Decomposing Large-Scale Capacitated Arc Routing Problems using a Random Route Grouping Method

Yi Mei
School of Computer Science and Information Technology
RMIT University
Melbourne, Victoria 3000, Australia
Email: yi.mei@rmit.edu.au

Xiaodong Li
School of Computer Science and Information Technology
RMIT University
Melbourne, Victoria 3000, Australia
Email: xiaodong.li@rmit.edu.au

Xin Yao
School of Computer Science
University of Birmingham
B15 2TT Birmingham, UK
Email: x.yao@cs.bham.ac.uk

***Abstract*—In this paper, a simple but effective Random Route Grouping (RRG) scheme is developed to decompose the Large-Scale Capacitated Arc Routing Problem (LSCARP). A theoretical analysis is given to show that the decomposition is guaranteed to be improved by RRG along with the improvement of the best-so-far solution during the search process. Then, RRG is combined with a cooperative co-evolution model to solve LSCARP. The experimental results on the EGL-G LSCARP set showed that given the same computational budget, the proposed approach obtained much better results than its counterpart without using decomposition.**

I. Introduction

As a practically significant and challenging problem, the Capacitated Arc Routing Problem (CARP) has been intensively investigated for decades [1]. Briefly speaking, given a graph representing a road network, CARP aims to schedule the routes of the vehicles to serve a set of edges and arcs (i.e., directed edges) in the graph subject to certain side constraints.

CARP has many applications in the logistics field, such as waste collection [2], winter gritting [3] and snow removal [4]. In spite of extenstive studies on CARP in the past, there is still a gap between the academic research and real-world situation. In particular, the problem size considered by previous research (e.g., less than 200 *required edges*) is much smaller than the practical instances (usually have more than 1000 required edges). Since CARP is NP-hard [5], which means that the solution space increases exponentially as the increase of the problem size, the algorithms that perform well for small and medium sizes may no longer retain its good performance for the large scale ones. In fact, preliminary studies [6] [7] [8] have shown that when the problem size increases from less than 200 to about 350, the performance of the algorithms deteriorate rapidly, either in solution quality [6] [7] or in convergence speed [8]. This paper studies CARPs with more than 300 required edges, which we refer to as *Large Scale CARP* (LSCARP), since the previous studies show that the size of 300 raises a scalability issue.

Although solving LSCARP is important, it has been overlooked so far. Most of the previous algorithms (e.g., [9] [10] [11] [12] [13] [14] [15]) were only evaluated on the commonly used *gdb* [16], *val* [17], *egl* [18] and Beullens's test sets [9], whose problem sizes are less than 200. The only existing LSCARP test set is the EGL-G set, which was generated by Brandão and Eglese in [6]. However, there are only limited studies about this LSCARP test set [6] [7] [8].

When solving LSCARP, an intuitive strategy is the divide-and-conquer strategy that decomposes the original huge problem into a number of smaller-sized sub-problems that can be solved individually. In the context of evolutionary computation, the Cooperative Co-evolution (CC) framework is a natural way to implement the divide-and-conquer strategy. In fact, the CC framework has been successfully applied to large scale continuous function optimization [19] [20] [21]. This paper studies how to apply CC to solve LSCARP.

In the CC framework, the decision vector $\mathbf{x}$ is decomposed into a number of non-overlapping subcomponents $(\mathbf{x}_1, \ldots, \mathbf{x}_g)$. A decomposition is called the *ideal decomposition*, if there exists an optimization problem for each subcomponent $\mathbf{x}_i, \forall i = 1, \ldots, g$, say $\mathcal{P}_i(\mathbf{x}_i)$, so that the optimal solution to the original optimization problem $\mathcal{P}(\mathbf{x})$ can be obtained by solving the $\mathcal{P}_i(\mathbf{x}_i)$'s separately. Then, the key issue of CC is to find such an ideal decomposition.

Take the following constrained optimization problem with equality constraints (inequalities can be transformed into equalities by adding slack variables) as an example:

$$\begin{aligned} \mathcal{P}(\mathbf{x}): \quad \min \ & f(\mathbf{x}) = \sum_{i=1}^{g} f_i(\mathbf{x}_i) \\ s.t. \ & h_1(\mathbf{x}_1) = 0 \\ & \ldots \\ & h_g(\mathbf{x}_g) = 0 \end{aligned}$$

Obviously, $(\mathbf{x}_1, \ldots, \mathbf{x}_g)$ is an ideal decomposition and the $\mathcal{P}_i(\mathbf{x}_i)$'s are defined as follows:

$$\begin{aligned} \mathcal{P}_i(\mathbf{x}_i): \quad \min \ & f_i(\mathbf{x}_i) \\ s.t. \ & h_i(\mathbf{x}_i) = 0 \end{aligned}$$

In LSCARP, finding the ideal decomposition is quite challenging, since the proportion of the ideal decomposition among all the possible decompositions is very low. Basically, the domain knowledge is not enough to obtain the ideal decomposition for LSCARP beforehand. In this case, the dynamic decomposition methods [19] [20] are more promising than the static ones, as they have higher chance to obtain the ideal decomposition.

In this paper, a simple but effective *Random Route Grouping* (RRG) decomposition scheme is developed. RRG uses the route information of the best-so-far solution so that the decomposition is guaranteed to be improved as the best-so-far solution is improved. RRG is combined with the CC framework proposed in [19], and the MAENS [12] to optimize the subcomponents. The resultant algorithm is evaluated on the EGL-G set with various parameter settings, and the results demonstrate the efficacy of the CC framework and RRG.

The rest of the paper is organized as follows: First, CARP is introduced in Section II. LSCARP is a special case of CARP which has more than 300 required edges. In Section III, the CC framework for LSCARP is derived from the one proposed in [19]. Then, in Section IV, RRG is developed. The experimental studies are carried out in Section V. Finally, Section VI gives the conclusion and future work.

II. Capacitated Arc Routing Problem

CARP is defined on a connected graph $G(V, E, A)$, where V, E and A are the set of vertices, edges and arcs (directed edges), respectively. There is a subset $Z \subseteq E \cup A$ which is called the *task* set. Each element $z \in Z$ is then called a task. Each task $z \in Z$ is associated with a positive *demand* $d(z) > 0$, and a positive *serving cost* $sc(z) > 0$. Besides, each edge (v_i, v_j) or arc $\langle v_i, v_j \rangle$ has a positive *deadheading cost* $dc(v_i, v_j) > 0$, standing for the cost of traversing from v_i to v_j without service. Here, the graph is assumed to be symmetric, and thus $dc(v_i, v_j) = dc(v_j, v_i)$ for each edge (v_i, v_j). The services are finished by a fleet of vehicles with identical *capacity* of Q locating at the *depot* $v_0 \in V$. Then, CARP aims to design the routes of the vehicles so that the the *total cost* (sum of all the serving and deadheading costs) is minimized subject to the following constraints:

- Each vehicle starts and ends at the depot;
- Each task is served exactly once;
- The total demand of the tasks served by each vehicle cannot exceed its capacity Q.

