Improved Memetic Algorithm for Capacitated Arc Routing Problem

Yi Mei, Ke Tang and Xin Yao

Abstract— **Capacitated Arc Routing Problem (CARP) has attracted much interest because of its wide applications in the real world. Recently, a memetic algorithm proposed by Lacomme et al. (LMA) has been demonstrated to be a competitive approach to CARP. The crossover operation of LMA is carried out based on an implicit representation scheme, while it conducts local search on the basis of an explicit representation scheme. Hence, the search process of LMA involves frequent switch between the spaces defined by the two representation schemes. However, a good solution in one space is not necessarily good in the other. In this paper, we show that the local search process of LMA might be ineffective due to such reason, and suggest adopting a more careful way to coordinate the local search. As a result, two new local search methods are proposed, which resulted in two improved LMA (ILMA) algorithms. Experimental results on benchmark instances of CARP showed that the ILMA significantly outperformed LMA in terms of solution quality, and sometimes even in terms of computational time. Furthermore, ILMA improved the best known solutions for 8 problem instances out of the total 24 instances.**

I. Introduction

The arc routing problem is a classic combinatorial optimization problem with wide applications in real world, such as urban waste collection, post delivery, salting route optimization, winter gritting, school bus scheduling, etc [1]. Within the various forms of arc routing problem, the Capacitated Arc Routing Problem (CARP) is the most typical form which has been studied by most researchers. Briefly speaking, CARP is to determine a least cost plan for vehicles to serve certain edges and arcs in a given graph under some side constraints. More specifically, the problem is defined on a graph. A number of vehicles with limited capacities are based at a special vertex of the graph, which is called the depot vertex. Some certain edges and arcs, which are called tasks, are required to be served by the vehicles. Each edge and arc could be traversed for any number of times. CARP is to make a schedule for the vehicles to serve all the tasks so that the total traversal cost is minimized and the following constraints are satisfied:

- Each vehicle starts and ends at the depot.
- Each task is served by exactly one vehicle.
- The total demand of the tasks served by each vehicle does not exceed its capacity.

The authors are with the Nature Inspired Computation and Applications Laboratory, the Department of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China. Xin Yao is also with CERCIA, the School of Computer Science, University of Birmingham, Edgbaston, Birmingham B15 2TT, U.K. (emails: meiyi@mail.ustc.edu.cn, ketang@ustc.edu.cn, x.yao@cs.bham.ac.uk).

Corresponding author: Ke Tang (Phone: +86–551–3600754).

CARP is NP-hard [2] and exact methods are only available for instances with small and medium size. For large instances, heuristics and meta-heuristics are promising methods for finding near-optimal solutions within the limited time budget. Up to now, many heuristics ([2], [3], [4], [5], [6]) and meta-heuristics ([7], [8], [9], [10], [11], [12], [13], [14], [15]) have been proposed for solving CARP.

Recently, Lacomme et al. proposed a memetic algorithm in [11] that has been demonstrated as a competitive approach to CARP. In Lacomme's memetic algorithm (called LMA for short), solutions are encoded implicitly as chromosomes. The evaluation of a chromosome is divided into two steps. Firstly, the chromosome is converted to an explicit solution by a decoding procedure. Then the explicit solution is evaluated and the fitness of the chromosome is defined as the total cost of the explicit solution. In this minimization problem, a chromosome with a lower fitness is thus considered to be better. For the sake of simplicity, "explicit solution" will be called "solution" for short hereafter. In LMA, the crossover operator is specifically designed for the implicit representation scheme, while the local search process is still conducted in the solution space instead of in the chromosome space. At each generation, the solution obtained from local search will be transferred back into the chromosome space. The switch between the implicit and explicit representation may improve the performance of the algorithm. Another example of switching between multiple representations, which is for numerical optimization problem, can be found in [16]. After the crossover and the local search, the parent and offspring populations will be combined together and the parent population for the next generation will be selected based on fitness of the chromosomes. However, the fitness of an offspring chromosome is not directly evaluated by the solution from which the chromosome was obtained. Instead, a split procedure is applied to the chromosome again to generate a feasible solution, and the fitness of the chromosome is the total cost of this feasible solution. Furthermore, the local search of LMA mainly focuses on reducing the total cost of the solution, while overlooks its feasibility. Hence, it is very likely that the local search procedure generates a low cost infeasible solution, while after being transferred back to the chromosome space, the corresponding chromosome has a very high fitness (because the feasibility is more focused in this stage). Such inconsistency between the quality of the resultant solution and the corresponding chromosome either causes a waste of computational resources or leads to the inefficiency of the local search process in the memetic algorithm. In this paper, we improve the local search in the

way of evaluating the candidate solutions during the process and selecting the resultant solution. Based on the fact that a chromosome converted from a *feasible* solution with a low total cost must have a low fitness, the improved local search focuses more on the *feasible* solutions by punishing the infeasible solutions and returns the best *feasible* solution so as to restrict the fitness of the resultant chromosome by the lowest upper bound. Therefore, the improved local search process is more likely to result in a chromosome with a low fitness.

The rest of the paper is organized as follows: The background of our work is introduced in section II, including the detailed introduction of CARP and LMA. In section III, the improved local search process is proposed. Afterwards, experimental studies of the improved local search is carried out on a well-known CARP benchmark in section IV. Finally, the conclusion and future work are described in section V.

