**	**6317**	**6317**
egl-e2-C	77	72	8234	8354	125.32	8347	**8335**	**8335**	**8335**	**8335**	**8335**
egl-e3-A	77	87	5898	5902	148.53	5902	**5898**	**5898**	**5898**	**5898**	**5898**
egl-e3-B	77	87	7697	7801	160.23	7816	**7775**	7777	7777	**7775**	7777
egl-e3-C	77	87	10163	10313	145.30	10309	**10292**	**10292**	**10292**	**10292**	**10292**
egl-e4-A	77	98	6395	6475	214.35	6473	6446	6456	**6444**	**6444**	6456
egl-e4-B	77	98	8884	9026	253.71	9063	9004	**8990**	9002	**8990**	8991
egl-e4-C	77	98	11427	11645	320.16	11627	11652	**11624**	11626	**11624**	**11587**
egl-s1-A	140	75	5014	5019	104.81	5072	**5018**	**5018**	**5018**	**5018**	**5018**
egl-s1-B	140	75	6379	6435	150.93	**6388**	**6388**	**6388**	**6388**	**6388**	**6388**
egl-s1-C	140	75	8480	8518	124.56	8535	**8518**	**8518**	**8518**	**8518**	**8518**
egl-s2-A	140	147	9824	9959	708.29	10038	9944	**9895**	9903	**9895**	**9895**
egl-s2-B	140	147	12968	13236	863.10	13178	**13167**	13194	13169	**13167**	**13150**
egl-s2-C	140	147	16353	16535	974.85	16505	16491	16461	**16442**	**16442**	**16442**
egl-s3-A	140	159	10143	10333	936.22	10451	10259	10249	**10221**	**10221**	10261
egl-s3-B	140	159	13616	13890	1086.79	13981	13751	13786	**13694**	**13694**	13786
egl-s3-C	140	159	17100	17304	997.40	17346	17299	17269	**17221**	**17221**	**17221**
egl-s4-A	140	190	12143	12428	1436.82	12462	12375	12324	**12297**	**12297**	12341
egl-s4-B	140	190	16093	16452	1412.34	16490	16353	16428	**16333**	**16333**	16345
egl-s4-C	140	190	20375	20761	1626.60	20733	20640	20595	**20563**	**20563**	**20556**

Table 4. Number of the best solutions found by each metaheuristic.

	TSA	VNS	ACO	GRASP	MAILS
gdb	21	-	22	23	23
val	30	34	32	30	30
egl	4	12	14	19	17
total	56	-	68	72	70

Table 5. Average deviation of each metaheuristic's best solutions from BKS.

	TSA	VNS	ACO	GRASP	MAILS
gdb	0.10%	-	0.03%	0.00%	0.00%
val	0.19%	0.03%	0.03%	0.09%	0.09%
egl	0.71%	0.21%	0.16%	0.01%	0.06%

Table 6. Average deviation of each metaheuristic's average solutions from BKS.

	TSA	VNS	ACO	GRASP	MAILS
gdb	-	-	0.17%	0.18%	0.05%
val	-	0.10%	0.15%	0.22%	0.19%
egl	-	0.53%	0.57%	0.44%	0.56%

Table 7. Average computing time (s).

CPU factor	TSA	VNS	ACO	GRASP	MAILS
	1.4/1.8	3.6/1.8	1.0/1.8	3.0/1.8	1.0
	1.4/1.8	3.6/1.8	1.0/1.8	3.0/1.8	1.0
gdb	1.94	-	1.89	8.50	3.52
val	15.71	87.89	14.06	102.5	32.90
egl	226.64	1006.44	279.44	1331	504.73

processing unit (CPU) frequency ratio between the original computer and ours. Table 7 shows that the MAILS is relatively time consuming but is still acceptable, with less time than that of GRASP and the VNS. In conclusion, the comparisons of these algorithms over the quality of the best solutions and average solutions, and computing times both support the good performance of our MAILS.

The good results can be explained as follows: (1) the novel LCSX captures the essence of good solutions, i.e. tasks with the shortest deadheading path in between should be adjacent; in addition, two crossover operators are used simultaneously to keep the balance of exploration and exploitation abilities; (2) the proposed ILS further exploits promising solutions, which provides a strong intensification; and (3) allowance of infeasible solutions during the local search process, the binary tournament replacement, the perturbation mechanism and the second, the fourth, the sixth … and the ps^{th} chromosomes replacements in the beginning of each restart phase increase the population diversity.

4. Conclusions

This paper presents a memetic algorithm with iterated local search for the capacitated arc routing problem. In the proposed MAILS, a novel longest common substring crossover operator and an iterated local search are proposed to be embedded into the memetic algorithm, which increase, the intensification. In addition, we allow infeasible solutions during the local search process, and present the perturbation mechanism, the binary tournament replacement, and the second, the fourth, the sixth … and the ps^{th} chromosomes' replacements in the beginning of each restart phase, which all increase the diversification.

A large amount of benchmark instances are tested and the results show that the MAILS is competitive with existing metaheuristics and that the computing times are reasonable. One point needs further study: the ILS is time consuming, which is only occasionally implemented with limited iterations each time. A better ILS should be proposed to balance the solution quality and computing times.

Acknowledgements

This work was supported by the National Natural Science Foundation of China under research grant number 70872077 and the National Natural Science Foundation of China – Research Grants Council of Hong Kong (NSFC–RGC) joint research projects (grant number 70831160527). We thank the editor and the anonymous referees for their valuable comments.