The last constraint is often called the *capacity constraint*.

Here, we adopt the problem formulation used in [15]. Concretely, each edge task is assigned two IDs (say x_1 and x_2), one for each direction, and each arc task is assigned one ID x. All the IDs are unique positive integers. For an ID $x \in \mathbb{N}^+$, the *tail node* $tn(x)$, *head node* $hn(x)$, deadheading cost $dc(x)$, serving cost $sc(x)$, demand $d(x)$ and *inverse ID* $inv(x)$ are associated as follows: For an edge task (v_i, v_j) and its corresponding IDs x_1 and x_2,

- $hn(x_1) = tn(x_2) = v_i$, $tn(x_1) = hn(x_2) = v_j$;
- $dc(x_1) = dc(x_2) = dc(v_i, v_j)$;
- $sc(x_1) = sc(x_2) = sc(v_i, v_j)$;
- $d(x_1) = d(x_2) = d(v_i, v_j)$;
- $inv(x_1) = x_2$, $inv(x_2) = x_1$.

For an arc task $\langle v_i, v_j \rangle$ and its ID x,

- $hn(x) = v_i$, $tn(x) = v_j$;
- $dc(x) = dc(v_i, v_j)$, $sc(x) = sc(v_i, v_j)$;
- $d(x) = d(v_i, v_j)$, $inv(x) = -1$.

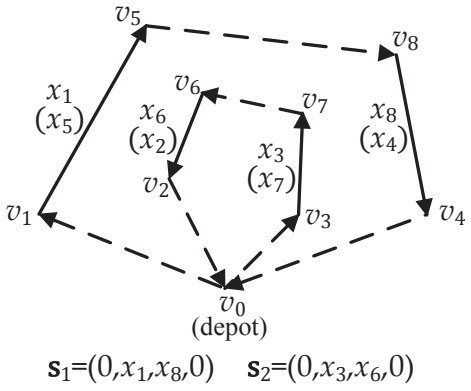

Fig. 1. An example of a CARP candidate solution.

Since all the IDs are positive, $inv(x) = -1$ indicates that x has no inverse ID. Finally, the ID 0 is defined to represent the depot loop as follows:

- $tn(0) = hn(0) = v_0$;
- $dc(0) = sc(0) = d(0) = 0$;
- $inv(0) = 0$.

Using the above notations, a CARP candidate solution $\mathbf{s}$ can be represented as a set of routes $\mathbf{s} = \{\mathbf{s}_1, \ldots, \mathbf{s}_m\}$, each of which is a sequence of the IDs starting and ending at 0. That is, $\mathbf{s}_k = (0, x_{k1}, \ldots, x_{kn_k}, 0)$.

An example of the above representation is illustrated in Fig. 1, where the bold lines represent tasks $Z = \{(v_1, v_5), (v_2, v_6), (v_3, v_7), (v_4, v_8)\}$, and the dashed lines represent the intermediate paths between the two end-nodes. The depot is denoted by v_0. Each task is assigned with two IDs, one standing for the current direction, and the other in the parenthesis indicating the inverse direction (e.g., ID x_1 from v_1 to v_5, while ID x_5 from v_5 to v_1). The candidate solution in the figure has two routes: $\mathbf{s}_1 = (0, x_1, x_8, 0)$ and $\mathbf{s}_2 = (0, x_3, x_6, 0)$. A more concise representation of $\mathbf{s}$ is to combine all the routes together as a single sequence with delimiters of 0 that distinguish different routes. For example, in Fig. 1, $\mathbf{s} = (0, x_1, x_8, 0, x_3, x_6, 0)$.

Then, CARP can be represented as follows:

$$\min_{x_{kl}} \; tc(\mathbf{s}) = \sum_{k=1}^{m} \sum_{l=0}^{n_k} \Big(sc(x_{kl}) + \Delta(x_{kl}, x_{k(l+1)}) \Big) \tag{1}$$

$$s.t.: \; x_{kl} \in ID(Z), \quad \forall\, 1 \le k \le m, 1 \le l \le n_k \tag{2}$$

$$x_{k_1 l_1} \neq x_{k_2 l_2}, \qquad \forall\, (k_1, l_1) \neq (k_2, l_2) \tag{3}$$

$$x_{k_1 l_1} \neq inv(x_{k_2 l_2}), \quad \forall\, (k_1, l_1) \neq (k_2, l_2) \tag{4}$$

$$\sum_{l=1}^{n_k} d(x_{kl}) \leqslant Q, \qquad \forall\, 1 \le k \le m \tag{5}$$

where $x_{k0} = x_{k(n_k+1)} = 0$. The function $\Delta(x_1, x_2)$ means the shortest distance from $tn(x_1)$ to $hn(x_2)$ in the graph, which can be obtained by Dijkstra's algorithm [22]. $ID(Z)$ stands for the ID set assigned to Z, and the inequality $(k_1, l_1) \neq (k_2, l_2)$ means that at least one of the inequalities $k_1 \neq k_2$ and $l_1 \neq l_2$ is satisfied. The objective (1) is the total cost of all the routes. Constraints (2)-(4) guarantee that each task is served exactly once. Constraint (5) is the capacity constraint.

III. Cooperative Co-evolution for LSCARP

The CC framework for LSCARP is derived from the one proposed by Yang *et al.* in [19], which uses a random grouping method to decompose a large scale optimization problem. It is described as follows:

Step 1 Set $i = 1$ to start a new cycle.
Step 2 Split the original n-dimensional vector $\mathbf{x} = (x_1, \ldots, x_n)$ into g non-overlapping l-dimensional subcomponents $\mathbf{x}_1, \ldots, \mathbf{x}_g$ ($g \cdot l = n$) randomly. Here, "randomly" means that each variable x_i ($i = 1, \ldots, n$) has the same chance to be assigned into any of the subcomponents.
Step 3 Optimize the subcomponent $\mathbf{x}_i$ by an Evolutionary Algorithm (EA) for a certain number of iterations;
Step 4 If $i < g$, then set $i = i + 1$ and go to Step 3;
Step 5 If the stopping criteria are met, then stop. Otherwise go to Step 1 for the next cycle.