II. BACKGROUND

A. Capacitated Arc Routing Problem

For the sake of convenience, we only consider the undirected form of CARP, which is defined on an undirected graph $G(V, E)$. Each edge $(v_i, v_j) \in E$ is associated with three nonnegative features, i.e., the traversal cost $c^{trav}(v_i, v_j)$, the serving cost $c^{serv}(v_i, v_j)$ and the demand $d(v_i, v_j)$. An edge with a positive demand is required to be served by vehicles at the cost of its serving cost. Such an edge is called a task, and the set E_R of all the tasks is called the task set, i.e., $E_R = \{(v_i, v_j) \in E | d(v_i, v_j) > 0\}$. Here we suppose that there are n tasks, i.e., $|E_R| = n$. Note that the serving cost is only induced by serving a task, we have $c^{serv}(v_i, v_j) > 0 \iff d(v_i, v_j) > 0$ and $c^{serv}(v_i, v_j) = 0 \iff d(v_i, v_j) = 0$. m vehicles with an identical capacity Q are based at the depot $v_s \in V$ to serve the tasks. For an edge task (v_i, v_j), service in either positive direction (denoted as the arc $\langle v_i, v_j \rangle$) or negative direction (denoted as the arc $\langle v_j, v_i \rangle$) is allowed. Then each direction of each edge task is assigned a unique positive integer task index. A task index $t \in \mathbb{N}^+$ is associated with six features: the tail vertex $tv(t)$, the head vertex $hv(t)$, the traversal cost $c^{trav}(t)$, the serving cost $c^{serv}(t)$, the demand $d(t)$, and the inverse task index $inv(t)$. For an edge task (v_i, v_j) and the two task indices t_1 assigned to $\langle v_i, v_j \rangle$ and t_2 assigned to $\langle v_j, v_i \rangle$, the features are defined as follows:

- $hv(t_1) = tv(t_2) = v_i$;
- $tv(t_1) = hv(t_2) = v_j$;
- $c^{trav}(t_1) = c^{trav}(t_2) = c^{trav}(v_i, v_j)$;
- $c^{serv}(t_1) = c^{serv}(t_2) = c^{serv}(v_i, v_j)$;
- $d(t_1) = d(t_2) = d(v_i, v_j)$;
- $inv(t_1) = t_2$, $inv(t_2) = t_1$.

Furthermore, a depot loop is assigned the index 0 and its features are defined as:

- $tv(0) = hv(0) = v_s$;
- $c^{trav}(0) = c^{serv}(0) = d(0) = 0$;
- $inv(0) = 0$.

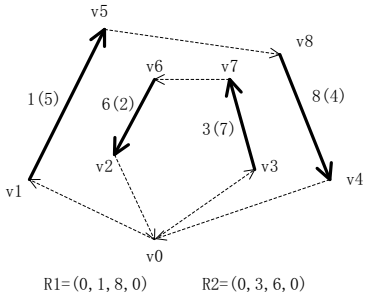

Fig. 1. An example of a CARP solution

After the definitions, a CARP solution could be explicitly represented as a set of routes $S = (R_1, R_2, ..., R_m)$, where each route R_k is a sequence of the task indices, i.e., $R_k = (t_1^k, t_2^k, ..., t_{l_k}^k)$, where $t_p^k (1 \leqslant p \leqslant l_k)$ are task indices previously defined. In order to ensure that each route starts and ends at the depot, the task sequence starts and ends at the depot loop 0, i.e., $t_1^k = t_{l_k}^k = 0$. An illustration is given in Fig. 1. In the graph, the task set $E_R = \{(v_1, v_5), (v_2, v_6), (v_3, v_7), (v_4, v_8)\}$ and the depot is v_0. The task indices $1, 2, 3, 4$ are assigned to $\langle v_1, v_5 \rangle$, $\langle v_2, v_6 \rangle$, $\langle v_3, v_7 \rangle$ and $\langle v_4, v_8 \rangle$, respectively, while $5, 6, 7, 8$ are assigned to their inversions. There are two numbers associated with each task, the one out of the bracket denotes the task index of the direction traversed by the route, while the one in the bracket denotes that of its inversion. The dashed arrows between adjacent task indices (e.g., $\langle v_0, v_1 \rangle$ and $\langle v_7, v_6 \rangle$ in Fig.1) stand for the shortest path from the earlier vertex to the latter vertex, which can be obtained by Dijkstra's algorithm [17].

For each route $R_k = (t_1^k, t_2^k, ..., t_{l_k}^k)$, its total cost $c^{tot}(R_k)$ and total demand $d(R_k)$ can be calculated as follows:

$$c^{tot}(R_k) = \sum_{p=1}^{l_k - 1} [c^{serv}(t_p^k) + dist(tv(t_p^k), hv(t_{p+1}^k))]$$

$$d(R_k) = \sum_{p=1}^{l_k} d(t_p^k)$$

where the function $dist(v_1, v_2)$ indicates the distance from vertex v_1 to vertex v_2, which is equal to the length of the shortest path from v_1 to v_2.

Under such representation scheme, the problem can be formulated as follows:

$$\min \quad c^{tot}(S) = \sum_{k=1}^{m} c^{tot}(R_k) \tag{1}$$

$$s.t. : \sum_{k=1}^{m} (l_k - 2) = n \tag{2}$$

$$t_{p_1}^{k_1} \neq t_{p_2}^{k_2}, \forall (k_1, p_1) \neq (k_2, p_2) \tag{3}$$

$$t_{p_1}^{k_1} \neq inv(t_{p_2}^{k_2}), \forall (k_1, p_1) \neq (k_2, p_2) \tag{4}$$

$$d(R_k) \leqslant Q, \forall 1 \leqslant k \leqslant m \tag{5}$$

Each pair (k, p) of two positive integers k and p must satisfy the conditions of $1 \leqslant k \leqslant m$ and $1 < p < l_k$. The inequality $(k_1, p_1) \neq (k_2, p_2)$ between the two pairs (k_1, p_1) and (k_2, p_2) indicates that either $k_1 \neq k_2$ or $p_1 \neq p_2$.