References

Benavent, E., V. Campos, A. Corberán, and E. Mota. 1990. "The capacitated arc routing problem. A heuristic algorithm." *Qüestiió* 14 (1–3): 107–122.

Beullens, P., L. Muyldermans, D. Cattrysse, and D. Van Oudheusden. 2003. "A guided local search heuristic for the capacitated arc routing problem." *European Journal of Operational Research* 147 (3): 629–643.

Brandão, J., and R. Eglese. 2008. "A deterministic tabu search algorithm for the capacitated arc routing problem." *Computers & Operations Research* 35 (4): 1112–1126.

Chapleau, L., J.A. Ferland, G. Lapalme, and J.M. Rousseau. 1984. "A parallel insert method for the capacitated arc routing problem." *Operations Research Letters* 3 (2): 95–99.

Chiang, T.C., and L.C. Fu. 2008. "A rule-centric memetic algorithm to minimize the number of tardy jobs in the job shop." *International Journal of Production Research* 46 (24): 6913–6931.

Corberán, A., and C. Prins. 2010. "Recent results on arc routing problems: An annotated bibliography." *Networks* 56 (1).

Eglese, R.W. 1994. "Routeing winter gritting vehicles ." *Discrete Applied Mathematics* 48 (3): 231–244.

Franca, P.M., G. Tin, Jr, and L.S. Buriol. 2006. "Genetic algorithms for the no-wait flowshop sequencing problem with time restrictions ." *International Journal of Production Research* 44 (5): 939–957.

Frederickson, G. 1979. "Approximation algorithms for some postman problems." *Journal of the ACM* 26 (3): 538–554.

Fung, R.Y.K., Liu, R. & Z. Jiang, 2013. “A memetic algorithm for the open capacitated arc routing problem.” *Transportation Research Part E: Logistics and Transportation Review*. doi: 10.1016/j.tre.2012.11.003

Golden, B.L., and R.T. Wong. 1981. “Capacitated arc routing problems.” *Networks* 11 (3): 305–315.

Golden, J.S., J.S. Dearmon, and E.K. Baker. 1983. “Computational experiments with algorithms for a class of routing problems.” *Computers & Operations Research* 10 (1): 47–59.

Hertz, A., G. Laporte, and M. Mittaz. 2000. “A tabu search heuristic for the capacitated arc routing problem.” *Operations Research* 48 (1): 129–135.

Hertz, A., and M. Mittaz. 2001. “A variable neighborhood descent algorithm for the undirected capacitated arc routing problem.” *Transportation Science* 35 (4): 425–434.

Lacomme, P., C. Prins, and W. Ramdane-Cherif. 2001. A genetic algorithm for the capacitated arc routing problem and its extensions. *In* Boers, E.J.W. ed. *Applications of evolutionary computing*. Springer, 473–483.

Lacomme, P., C. Prins, and W. Ramdane-Cherif. 2004. “Competitive memetic algorithms for arc routing problems.” *Annals of Operations Research* 131 (1): 159–185.

Liu, T., Z. Jiang, F. Chen, R. Liu and S. Liu. 2008. Combined location-arc routing problems: A survey and suggestions for future researched.^eds. *2008 IEEE International Conference on Service Operations and Logistics, and Informatics(SOLI'2008)*, Beijing: IEEE, 2336–2341.

Lourenço, H., O. Martin, and T. Stützle. 2003. “Iterated local search.” In *Handbook of metaheuristics*, edited by F. Glover and G. Kochenberger, 320–53. Boston: Kluwer.

Mendoza, J.E., B. Castanier, C. Guéret, A.L. Medaglia, and N. Velasco. 2010. “A memetic algorithm for the multi-compartment vehicle routing problem with stochastic demands.” *Computers & Operations Research* 37 (11): 1886–1898.

Moscato, P. 1989. “On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms.” *Caltech Concurrent Computation Program, C3P. Report* 826: 1989.

Naderi, B., M. Zandieh, and S.M.T.F. Ghomi. 2009. “Scheduling job shop problems with sequence-dependent setup times.” *International Journal of Production Research* 47 (21): 5959–5976.

Nagata, Y., O. Bräysy, and W. Dullaert. 2010. “A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows.” *Computers & Operations Research* 37 (4): 724–737.

Pearn, W. 1989. “Approximate solutions for the capacitated arc routing problem.” *Computers & Operations Research* 16 (6): 589–600.

Pearn, W. 1991. “Augment-insert algorithms for the capacitated arc routing problem.” *Computers & Operations Research* 18 (2): 189–198.

Polacek, M., K.F. Doerner, R.F. Hartl, and V. Maniezzo. 2008. “A variable neighborhood search for the capacitated arc routing problem with intermediate facilities .” *Journal of Heuristics* 14 (5): 405–423.

Prins, C. 2004. “A simple and effective evolutionary algorithm for the vehicle routing problem.” *Computers & Operations Research* 31 (12): 1985–2002.

Santos, L., J. Coutinho-Rodrigues, and J. Current. 2010. “An improved ant colony optimization based algorithm for the capacitated arc routing problem.” *Transportation Research Part B: Methodological* 44 (2): 246–266.

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., P.M. França and A.L.M. França. 2012. “Grasp with evolutionary path-relinking for the capacitated arc routing problem.” *Computers & Operations Research*. doi: 10.1016/j.cor.2011.10.014.