When applying the above CC framework to LSCARP, each task may be considered as a variable. Then, in each cycle, the task set Z is decomposed into a number of non-overlapping task subsets $Z_1, \ldots, Z_g$ at Step 2. Then, for each Z_i, the optimization problem $\mathcal{P}_i(Z_i)$ can be defined as follows:

$$\min_{x_{kl}} \ tc\Big(\mathbf{s}(Z_i)\Big) = \sum_{k=1}^{m}\sum_{l=0}^{n_k}\Big(sc(x_{kl}) + \Delta(x_{kl}, x_{k(l+1)})\Big) \quad (6)$$

$$s.t.: \ x_{kl} \in ID(Z_i), \quad \forall\, 1 \le k \le m, 1 \le l \le n_k \quad (7)$$

$$x_{k_1 l_1} \ne x_{k_2 l_2}, \qquad \forall\, (k_1, l_1) \ne (k_2, l_2) \quad (8)$$

$$x_{k_1 l_1} \ne inv(x_{k_2 l_2}), \quad \forall\, (k_1, l_1) \ne (k_2, l_2) \quad (9)$$

$$\sum_{l=1}^{n_k} d(x_{kl}) \leqslant Q, \qquad \forall\, 1 \le k \le m \quad (10)$$

It is the same as Eqs. (1)–(5) except that it reduces the domain of x_{kl}'s by replacing Z with its subset Z_i, as shown in Eq. (7). In other words, $\mathcal{P}_i(Z_i)$ is a smaller sized CARP with the task set of Z_i instead of Z.

Given the above $\mathcal{P}_i(Z_i)$'s, one can prove that there must be an ideal decomposition $(Z_1, \ldots, Z_g)$ of Z, so that solving Eqs. (6)–(10) for $i = 1, \ldots, g$ is equivalent to solving Eqs. (1)–(5). In fact, let the optimal solution to $\mathcal{P}(Z)$ be $\mathbf{s}^* = \{\mathbf{s}_1^*, \ldots, \mathbf{s}_m^*\}$, one can simply define $Z_i = \{z | z \in \mathbf{s}_i^*\}$. In this way, $\mathbf{s}^*(Z_i) = \mathbf{s}_i^*$, and thus $tc(\mathbf{s}^*) = \sum_{i=1}^{g} tc(\mathbf{s}_i^*) = \sum_{i=1}^{g} tc(\mathbf{s}^*(Z_i))$. Therefore, the $\mathbf{s}^*(Z_i)$'s obtained from solving Eqs. (6)–(10) lead to $\mathbf{s}^*$ in the original Eqs. (1)–(5).

Next at Step 3, an EA is used to optimize each Z_i. To this end, a subpopulation $\mathbf{p}(Z_i) = \{\mathbf{p}_1(Z_i), \ldots, \mathbf{p}_N(Z_i)\}$ is maintained, where N is the predefined population size. Each individual $\mathbf{p}_j(Z_i) \in \mathbf{p}(Z_i)$ is a candidate solution to $\mathcal{P}_i(Z_i)$, and is evaluated by the objective function Eq. (6). This evaluation is essentially the same as the one based on the *context vector* [23] [24], which combines the evaluated individual with the best-so-far individuals of the other subcomponents, and then evaluates the combined individual. In LSCARP, let $\bar{\mathbf{s}}(Z \setminus Z_i)$ be the best-so-far individual to the subcomponent $Z \setminus Z_i$, then an individual with a smaller $tc(\mathbf{s}(Z_i))$ must have a smaller $tc(\mathbf{s}(Z_i) \cup \bar{\mathbf{s}}(Z \setminus Z_i)) = tc(\mathbf{s}(Z_i)) + tc(\bar{\mathbf{s}}(Z \setminus Z_i))$, since $tc(\bar{\mathbf{s}}(Z \setminus Z_i))$ can be seen as a constant that is independent from $\mathbf{s}(Z_i)$.

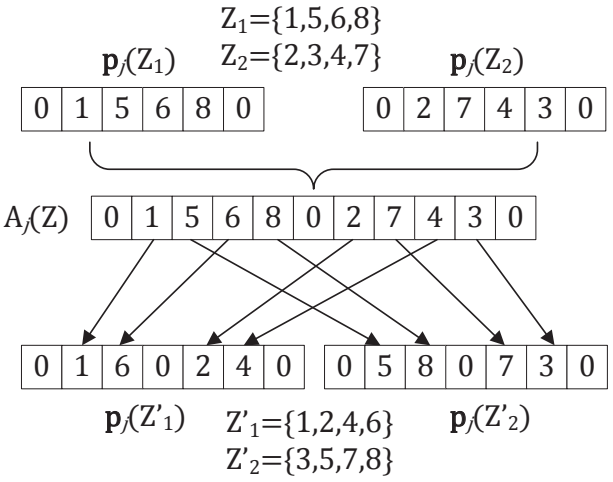

Fig. 2. An example of updating $\mathbf{p}_j(Z_1)$ and $\mathbf{p}_j(Z_2)$ for the next cycle.

Finally, after all the $\mathbf{p}(Z_i)$'s have been evolved by the EA, they are assembled to form an archive $\mathbf{A}(Z)$. That is, $\mathbf{A}_j(Z) = \{\mathbf{p}_j(Z_1), \ldots, \mathbf{p}_j(Z_g)\}$, $j = 1, \ldots, N$. Then, in the next cycle with the new decomposition $(Z_1', \ldots, Z_g')$, the corresponding subpopulations $\mathbf{p}(Z_i')$'s are obtained as follows: Given $(Z_1', \ldots, Z_g')$, for each $j \in \{1, \ldots, N\}$, the individual $\mathbf{p}_j(Z_i')$ is obtained by scanning the sequence of $\mathbf{A}_j(Z)$ from the beginning to the end, and removing the IDs belonging to $Z \setminus Z_i'$. In this way, the sequential structure of the subpopulations is maintained when updating them for the next cycle.

An example of updating the subpopulations for the next cycle is given in Fig. 2. Here, the current subcomponents are $Z_1 = \{1, 5, 6, 8\}$ and $Z_2 = \{2, 3, 4, 7\}$. First, the two individuals $\mathbf{p}_j(Z_1) \in \mathbf{p}(Z_1)$ and $\mathbf{p}_j(Z_2) \in \mathbf{p}(Z_2)$ are combined to obtain $\mathbf{A}_j(Z)$. Assume that the subcomponents for the next cycle are $Z_1' = \{1, 2, 4, 6\}$ and $Z_2' = \{3, 5, 7, 8\}$, then the new individual $\mathbf{p}_j(Z_1')$ ($\mathbf{p}_j(Z_2')$, resp.) is obtained by scanning the sequence of $\mathbf{A}_j(Z)$ and removing the tasks in Z_2' (Z_1', resp.). The other individuals of $\mathbf{p}(Z_1)$ and $\mathbf{p}(Z_2)$ are updated in the same way.

IV. Random Route Grouping Decomposition

In the CC framework described in Section III, Step 2 is the so-called random grouping. It can be directly adopted in decomposing LSCARP, i.e., each task $z \in Z$ has the same probability to be assigned into any of Z_i, $\forall i = 1, \ldots, g$. However, simply using the random grouping can by no means lead to satisfactory results due to the following two reasons. First, it is difficult to obtain the ideal decomposition by random sampling. In LSCARP, all the tasks in the same route must be placed in the same subcomponent. Usually, an LSCARP has many such subcomponents, each with quite a few tasks. As proved in [21], the probability of placing the interacting variables in the same subcomponent dramatically decreases as the subcomponent size increases. For example, suppose that the problem has 300 tasks, which can be divided into 30 10-task subcomponents, and $g = 2$. According to the formula in [21], the probability of decomposing all the tasks correctly in at least one of 50 cycles is 10^{-31}, which is nearly zero. Second, for each subcomponent, the optimization problem $\mathcal{P}_i(Z_i)$ is still NP-hard. Hence, even under the ideal decomposition, one

can hardly solve the $\mathcal{P}_i(Z_i)$'s within such a limited iterations in a single cycle. In this case, a more intelligent decomposition scheme than the random grouping is needed to bias to the more promising decompositions. Here, the effects of the constraints on the decomposition are considered and a decomposition scheme heuristic based on the routes of the best-so-far solution to the original problem is developed.