The objective function (1) is the total cost of all the routes. Constraints (2)-(4) guarantee that each task is served exactly once by one vehicle. Constraint (5), which is also called the capacity constraint, indicates that the total demand served by each vehicle does not exceed it capacity.

B. Lacomme's Memetic Algorithm

In LMA, a solution is encoded as a chromosome which is a sequence of n positive task indices, i.e., $C = (t_1, t_2, ..., t_n)$ where $t_i > 0, \forall 1 \leqslant i \leqslant n$. Each task index t_i corresponds to a unique edge task, which means $t_i \neq t_j$ and $t_i \neq inv(t_j)$, $\forall i \neq j$. While being evaluated, a chromosome is firstly converted to a solution by the *split* procedure which was proposed by Ulusoy in [4]. The *split* procedure cuts the given chromosome $C = (t_1, t_2, ..., t_n)$ at some intermediate positions to generate a set of feasible routes so that the additional cost caused by the cut is minimized. The resultant solution $S = (R_1, R_2, ...)$ is thus the optimal feasible solution with respect to the task sequence $(t_1, t_2, ..., t_n)$. The fitness of the chromosome C, denoted as $f(C)$, is then defined as equal to the total cost of S, i.e., $f(C) = c^{tot}(S)$. An illustration of the *split* procedure is given in Fig.2. Besides the task indices associated with each task being introduced in Fig.1, the distance between each pair of vertices and the demand of each task are given in the brackets $d()$ and $c()$, respectively. For each task index, the serving cost is equal to the distance between its end-vertices, and thus is also given in the bracket $c()$ associated with it. For example, the demand and the serving cost of the task (v_1, v_2) are 4 and 5, respectively, while the distance from v_0 to v_1 is 20. Assume that the capacity *Q* is 9. Then through the *split* procedure, the chromosome C is decoded to the optimal feasible solution $S = (R_1, R_2, R_3)$, which induces minimal additional cutting cost. The total demands of R_1, R_2 and R_3 are 7, 5 and 7, respectively, and the total cost of S is 141. Therefore, the fitness of the chromosome $C = (1, 2, 3, 4, 5)$ is $f(C) = 141$.

The process of LMA can be summarized as follows:

Step 1: Generate an initial population with nc distinct chromosomes, which includes three good chromosomes those are generated by augment-merge [2], path scanning [3], and Ulusoy's heuristic [4], respectively, for speeding up the convergence of the algorithm. Then sort the population in the increasing order of the fitness of the chromosomes.

Step 2: Select two chromosomes P_1 and P_2 from the current population by the binary tournament selection.

Step 3: Apply the ordered crossover (OX) operator to P_1 and P_2 to generate a chromosome O_x and set the offspring $O = O_x$. OX was firstly proposed for traveling salesman problem and can be referred in [18].

Step 4: Apply the local search process to O_x with a certain probability. Then replace O with the resultant chromosome O_m if $f(O_m)$ is not used in the current population. The local search process is complicated and thus will be described in detail independently.

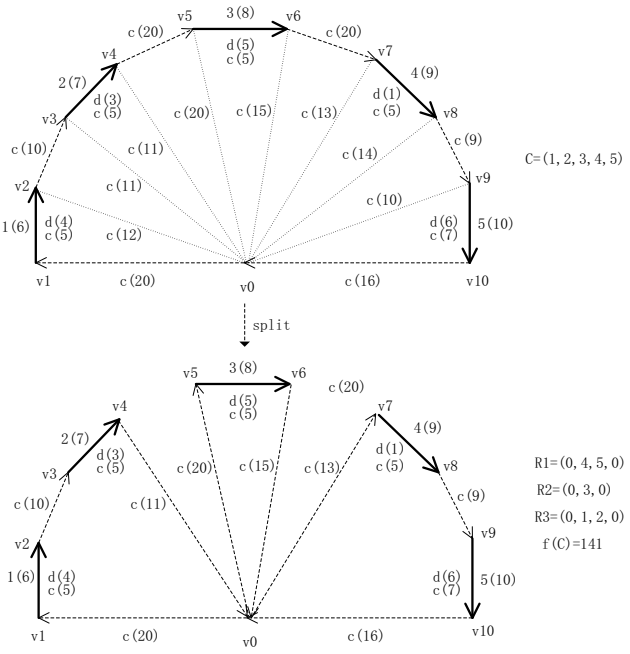

Fig. 2. The *split* procedure

Step 5: Evaluate the final offspring O. Three cases may occur: (1) $f(O)$ is equal to $f(P_1)$ or $f(P_2)$; (2) $f(O)$ is equal to a chromosome other than P_1 and P_2 in the current population; (3) $f(O)$ is not used in the current population. In case (1), replace the corresponding parent by O; In case (2), discard O and maintain the current population; In case (3), randomly choose a chromosome whose position in the current population is between $\lfloor nc/2 \rfloor$ and nc, and replace it with O. Then resort the modified population so that the fitness of the chromosomes follows an increasing order.

Step 6: If the replacement criteria are met, replace $nrep$ chromosomes with other randomly generated distinct chromosomes and go to step 2;

Step 7: If the termination criteria are met, stop the algorithm; Otherwise, go to step 2.

The local search process at step 4 can be divided into the following three phases:

1) Convert the starting chromosome C_0 to a solution S_0;
2) Conduct the local search $S^* = LS_{sol}(S_0)$ in the solution space;
3) Convert the resultant solution S^* back to a chromosome C_a^*;

During phase 1), the chromosome C_0 is converted to the solution S_0 by applying the *split* procedure to C_0 (Detailed description of the *split* procedure can be found in the first paragraph of sub-section II-B). While during phase 3), the resultant solution S^* is converted to the chromosome C_a^*

by simply concatenating its routes together. For the sake of convenience, we denote the converting operation from a chromosome C to a solution S and the reconverting operation from a solution S to a chromosome C as $S = cvt(C)$ and $C = recvt(S)$, respectively.