Intuitively, an intelligent decomposition scheme should be able to improve the decomposition scheme as more information is gathered during the search process. To this end, the evaluation of decomposition needs to be defined first. It is obvious that the measure $\sum_{i=1}^{g} tc(\mathbf{s}^*(Z_i))$ can be used to evaluate the decomposition $(Z_1, \ldots, Z_g)$, where $\mathbf{s}^*(Z_i)$ is the optimal solution to $\mathcal{P}_i(Z_i)$. That is, a better decomposition should have smaller $\sum_{i=1}^{g} tc(\mathbf{s}^*(Z_i))$. However, as mentioned before, $\mathbf{s}^*(Z_i)$ cannot be obtained as $\mathcal{P}_i(Z_i)$ cannot be solved exactly. Then, an alternative is to approximate the value of $\sum_{i=1}^{g} tc(\mathbf{s}^*(Z_i))$ by $\sum_{i=1}^{g} tc(\bar{\mathbf{s}}(Z_i))$, where $\bar{\mathbf{s}}(Z_i)$ is the best-so-far individual for Z_i. It is clear that

$$tc(\mathbf{s}^*(Z_i)) \leq tc(\bar{\mathbf{s}}(Z_i)),\ \forall\ Z_i \subseteq Z \tag{11}$$

$$\sum_{i=1}^{g} tc(\mathbf{s}^*(Z_i)) \leq \sum_{i=1}^{g} tc(\bar{\mathbf{s}}(Z_i)) \tag{12}$$

In other words, $\sum_{i=1}^{g} tc(\bar{\mathbf{s}}(Z_i))$ is an upper bound of $\sum_{i=1}^{g} tc(\mathbf{s}^*(Z_i))$. When comparing between two unknown variables with no other prior information, the one with smaller upper bound should be more likely to be smaller.

On the other hand, it is known that the best-so-far solution $\bar{\mathbf{s}}$ to the original problem keeps improving during the search process. If the Z_i's are set in such a way that $\sum_{i=1}^{g} tc(\bar{\mathbf{s}}(Z_i)) = tc(\bar{\mathbf{s}})$, then the decomposition can be improved along with the improvement of $\bar{\mathbf{s}}$. Similar to generating $\mathbf{p}_j(Z_i)$'s from $A_j(Z)$, each $\bar{\mathbf{s}}(Z_i)$ is obtained by scanning the sequence of $\bar{\mathbf{s}}$ and removing the tasks not belonging to Z_i. For example, in Fig. 2, one can assume that $\bar{\mathbf{s}} = \mathbf{A}_j(Z)$. Then, $\mathbf{p}_j(Z'_1)$ and $\mathbf{p}_j(Z'_2)$ can be seen as $\bar{\mathbf{s}}(Z'_1)$ and $\bar{\mathbf{s}}(Z'_2)$ corresponding to $Z'_1 = \{1, 2, 4, 6\}$ and $Z'_2 = \{3, 5, 7, 8\}$, respectively. It can be seen that the constraints affect the sequences of the $\bar{\mathbf{s}}(Z_i)$'s (e.g., if $x_{k(l+1)} \notin Z_i$, then $(x_{kl}, x_{k(l+1)})$ cannot occur in $\bar{\mathbf{s}}(Z_i)$). Thus, not all the decompositions satisfy that $\sum_{i=1}^{g} tc(\bar{\mathbf{s}}(Z_i)) = tc(\bar{\mathbf{s}})$. A natural way to satisfy the above condition is to divide the routes of $\bar{\mathbf{s}}$ into non-overlapping groups of routes. For example, in Fig. 2, we can set $\bar{Z}_1 = \{1, 5, 6, 8\}$ and $\bar{Z}_2 = \{2, 7, 4, 3\}$ according to the routes of $\bar{\mathbf{s}} = \mathbf{A}_j(Z)$. In this way, $\cup_{i=1}^{g} \bar{\mathbf{s}}(\bar{Z}_i) = \bar{\mathbf{s}}$ and thus $\sum_{i=1}^{g} tc(\bar{\mathbf{s}}(\bar{Z}_i)) = tc(\bar{\mathbf{s}})$.

In summary, to guarantee that the decomposition keeps on improving in terms of the approximated measure $\sum_{i=1}^{g} tc(\bar{\mathbf{s}}(Z_i))$, one can divide the routes of the best-so-far solution $\bar{\mathbf{s}}$ into groups. Similar to random grouping, we adopt the Random Route Grouping (RRG) strategy here, which means that each route of $\bar{\mathbf{s}}$ has the same chance to be assigned into any of the subcomponents. In other words, RRG is the same as random grouping except that it operates on the routes of the best-so-far solution instead of the tasks directly.

In practice, when grouping the routes of the best-so-far solution, the number of routes may not be divided exactly by g. In this case, the size of each group is set to be as equal as possible. For example, suppose that there are 20 routes and $g = 3$, then the groups have 7, 7 and 6 routes, respectively.

Algorithm 1 $(Z_1, \ldots, Z_g) = \text{RRG}(\bar{\mathbf{s}}, g, t)$

```
1:  if t = 1 then
2:      (Z_1, ..., Z_g) = k-medoids(Δ_{|Z|×|Z|}, g);
3:  else
4:      for i = 1 → g do
5:          m_i = ⌊m/g⌋;
6:      end for
7:      for i = 1 → m − g · ⌊m/g⌋ do
8:          m_i ← m_i + 1;
9:      end for
10:     for i = 1 → g do
11:         Z_i = {};
12:     end for
13:     (p_1, ..., p_m) = RandPerm(m);
14:     for i = 1 → g do
15:         j_1 = Σ_{k=1}^{i−1} m_k + 1, j_2 = Σ_{k=1}^{i−1} m_k + m_i;
16:         for j = j_1 → j_2 do
17:             for all z ∈ s̄_{p_j} do
18:                 Z_i ← Z_i ∪ z;
19:             end for
20:         end for
21:     end for
22: end if
23: return (Z_1, ..., Z_g);
```

Initially, there is no best-so-far solution that can be used to decompose the problem. Thus, the initial decomposition needs to be decided additionally. Here, we adopt the k-medoids heuristic [25] that clusters the tasks into k groups according to the distance matrix between them. Intuitively, to minimize the total cost, the tasks which are closer to each other should be more likely to be linked, and thus be in the same route. Thus, the k-medoids heuristic can obtain a relatively good initial decomposition without knowing any information about the best-so-far solution.

The pseudo code of RRG is described in Algo. 1. In line 2, $\Delta_{|Z|\times|Z|}$ is the distance matrix between each pair of tasks. In line 13, the function $RandPerm(m)$ is a random permutation of the vector $(1, \ldots, m)$.

Finally, the CC framework with RRG is described in Algo. 2. In line 8, each individual $\mathbf{p}_j(Z_i) \in \mathbf{p}(Z_i)$ is obtained by scanning the sequence of $\mathbf{A}(Z)$ and removing the tasks belonging to $Z \setminus Z_i$. In line 10, $\bar{\mathbf{s}}^{(t)}(Z_i)$ is obtained from $\bar{\mathbf{s}}$ in the same way.

V. Experimental Studies

The performance of the proposed CC framework may largely depend on the following two parameters: the number of subcomponents g and the number of cycles. First, as g increases, the number of possible decompositions increases dramatically ($O(g^m)$, where m is the number of routes in $\mathbf{s}$). This may make it much harder to obtain promising decompositions by RRG (unless the routes are clearly distributed in g clusters). Second, the number of cycles should be neither too small nor too large. If it is too small, the chance to obtain promising decompositions is small. If it is too large, the computational resources allocated for each cycle is not enough

Algorithm 2 $\bar{\mathrm{s}} = \mathrm{RRGCC}(\mathcal{P}(Z), g)$

Set $tc(\bar{\mathbf{s}}) = \infty$ and $t = 1$;
repeat
  $(Z_1, \ldots, Z_g) = \mathrm{RRG}(\bar{\mathbf{s}}, g, t)$;
  for $i = 1 \rightarrow g$ **do**
    if $t = 1$ **then**
      Initialize $\mathbf{p}(Z_i)$;
    else
      Obtain $\mathbf{p}(Z_i)$ from the archive $\mathbf{A}(Z)$;
    end if
    Obtain $\bar{\mathbf{s}}^{(t)}(Z_i)$ from $\bar{\mathbf{s}}$;
    $(\bar{\mathbf{s}}^{(t)}(Z_i), \mathbf{p}(Z_i)) = EA(\bar{\mathbf{s}}^{(t)}(Z_i), \mathbf{p}(Z_i))$;
  end for
  Assemble all the $\mathbf{p}(Z_i)$'s to form $\mathbf{A}(Z)$;
  $\bar{\mathbf{s}}^{(t)} = \cup_{i=1}^{g} \bar{\mathbf{s}}^{(t)}(Z_i)$;
  if $tc(\bar{\mathbf{s}}^{(t)}) < tc(\bar{\mathbf{s}})$ **then**
    $\bar{\mathbf{s}} = \bar{\mathbf{s}}^{(t)}$;
  end if
  $t \leftarrow t + 1$;
until Stopping criteria are met
return $\bar{\mathbf{s}}$;

to exploit the subcomponents adequately. To investigate the effects of the above two parameters, experimental studies were carried out by comparing the algorithms with various g values and numbers of cycles. Note that when $g = 1$, there is only one subcomponent, and thus there is no decomposition.

A. Parameter Settings

First, the algorithm to optimize each subcomponent has to be decided. Here, we select MAENS [12], which is competitive to solve small and medium sized CARPs. Then, the resultant algorithm is evaluated on the EGL-G set, which consists of 10 large instances defined on a road network with 255 vertices and 375 edges in the Lancashire County, UK. Different instances have different task sets and capacities.

The parameter settings of the proposed algorithm are given in Table I. Here, g is set to 1, 2 and 3, and the number of cycles is set to 25, 50 and 100. Note that the generations of MAENS in each cycle is naturally decided by the maximal generations and number of cycles. For example, with the maximal generations of 500 and 50 cycles, there are $500/50 = 10$ generations of MAENS for each subcomponent in each cycle.

For each parameter setting, 30 independent runs were conducted on all the test instances. Then, the mean and standard deviation of the 30 results were computed. The algorithms were implemented by C++, compiled by GNU Compiler Collection (GCC) for windows and run on the CPU Intel Core i7-2600 @ 3.4GHz.

B. Results and Analysis

The performance of the compared algorithms are shown in Table II, along with the features of the test instances. The columns "$|V|$", "$|E|$" and "$|Z|$" stand for the number of vertices, edges and tasks, respectively. The column "τ" is the minimal number of vehicles required to serve all the tasks, which is obtained as follows:

$$\tau = \left\lceil \frac{\sum_{z \in Z} d(z)}{Q} \right\rceil \tag{13}$$

TABLE I. THE PARAMETER SETTINGS OF THE PROPOSED ALGORITHM

Parameter	Description	Value
g	Number of subcomponents	1, 2, 3
$psize$	Population size	30
$offsize$	Offspring population size	$6 \cdot psize$
P_{ls}	Probability of local search	0.2
gen	Maximal generations	500
$cycle$	Number of cycles	25, 50, 100

In general, a larger τ leads to a higher difficulty of the problem.

In the table, for each instance and each algorithm, the values outside and inside the parenthesis are the mean and standard deviation of the 30 independent results. The minimal mean value of each row is marked with †. Besides, for each instance, the 30 independent results of $g = 1$ is compared with that of $g > 1$ statistically using Wilcoxon's rank sum test [26] under the significance level of 0.05. If the results of $g = 1$ are significantly smaller than that of all the algorithms with $g > 1$, then the mean of $g = 1$ is marked in bold. Otherwise, for each algorithm with $g > 1$, if its results are significantly smaller than that of $g = 1$, its mean is marked in bold.

From Table II, it can be seen that for only 2 out of 10 instances, $g = 1$ obtained significantly better results than that of all the algorithms with $g > 1$. In other words, for 8 out of 10 the instances, there is at least one parameter setting of the decomposition-based algorithm that can obtain statistically comparable results to that of its counterpart without decomposition. For 7 out of 10 instances, the results of $g = 2$ are better than that of $g = 3$. This is consistent with the expectation that a larger g tends to make it more difficult to obtain the promising decompositions. Besides, for $g = 2$, $cycle = 25$ showed the best overall performance, while for $g = 3$, $cycle = 50$ performed the best. This indicates that the best number of cycles is proportional to the value of g. It can be explained as follows: since it is harder to obtain promising decomposition for larger g, more cycles is needed to increase such probability.

The average runtime of the compared algorithms are given in Table III. It is obvious that as g increases, the runtime decreases dramatically. Besides, with the same g, the number of cycles does not much affect the runtime.

In summary, according to Tables II and III, one can conclude that by decomposing the problem into smaller subcomponents, statistically comparable results can be obtained in a much shorter time.