Given a starting chromosome C_0 to be mutated by the local search, it is firstly converted to a solution $S_0 = cvt(C_0)$. Then the local search $S^* = LS_{sol}(S_0)$ is conducted in the solution space with the starting solution S_0. Initially, the current solution S_c is set to S_0. At each step, the local search examines the candidate solutions generated by applying the inversion, single insertion, double insertion, swap and 2-opt operators to S_c in order. Once a solution S' causing a cost reduction is obtained, S_c is immediately replaced by S' and a new step starts. The local search stops after a step without finding a cost reduction and returns the final current solution $S^* = S_c$. Finally, the resultant chromosome $C_a^* = recvt(S^*)$ is obtained.

During the local search that is conducted in the solution space, solutions are evaluated only with respect to the total cost and S^* will be the solution with the lowest total cost among all the solutions that have been explored throughout the local search process. However, since the violation of the capacity constraint is not considered, S^* may violate the capacity constraint much and a large additional cutting cost may be induced when applying the *split* procedure to the chromosome converted from it, i.e., C_a^*. Hence, the resultant solution S^*, which is the best solution with respect to the total cost, may be converted to a chromosome C_a^* with a high fitness after the addition of the cutting cost. In other words, the superiority of S^* is not consistent with the superiority of C_a^*. Therefore, the local search is not able to guarantee the quality of the resultant chromosome C_a^*. Actually, the resultant chromosome is usually even worse than the starting chromosome, especially when the fitness of the starting chromosome is low. To summarize, the inconsistency between the quality of the resultant solution and the converted chromosome will make the local search process inefficient.

III. Improving the Local Search

It is natural to think of conducting the local search process directly in the chromosome space. However, due to the high time complexity of the evaluation of a chromosome, this will cause a large amount of additional computational time that one cannot afford. Hence, it is expected that the local search is still conducted in the solution space but its performance is as close to that conducted in the chromosome space as possible, which means the resultant chromosome that is converted from the resultant solution of the local search is as close to a local optimum in the chromosome space as possible.

In an ideal situation, the local search should result in a solution S_{id}^* so that for the converted chromosome $C_{id}^* = recvt(S_{id}^*)$, we have $f(C_{id}^*) \leqslant f(C)$, $\forall C \in \mathcal{N}(C_{id}^*)$, where $\mathcal{N}(C_{id}^*)$ is the neighborhood of C_{id}^* in the chromosome space. In practice, it attempts to return such a solution that the fitness of the converted chromosome is as low as possible.

Considering the relationship between the quality of solutions and chromosomes, we have the following two properties:

prop 1) $c^{tot}(cvt(C)) = f(C)$ for each chromosome C;
prop 2) $f(recvt(S)) \leqslant c^{tot}(S)$ for each *feasible* solution S.

These can be inferred directly from the definition of the fitness of the chromosome. From prop 2), we can find that the *feasible* solutions can be converted to chromosomes whose fitness is no worse than its total cost. On the other hand, for infeasible solution, additional cutting cost for the infeasible routes will be caused when evaluating the chromosomes converted from it. Therefore, the fitness of the chromosome converted from an infeasible solution is not always less than the total cost of that infeasible solution. In fact, the fitness of the converted chromosome may be much greater than the total cost of the infeasible solution if the cutting cost is very large. Therefore, the superiority of a *feasible* solution can lead to the superiority of the converted chromosome, while that of an infeasible solution cannot.

Based upon the above considerations, the local search of LMA is modified in the following two aspects:

1) The violations of capacity constraints are taken into consideration during the local search. Concretely, for two solutions S_1 and S_2, it is considered that S_1 is better than S_2 if $g(S_1) < g(S_2)$, instead of the condition $c^{tot}(S_1) < c^{tot}(S_2)$ used in LMA. $g(S)$ is a compound fitness function of the total cost and the constraint violation that is defined as follow:
$$g(S) = c^{tot}(S) + \lambda \times tvl(S)$$
where $tvl(S) = \sum_k (\max[d(R_k(S)) - Q, 0])$ is the total violated load of S and $\lambda > 0$ is the penalty coefficient. It can be considered that the original local search employs the function $g(S)$ as well, but with the coefficient $\lambda = 0$.
2) The resultant solution S^* is set as the best *feasible* solution found during the local search instead of the final solution that with the lowest value of the fitness function g. Initially, S^* is set to the starting *feasible* solution S_0. During the local search process, if a new *feasible* solution S_f is obtained and $g(S_f) < g(S^*)$, then replace S^* with S_f.

By applying the first modification, the infeasible solutions are punished according to their constraint violations and the local search focuses more on the *feasible* solutions. Then better *feasible* solutions are more hopeful to be achieved. After the second modification, the best *feasible* solution is returned to be converted to the resultant chromosome at the end of the modified local search process. Note that $f(recvt(S)) \leqslant c^{tot}(S)$ for each *feasible* solution S, letting the resultant solution S^* be the best *feasible* solution will give the fitness of the converted chromosome C_a^* the lowest upper bound. At least, we can ensure that the resultant chromosome C_a^* is never worse than the starting chromosome C_0. It is induced from the following three properties:

$$c^{tot}(S_0) = f(C_0) \tag{6}$$

$$f(C_a^*) \leqslant c^{tot}(S^*) \tag{7}$$

$$c^{tot}(S^*) \leqslant c^{tot}(S_0) \tag{8}$$

Eq. (6), (7) are obtained from prop 1) and 2), while Eq. (8) is obvious since S_0 is a *feasible* solution. Then we have

$$f(C_a^*) \leqslant c^{tot}(S^*) \leqslant c^{tot}(S_0) = f(C_0)$$

IV. Experimental Studies

A. Experimental Setup

An improved LMA (ILMA) is generated by replacing the original local search process of LMA with the modified local search process, while maintaining all the other ingredients. To describe concretely, ILMA maintains a population of size 30 during the search process. At first, the probability of applying local search is set to 0.1. The first restart happens after 20000 productive crossovers or 6000 productive crossovers without finding new best solution and 8 solutions in the current population are replaced by randomly generated solutions. Here a productive crossover stands for a crossover generating an offspring distinct from all the solutions in the current population. Then the local search rate is modified to 0.2, and the replacement criterion is changed to 2000 productive crossovers, which means a restart is called after 2000 productive crossovers behind the previous restart. The whole algorithm terminates when the number of restarts exceeds 20.