A more illustrative comparison is made for the convergence curves, which are shown in Figures 3–12. In the figures, the x-axis stands for the computational time in seconds, and the y-axis is the average total cost of the best-so-far solutions over the 30 independent runs. For the sake of clarity, the x-axis starts from 100 seconds without losing any accuracy of the convergence. From the figures, it can be seen that for most of the instances, most of the curves of $g > 1$ are below that of $g = 1$ (except only for EGL-G2-A). This indicates that given the same computational time budget, the decomposition-based algorithms show obvious advantage over the counterpart with no decomposition.

It is worth noting that for EGL-G1-D, EGL-G2-B and EGL-G2-D, the curves of $g = 3$ are obviously lower than that

TABLE II. THE RESULTS OF THE COMPARED ALGORITHMS ON THE EGL-G SET. FOR EACH INSTANCE, THE MINIMAL MEAN VALUE OF THE COMPARED ALGORITHMS IS MARKED WITH †. FOR EACH ALGORITHM WITH $g > 1$, IF ITS RESULTS ARE STATISTICALLY SIGNIFICANTLY SMALLER THAN THAT OF $g = 1$, THEN THE CORRESPONDING MEAN VALUE IS MARKED IN BOLD. OTHERWISE, IF THE RESULTS OF $g = 1$ ARE STATISTICALLY SIGNIFICANTLY SMALLER THAN THAT OF ALL THE ALGORITHMS WITH $g > 1$, THEN THE CORRESPONDING MEAN VALUE IS MARKED IN BOLD.

Name	$\|V\|$	$\|E\|$	$\|Z\|$	τ	$g = 1$	$g = 2$			$g = 3$		
						$cycle = 25$	$cycle = 50$	$cycle = 100$	$cycle = 25$	$cycle = 50$	$cycle = 100$
G1-A	255	375	347	20	1009302†(4210)	1010934(3734)	1009922(4979)	1012252(5780)	1014103(5414)	1011281(5015)	1013051(4157)
G1-B	255	375	347	25	1128114†(5552)	1129905(5550)	1130765(6509)	1132133(6528)	1136521(7197)	1136291(6845)	1135963(7206)
G1-C	255	375	347	30	1255709†(4700)	1256814(4658)	1258425(5779)	1257493(4387)	1259178(3063)	1259252(3976)	1260989(4254)
G1-D	255	375	347	35	1389399(5645)	1392409(5992)	1392228(6227)	1393108(6177)	1386444†(5147)	1388179(4637)	1388315(4376)
G1-E	255	375	347	40	1535905(7776)	1534496(6798)	1533000†(6202)	1535933(6549)	1536948(8173)	1535854(7478)	1538175(6661)
G2-A	255	375	375	22	**1109376**†(4672)	1112959(7223)	1113518(5801)	1113959(5442)	1117274(9514)	1116627(7645)	1116277(7408)
G2-B	255	375	375	27	1225361(5048)	1226477(6030)	1228069(4777)	1229522(6206)	1225341†(4605)	1226656(5264)	1227541(8833)
G2-C	255	375	375	32	1358398(5101)	1358374†(4117)	1358543(5724)	1359923(7239)	1365459(5379)	1364446(6627)	1365075(6853)
G2-D	255	375	375	37	**1498334**†(6319)	1502574(8255)	1505581(7346)	1503352(8972)	1501094(6029)	1502667(5605)	1501331(5106)
G2-E	255	375	375	42	1642124(7545)	1641750†(5467)	1642260(7558)	1643928(7760)	1642797(4825)	1643586(8155)	1646596(9223)
Overall	-	-	-	-	1315202(5657)	1316669(5782)	1317231(6090)	1318160(6504)	1318516(5935)	1318484(6125)	1319331(6408)

TABLE III. THE AVERAGE RUNTIME (IN SECONDS) OF THE COMPARED ALGORITHMS FOR THE EGL-G SET.

Name	$g = 1$	$g = 2$			$g = 3$		
		$cycle = 25$	$cycle = 50$	$cycle = 100$	$cycle = 25$	$cycle = 50$	$cycle = 100$
G1-A	2453	1277	1280	1328	797	799	818
G1-B	2058	1199	1213	1250	761	765	786
G1-C	1841	1128	1151	1192	738	745	767
G1-D	1655	1007	1027	1080	731	741	758
G1-E	1558	930	936	983	736	743	761
G2-A	2593	1434	1447	1480	906	907	927
G2-B	2268	1375	1384	1431	855	864	891
G2-C	2016	1251	1266	1324	856	863	888
G2-D	1850	1132	1150	1205	842	847	874
G2-E	1769	1054	1069	1115	847	860	886
Overall	2006	1179	1192	1239	807	813	836

of $g = 2$. It may be caused by the clearly clustered distribution of the routes. Our future studies will investigate the reasons.

VI. CONCLUSION

In this paper, the practically important LSCARP is investigated. To search effectively in the huge solution space, we adopt the divide-and-conquer method in the form of the CC framework. When decomposing LSCARP, the RRG scheme is developed to bias the search to the more promising decompositions. It is proved that the decomposition is guaranteed to be improved as the best-so-far solution is improved during the search process. The experimental studies verified the efficacy of the proposed CC framework with RRG for solving LSCARP.

The future work includes the following aspects: First, the random grouping of the routes in RRG is still simple. More information such as the distance between tasks may be employed to further improve the quality of the decomposition scheme. Second, the effect of g on different instances is to be investigated to decide the best g according to the characteristics of the given instance. Finally, the tradeoff between the number of cycles and the degree of exploitation in each cycle is to be further investigated, which may depend on g and the size of each subcomponent.

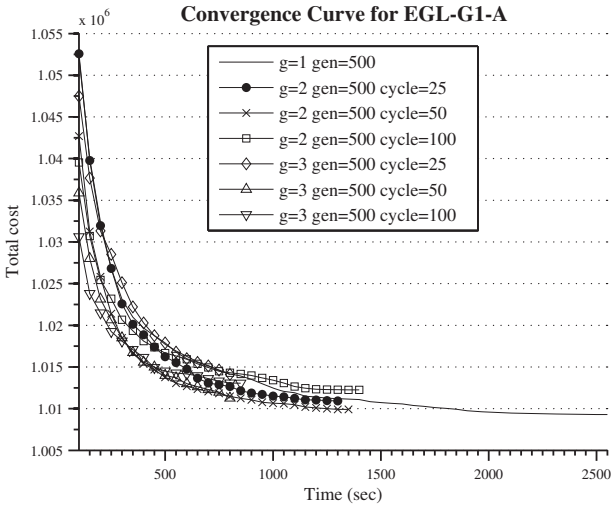

Fig. 3. The convergence curve of the compared algorithms on EGL-G1-A.