Two versions of ILMA are generated by using two different settings of λ, denoted as ILMA1 and ILMA2, respectively. They are described as follows:

- ILMA1: In this version, we set $\lambda = \infty$ throughout the local search process. In this case, only feasible candidate solutions are considered during the local search process. Once an infeasible solution is found, it is immediately discarded.
- ILMA2: In this version, we set λ as an adaptive parameter. Initially, λ is set to 1. Subsequently, it is halved after 5 consecutive steps generating feasible solutions, while is doubled after 5 consecutive steps generating infeasible solutions. The reason is that when feasible solutions are consecutively accepted, it is possible that most of the solutions around the current solution are feasible, and halving λ can help the search focus more on the total cost. On the other hand, the search might have difficulty in finding feasible solutions when infeasible solutions are chosen sequentially. Hence, doubling λ can make the search focus more on feasibility and increase the probability of obtaining feasible solutions.

ILMA1 and ILMA2 are tested on the egl benchmark instances, which is a well-known CARP benchmark. It was generated by Eglese based on data from a winter gritting application in Lancashire [19], [20], [21], and consists of 24 instances based on two graphs, each with distinct set of required edges and capacity constraints. Note that there are two other commonly used benchmark instance sets in the literature of CARP, namely the gdb set [22] and the val set [23]. However, all the gdb instances and most of the val instances have been solved optimally so far, while most egl instances are still difficult to tackle because of their large problem sizes. Therefore, we select the egl set as our testing set so that the difference among algorithms can be displayed more clearly. In our experiments, only LMA is picked for reference. Since most of the previously proposed meta-heuristic approaches have been dominated by LMA on the egl set, comparison to them is unnecessary and thus is omitted due to the space limitation. As a competitive algorithm for CARP, LMA is only outperformed by a tabu search algorithm proposed by Brandão et al. in [13], which provided best known solutions for all the egl instances. Hence, comparing with Brandão's tabu search is equivalent to comparing with the best known solutions.

B. Results

30 runs were conducted for both ILMA1 and ILMA2 on the egl instances. Firstly, the best results are presented in Table I. The columns headed "$|V|$", "$|E_R|$" and "$|E|$" stand for the number of vertices, tasks and total edges of the instance, respectively. In this benchmark, the number of tasks varies from 51 to 190. The column headed "LB" gives the lower bounds for the instances which are collected from [23] [24] [25] [26], and the column headed "Best Known" provides the total cost of the best known solutions, which were obtained in [13]. The columns headed "B.Cost" and "CPU(s)" for "LMA" indicate the total cost of the best solutions obtained by LMA and the runtime needed for LMA, which were directly obtained from the original literature [11]. For "ILMA1" and "ILMA2", the column headed "B.Cost" indicates the total cost of the best solution obtained in all the 30 runs by the corresponding algorithm, and the column headed "CPU(s)" indicates the average runtime. The runtime of all the compared algorithms are shown in seconds. The last three rows headed "mean", "No.opt" and "APD" are the mean value, number of optimal results and average percentage deviation from the lower bound, respectively. For "B.Cost", the optimal values are with stars, and the best results among the compared algorithms are in bold. The updated best known solutions are marked with underline.

In our experiments, ILMA1 and ILMA2 were coded in C and run using an Intel(R) Xeon(R) E5335 2.00GHz. Since LMA was implemented on a Pentium III 1GHz, the running times given in [11] were divided by 2 in order to make a fair comparison.

Firstly, we keep our eyes on the performance of ILMA1. It is observed that it was much faster than the other two algorithms. The mean runtime of ILMA1 was about 1/3 as that of LMA, and even less than 1/3 as that of ILMA2. At the same time, the performance of ILMA1 on solution quality was a little better than that of LMA. For the 5 instances that LMA reached the optimal solutions (E1-A, E1-B, E2-A, E3-A and S1-A), ILMA1 also reached the optimal solutions. For

TABLE I

BEST RESULTS ON THE EGL BENCHMARK. "B.COST" AND "CPU(S)" STAND FOR THE TOTAL COST OF THE BEST SOLUTION AND THE RUNTIME, RESPECTIVELY. "MEAN", "NO.OPT" AND "APD" REPRESENT THE MEAN VALUE, NUMBER OF OPTIMAL RESULTS AND AVERAGE PERCENTAGE DEVIATION FROM THE LOWER BOUND, RESPECTIVELY. * MEANS THE LOWER BOUND WAS REACHED.