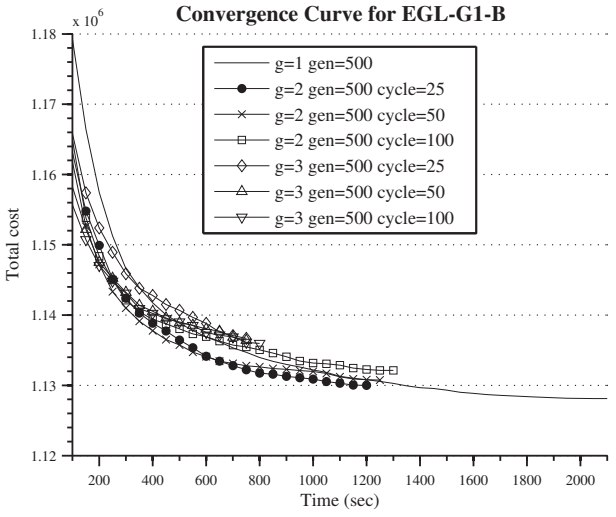

Fig. 4. The convergence curve of the compared algorithms on EGL-G1-B.

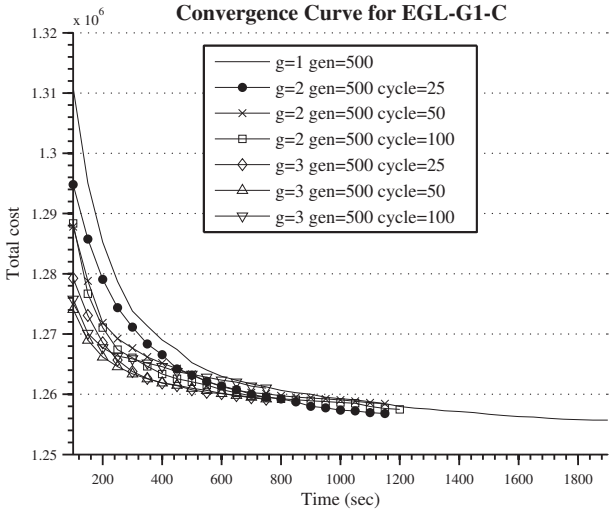

Fig. 5. The convergence curve of the compared algorithms on EGL-G1-C.

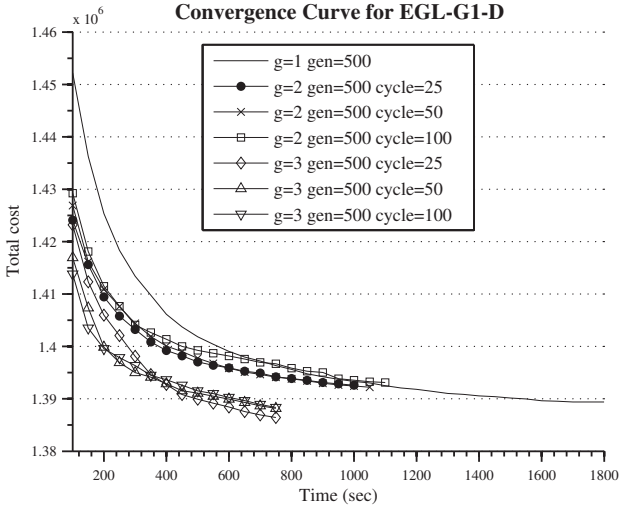

Fig. 6. The convergence curve of the compared algorithms on EGL-G1-D.

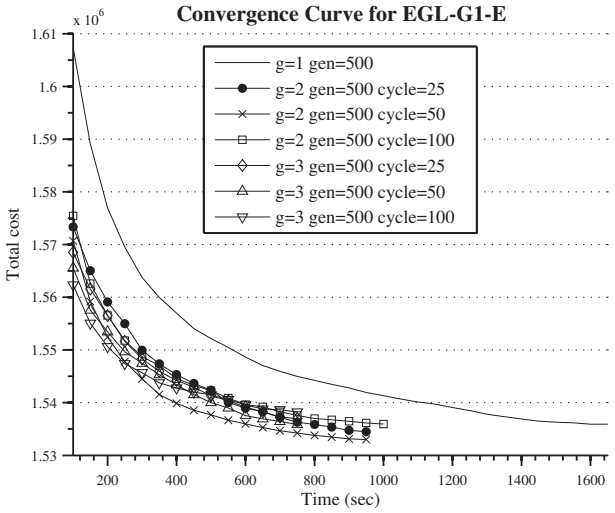

Fig. 7. The convergence curve of the compared algorithms on EGL-G1-E.

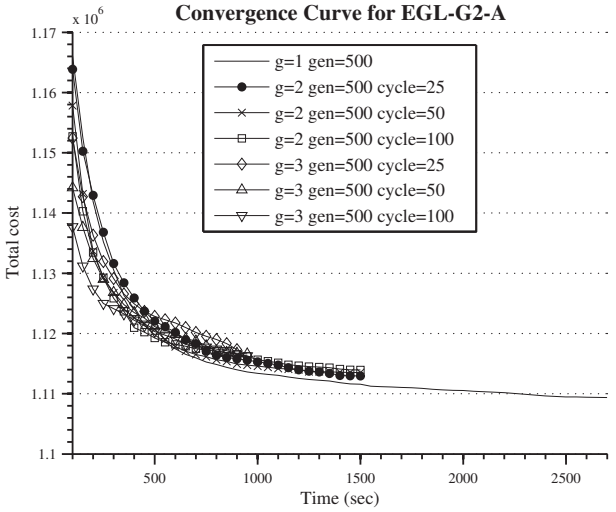

Fig. 8. The convergence curve of the compared algorithms on EGL-G2-A.

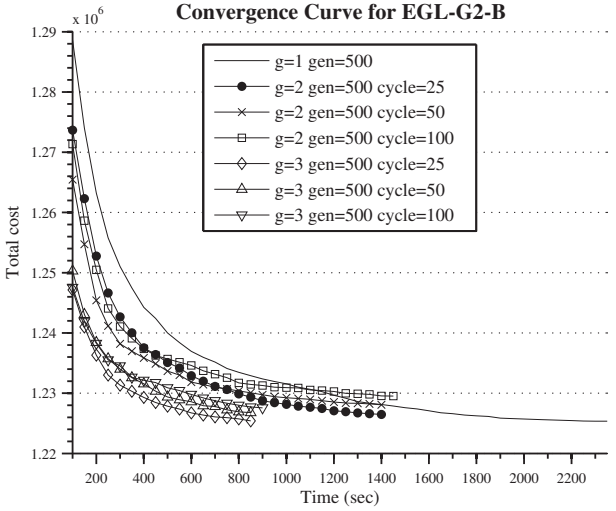

Fig. 9. The convergence curve of the compared algorithms on EGL-G2-B.