Name	$\|V\|$	$\|E_R\|$	$\|E\|$	LB	Best	LMA		ILMA1		ILMA2	
					Known	B.Cost	CPU(s)	B.Cost	CPU(s)	B.Cost	CPU(s)
E1-A	77	51	98	3548	3548*	**3548***	37.1	**3548***	0.0	**3548***	0.2
E1-B	77	51	98	4498	4498*	**4498***	34.7	**4498***	14.3	**4498***	34.5
E1-C	77	51	98	5566	5595	**5595**	35.6	**5595**	15.1	**5595**	51.7
E2-A	77	72	98	5018	5018*	**5018***	76.3	**5018***	6.5	**5018***	8.5
E2-B	77	72	98	6305	6317	6340	76.7	**6317**	32.6	**6317**	92.8
E2-C	77	72	98	8243	8335	8415	64.8	**8335**	29.7	**8335**	109.6
E3-A	77	87	98	5898	5898*	**5898***	121.0	**5898***	40.3	**5898***	59.7
E3-B	77	87	98	7704	7777	7822	127.7	7778	49.2	**7777**	144.5
E3-C	77	87	98	10163	10305	10433	103.2	10305	43.6	**10292**	160.7
E4-A	77	98	98	6408	6456	6461	145.9	6461	68.5	**6456**	157.1
E4-B	77	98	98	8884	9000	9021	156.4	9025	54.5	**8991**	205.5
E4-C	77	98	98	11427	11601	11779	126.2	11668	51.5	**11587**	229.4
S1-A	140	75	190	5018	5018*	**5018***	104.3	**5018***	31.5	**5018***	22.4
S1-B	140	75	190	6384	6388	6435	104.4	**6388**	40.9	**6388**	116.9
S1-C	140	75	190	8493	8518	**8518**	82.8	**8518**	35.4	**8518**	130.3
S2-A	140	147	190	9824	9956	9995	437.2	9966	174.6	**9911**	445.4
S2-B	140	147	190	12968	13165	13174	380.3	13223	144.1	**13150**	490.6
S2-C	140	147	190	16353	16505	16795	373.5	16575	123.6	**16489**	554.8
S3-A	140	159	190	10143	10260	10296	535.3	10295	209.4	**10261**	513.6
S3-B	140	159	190	13616	13807	14053	532.0	13823	164.4	**13813**	580.2
S3-C	140	159	190	17100	17234	17297	437.2	17341	147.3	**17288**	664.0
S4-A	140	190	190	12143	12341	12442	768.8	12373	263.2	**12348**	760.0
S4-B	140	190	190	16093	16442	16531	715.1	**16440**	220.2	**16345**	867.1
S4-C	140	190	190	20375	20591	20832	747.5	20772	200.6	**20556**	986.4
mean				9673.8	9773.9	9842.3	263.5	9799.1	90.0	9766.5	307.7
No.opt				-	5	5	-	5	-	5	-
APD				-	0.84	1.38	-	1.01	-	0.78	-

the rest of instances, ILMA1 was able to achieve solutions better than those obtained by LMA on 14 instances, while was defeated by LMA on only 3 instances. The mean value and APD of ILMA1 on the whole benchmark were also better than those of LMA. Besides, ILMA1 found a solution better than the best known solution on S4-B.

Next, ILMA2 is focused. It can be seen that ILMA2 was the slowest algorithm among the three compared algorithms. Its mean runtime was about 3.4 times as that of ILMA1, while about 1.1 times as that of LMA. However, ILMA2 provided solutions with much better qualities than the other two algorithms. The mean total cost of the best solutions reached by ILMA2 was even less than that of the best known solutions, not to mention ILMA1 and LMA. Furthermore, ILMA2 updated the best known solutions for 8 out of the 24 instances, while reached the best known solutions on 12 instances.

Furthermore, we find that ILMA1 and ILMA2 both obtained solutions as good as the best known solutions on all the instances with medium solution space (instances in E1, E2 and S1 set containing less than 80 tasks), while ILMA1 was outperformed by ILMA2 on most of the large instances (the rest of instances that have more than 80 tasks). For the exceptional instance E3-A on which ILMA1 achieved the global optimum as well, the solution space was the smallest among all the large instances since it required the least number of vehicles. The mean total cost and APD of ILMA1 and ILMA2 on the medium instances were both 5915.0 and 0.24. For the large instances, the mean total cost and APD of ILMA1 were 12129.5 and 1.47, while those of ILMA2 were 12077.5 and 1.10.

Table II presents the average results of ILMA1 and ILMA2. Since the average results of LMA is not available in the original literature [11], we just list the best results of LMA for reference. The average total cost and the standard deviation are given. Besides, we count the number of runs in which ILMA1 and ILMA2 provided solutions no worse than the best solutions obtained by LMA, which is denoted

TABLE II

AVERAGE RESULTS ON THE EGL BENCHMARK. "AVE.COST" AND "STD.DEV" STAND FOR THE AVERAGE TOTAL COST AND THE STANDARD DEVIATION, RESPECTIVELY. "$|nw|$" INDICATES THE NUMBER OF RESULTS THAT ARE NO WORSE THAN "B.COST" OF LMA.

Name	LMA	ILMA1			ILMA2		
	B.Cost	Ave.Cost	Std.Dev	$\|nw\|$	Ave.Cost	Std.Dev	$\|nw\|$
E1-A	**3548**	**3548.0**	0.0	30	**3548.0**	0.0	30
E1-B	**4498**	4533.5	7.9	1	4512.3	12.3	11
E1-C	**5595**	5625.2	20.0	4	5602.1	9.9	18
E2-A	**5018**	5018.3	1.6	29	**5018.0**	0.0	30
E2-B	6340	6351.8	19.2	5	**6338.2**	10.7	12
E2-C	8415	8391.7	48.2	23	**8356.2**	40.0	29
E3-A	**5898**	5924.3	33.0	15	5918.8	33.9	21
E3-B	7822	7844.5	43.4	9	**7797.8**	25.0	25
E3-C	10433	10398.4	57.3	21	**10314.2**	23.7	30
E4-A	**6461**	6476.1	15.9	1	6472.4	16.0	5
E4-B	**9021**	9077.4	40.7	0	9052.6	40.5	7
E4-C	11779	11806.0	51.6	5	**11665.9**	49.0	29
S1-A	**5018**	5052.7	27.0	11	5028.5	29.1	26
S1-B	6435	6478.8	51.0	8	**6404.9**	28.1	26
S1-C	**8518**	8591.1	60.8	7	8567.0	37.8	5
S2-A	**9995**	10074.7	51.7	2	10025.9	57.0	10
S2-B	**13174**	13346.8	53.9	0	13292.3	70.4	1
S2-C	16795	16741.1	78.9	24	**16621.3**	69.7	30
S3-A	**10296**	10350.6	42.0	1	10361.5	50.4	3
S3-B	14053	13989.4	57.2	28	**13948.3**	64.4	30
S3-C	**17297**	17504.1	112.8	0	17377.3	79.2	3
S4-A	**12442**	12544.2	65.2	1	12484.5	67.2	8
S4-B	**16531**	16605.3	86.1	6	16553.1	71.9	11
S4-C	20832	20869.1	59.0	7	**20776.9**	112.9	19
mean	9842.3	9881.0	45.2	9.9	9834.9	41.6	17.5

as $|nw|$. The lowest values of total cost among the compared algorithms are in bold. From Table II, it is observed that the average results of ILMA2 were significantly better than those of ILMA1. The average total cost of the solutions of ILMA2 were better than those of ILMA1 on 23 instances out of the total 24 instances and the mean value of the average total cost of ILMA2 was much less than that of ILMA1. The average results of ILMA2 were even comparable with the best results of LMA, since their mean value were 9834.9 and 9842.3, respectively. On the other hand, the average results of ILMA1 with the mean value 9881.0, was much worse than the best results of LMA. In addition, ILMA1 achieved solutions no worse than the best results of LMA in averagely 9.9 runs out of the total 30 runs, while ILMA2 did it in averagely 17.5 runs.

Overall, ILMA1 had a significant advantage on speed. In terms of solution quality, it performed a little better than LMA in the best case despite its relatively poor average performance, and could reproduce the best known solutions on the instances with not large solution spaces. ILMA2, on the other hand, was much slower than ILMA1, with its runtime a little more than that of LMA. However, it had a much better performance with respect to solution quality. It could achieve solutions even better than the best known solutions in the best case, and its average performance was comparable with the best performance of LMA.

It should be noted that the computation time as collected in this paper depends on many factors, such as the CPU clock speed, operating systems, compilers, coding skills of the programmer, etc. So the comparison with LMA's computation time is meant to be indicative. We use CPU time here because we do not have access to other computation time metrics for LMA. In spite of various factors that might influence the computation time, we believe ILMA1 is indeed much faster than the original LMA since the difference between ILMA1 and LMA is quite significant even if we have taken the CPU clock speed into account.

C. Discussion

It is observed that both ILMA1 and ILMA2 could achieve solutions better than LMA. This was due to the fact that after the modifications of the local search, the chromosome converted from the resultant solution was more likely to have a low fitness, and thus the local search process became more efficient for obtaining chromosomes as close to a local optimum in the chromosome space as possible.

In ILMA1, only feasible solutions were considered during the local search process. Then the search process only explored within the feasible region consisting of the starting solution and the resultant solution was no better than the local optimum within the current feasible region. Therefore, ILMA1 was quite fast, but was not able to reach solutions out of the current feasible region. The average performance of ILMA1 was naturally much worse than the best performance of LMA with respect to the quality of solution. However, thanks to the diversity brought by the crossover operator, it could still reach quite good solutions in the best case, especially on the instances with not large solution spaces. In ILMA2, on the other hand, each solution in the neighborhood of the current solution was considered, no matter it was feasible or not. Infeasible solutions were punished according to their constraint violations. Then the local search could jump out of the current feasible region to another through the bridges of infeasible solutions and better feasible solutions may be achieved. Hence, ILMA2 spent much more computational time for exploring a wider range of the solution space and thus was more capable of finding better solutions.

V. CONCLUSION AND FUTURE WORK

In this paper, we investigated Lacomme's Memetic Algorithm (LMA) which has been demonstrated as a competitive meta-heuristic approach for CARP. Unlike other meta-heuristics, LMA adopts an implicit representation scheme for solutions. That is, solutions are encoded as chromosomes and evaluated by a *split* procedure. ILMA conducts crossover in the chromosome space, while local search is conducted in the solution space. To be specific, the chromosome is firstly converted to a solution. Then local search is conducted in the solution space. Finally, the resultant solution is converted back to a chromosome again. A major defect of the local search is that the superiority of a solution may not lead to the superiority of the converted chromosome, and thus the quality of the resultant chromosome is not guaranteed. In order to improve the local search process so that the

resultant chromosome is as close to a local optimum in the chromosome space as possible, some modifications are made on it. The infeasible solutions encountered during the local search process are evaluated more appropriately by adding a penalty corresponding to the amount of their constraint violations. Besides, the modified local search returns the best *feasible* solution instead of the best solution with respect to the fitness function g used during the process. The modified local search never generate a worse chromosome. Furthermore, since the fitness of a chromosome converted from a *feasible* solution is bounded by the total cost of that solution, letting the resultant solution as the best *feasible* solution will restrict the fitness of the resultant chromosome with the lowest upper bound. The efficacy of the modified local search process is experimentally verified on a CARP benchmark set. The improved LMA (ILMA), generated by replacing the original local search with the modified local search, performed significantly better than the original LMA with respect to solution quality, and sometimes even with respect to speed. Furthermore, it improved best known solutions for 8 problem instances out of the total 24 instances.

In this paper, the modified local search focuses more on the feasible solutions. It selects the best feasible solution as the resultant solution of local search. The reason is that infeasible solution is quite difficult to be evaluated precisely and the qualities of an infeasible solution and the converted chromosome are not necessarily positively correlated. In fact, there still exists infeasible solutions whose total cost is quite low or the additional cutting cost is negligible so that the converted chromosomes are better than those converted from feasible solutions. In the future, we will try to evaluate the infeasible solutions during the local search more appropriately and consider allowing the local search return infeasible solutions. Moreover, the possibility of extending the modified local search to other local search-based approaches with implicit representation scheme will be investigated.