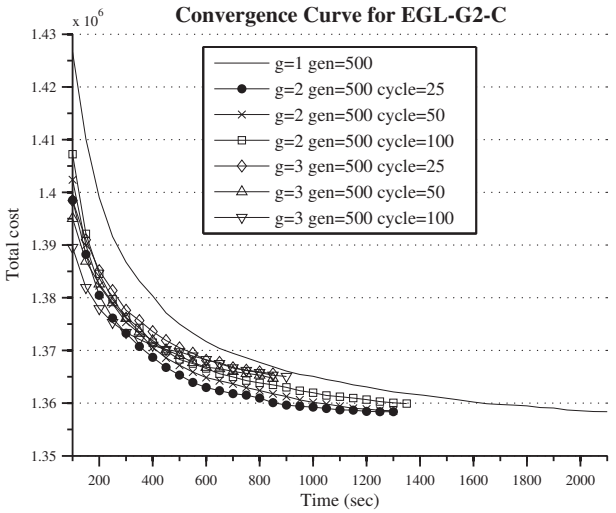

Fig. 10. The convergence curve of the compared algorithms on EGL-G2-C.

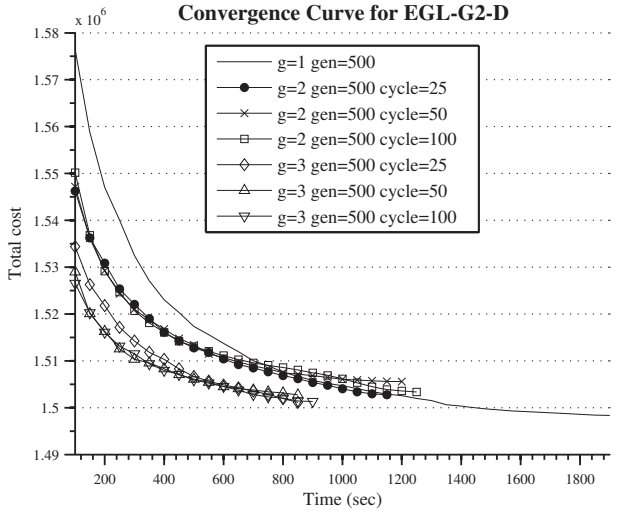

Fig. 11. The convergence curve of the compared algorithms on EGL-G2-D.

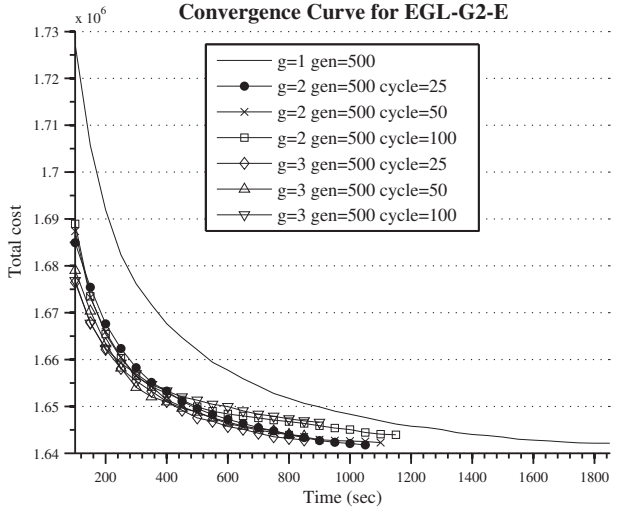

Fig. 12. The convergence curve of the compared algorithms on EGL-G2-E.

ACKNOWLEDGMENT

This work was supported by an ARC Discovery grant (No. DP120102205) and an EPSRC grant (No. EP/I010297/1). Xin Yao is supported by a Royal Society Wolfson Research Merit Award.

REFERENCES

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

[2] F. Chu, N. Labadi, and C. Prins, "A scatter search for the periodic capacitated arc routing problem," *European Journal of Operational Research*, vol. 169, no. 2, pp. 586–605, 2006.

[3] H. Handa, L. Chapman, and X. Yao, "Robust Salting Route Optimization Using Evolutionary Algorithms," in *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, vol. 1. Springer, 2006, pp. 10 455–10 462.

[4] M. Polacek, K. Doerner, R. Hartl, and V. Maniezzo, "A variable neighborhood search for the capacitated arc routing problem with intermediate facilities," *Journal of Heuristics*, vol. 14, no. 5, pp. 405–423, 2008.

[5] B. Golden and R. Wong, "Capacitated arc routing problems," *Networks*, vol. 11, no. 3, pp. 305–316, 1981.

[6] J. Brandão 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.

[7] Y. Mei, K. Tang, and X. Yao, "A Global Repair Operator for Capacitated Arc Routing Problem," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 39, no. 3, pp. 723–734, 2009.

[8] X. Chen, "Maens+: A divide-and-conquer based memetic algorithm for capacitated arc routing problem," in *2011 Fourth International Symposium on Computational Intelligence and Design (ISCID)*, vol. 1. IEEE, 2011, pp. 83–88.

[9] P. Beullens, L. Muyldermans, D. Cattrysse, and D. Van 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. Lacomme, C. Prins, and W. Ramdane-Cherif, "Competitive memetic algorithms for arc routing problems," *Annals of Operations Research*, vol. 131, no. 1, pp. 159–185, 2004.

[11] Y. Mei, K. Tang, and X. Yao, "Improved memetic algorithm for capacitated arc routing problem," in *Proceedings of the 2009 IEEE Congress on Evolutionary Computation*, 2009, pp. 1699–1706.

[12] K. Tang, Y. Mei, and X. Yao, "Memetic Algorithm with Extended Neighborhood Search for Capacitated Arc Routing Problems," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 5, pp. 1151–1166, 2009.

[13] L. Feng, Y. Ong, Q. Nguyen, and A. Tan, "Towards probabilistic memetic algorithm: An initial study on capacitated arc routing problem," in *Proceedings of the 2010 IEEE Congress on Evolutionary Computation*, 2010, pp. 18–23.

[14] H. Fu, Y. Mei, K. Tang, and Y. Zhu, "Memetic algorithm with heuristic candidate list strategy for capacitated arc routing problem," in *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2010.

[15] Y. Mei, K. Tang, and X. Yao, "Decomposition-based memetic algorithm for multiobjective capacitated arc routing problem," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 2, pp. 151–165, 2011.

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

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

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

[19] Z. Yang, K. Tang, and X. Yao, "Large scale evolutionary optimization using cooperative coevolution," *Information Sciences*, vol. 178, no. 15, pp. 2985–2999, 2008.

[20] M. Omidvar, X. Li, and X. Yao, "Cooperative co-evolution with delta grouping for large scale non-separable function optimization," in *Proceedings of the 2010 IEEE Congress on Evolutionary Computation*, 2010, pp. 1762–1769.

[21] M. Omidvar, X. Li, Z. Yang, and X. Yao, "Cooperative co-evolution for large scale optimization through more frequent random grouping," in *Proceedings of the 2010 IEEE Congress on Evolutionary Computation*, 2010, pp. 1–8.

[22] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[23] F. Van den Bergh and A. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 225–239, 2004.

[24] X. Li and Y. Yao, "Cooperatively coevolving particle swarms for large scale optimization," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 2, pp. 1–15, 2011.

[25] L. Kaufman and P. Rousseeuw, "Clustering by means of medoids," *Statistical data analysis based on the L1-norm and related methods*, vol. 405, 1987.

[26] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.