ACKNOWLEDGEMENT

This work is partially supported by the Fund for Foreign Scholars in University Research and Teaching Programs (Grant No. B07033), the Fund for International Joint Research Program of Anhui Science & Technology Department (No. 08080703016) and an EPSRC grant (EP/E058884/1) on "Evolutionary Algorithms for Dynamic Optimisation Problems: Design, Analysis and Applications."

REFERENCES

[1] M. Dror, *Arc routing. Theory, solutions and applications*. Boston: Kluwer Academic Publishers, 2000.

[2] B. Golden and R. Wong, "Capacitated arc routing problems," *Networks*, vol. 11, no. 3, pp. 305–315, 1981.

[3] B. Golden, J. DeArmon, and E. Baker, "Computational experiments with algorithms for a class of routing problems," *Computers and Operations Research*, vol. 10, no. 1, pp. 47–59, 1983.

[4] G. Ulusoy, "The fleet size and mix problem for capacitated arc routing," *European Journal of Operational Research*, vol. 22, no. 3, pp. 329–337, 1985.

[5] W. Pearn, "Approximate solutions for the capacitated arc routing problem," *Computers and Operations Research*, vol. 16, no. 6, pp. 589–600, 1989.

[6] ——, "Augment-insert algorithms for the capacitated arc routing problem," *Computers and Operations Research*, vol. 18, no. 2, pp. 189–198, 1991.

[7] A. Hertz, G. Laporte, and M. Mittaz, "A tabu search heuristic for the capacitated arc routing problem," *Operations Research*, vol. 48, no. 1, pp. 129–135, 2000.

[8] A. Hertz and M. Mittaz, "A variable neighborhood descent algorithm for the undirected capacitated arc routing problem," *Transportation Science*, vol. 35, no. 4, pp. 425–434, 2001.

[9] P. Beullens, L. Muyldermans, D. Cattrysse, and D. V. Oudheusden, "A guided local search heuristic for the capacitated arc routing problem," *European Journal of Operational Research*, vol. 147, no. 3, pp. 629–643, 2003.

[10] P. Greistorfer, "A Tabu Scatter Search Metaheuristic for the Arc Routing Problem," *Computers and Industrial Engineering*, vol. 44, no. 2, pp. 249–266, 2003.

[11] P. Lacomme, C. Prins, and W. Ramdane-Chérif, "Competitive memetic algorithms for arc routing problem," *Annals of Operational Research*, vol. 131, no. 1-4, pp. 159–185, 2004.

[12] H. Handa, D. Lin, L. Chapman, and X. Yao, "Robust solution of salting route optimisation using evolutionary algorithms," *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pp. 3098–3105, 2006.

[13] J. Brandao and R. Eglese, "A deterministic tabu search algorithm for the capacitated arc routing problem," *Computers and Operations Research*, vol. 35, no. 4, pp. 1112–1126, 2008.

[14] Y. Mei, K. Tang, and X. Yao, "A Global Repair Operator for Capacitated Arc Routing Problem," *IEEE Transactions on System, Man and Cybernetics, Part B*, accepted in October 2008.

[15] K. Tang, Y. Mei, and X. Yao, "Memetic Algorithm with Extended Neighborhood Search for Capacitated Arc Routing Problems," *IEEE Transactions on Evolutionary Computation*, conditionally accepted in January 2009.

[16] T. Schnier and X. Yao, "Using Multiple Representations in Evolutionary Algorithms," *Proceedings of the 2000 Congress on Evolutionary Computation (CEC'00)*, pp. 479–486, 2000.

[17] E. W. Dijkstra, "A Note on Two Problems in Connection with Graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[18] I. M. Oliver, D. J. Smith, and J. R. C. Holland, "A Study of Permutation Crossover Operators on the Traveling Salesman Problem," *Proceedings of the 2nd International Conference on Genetic Algorithms*, pp. 224–230, 1987.

[19] R. W. Eglese, "Routing winter gritting vehicles," *Discrete Applied Mathematics*, vol. 48, no. 3, pp. 231–244, 1994.

[20] R. W. Eglese and L. Y. O. Li, "A tabu search based heuristic for arc routing with a capacity constraint and time deadline," in *Metaheuristics: theory and applications*, I. H. Osman and J. P. Kelly, Eds. Boston: Kluwer Academic Publishers, 1996, pp. 633–650.

[21] L. Y. O. Li and R. W. Eglese, "An interactive algorithm for vehicle routing for winter-gritting," *Journal of the Operational Research Society*, vol. 47, no. 2, pp. 217–228, 1996.

[22] J. S. DeArmon, "A comparison of heuristics for the capacitated Chinese postman problem," Master's thesis, University of Maryland, College Park, MD, 1981.

[23] E. Benavent, V. Campos, A. Corberän, and E. Mota, "The capacitated arc routing problem: lower bounds," *Networks*, vol. 22, no. 7, pp. 669–690, 1992.

[24] D. Ahr, "Contributions to multiple postmen problems," Ph.D. dissertation, Rupercht-Karls-Universitat, Heidelberg, Germany, 2004.

[25] R. Baldacci and V. Maniezzo, "Exact methods based on node-routing formulations for undirected arc-routing problems," *Networks*, vol. 47, no. 1, pp. 52–60, 2006.

[26] H. Longo, D. A. M. Poggi, and E. Uchoa, "Solving capacitated arc routing problems using a transformation to the CVRP." *Computers and Operations Research*, vol. 33, no. 6, pp. 1823–1837, 2006.