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 size constraints.

CARP has many applications in the logistics field, such as waste collection [2], winter gritting [3] and snow removal [4]. In spite of extensive 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 studies 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] 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 \(x\) is decomposed into a number of non-overlapping subcomponents \((x_1, \ldots, x_g)\). A decomposition is called the ideal decomposition, if there exists an optimization problem for each subcomponent \(x_i\) for \(i=1,\ldots,g\), say \(P_i(x_i)\), so that the optimal solution to the original optimization problem \(P(x)\) can be obtained by solving the \(P_i(x_i)\)’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:

\[
P(x) : \min f(x) = \sum_{i=1}^{g} f_i(x_i) \\
\text{s.t. } h_1(x_1) = 0 \\
\quad \quad \quad \cdots \\
\quad \quad \quad h_g(x_g) = 0
\]

Obviously, \((x_1, \ldots, x_g)\) is an ideal decomposition and the \(P_i(x_i)\)’s are defined as follows:

\[
P_i(x_i) : \min f_i(x_i) \\
\text{s.t. } h_i(x_i) = 0
\]

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 \( (v_i, v_j) \) 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 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 \( x_2 \),

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

For an arc task \( (v_i, v_j) \) and its ID \( x \),

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

An example of a CARP candidate solution is shown in Fig. 1, where the bold lines represent tasks \( Z = \{(v_1, v_5), (v_2, v_6), (v_3, v_7), (v_4, v_9)\} \), 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: \( s_1 = (0, x_1, x_8, 0) \) and \( s_2 = (0, x_3, x_6, 0) \). A more concise representation of \( s \) is to combine all the routes together as a single sequence with delimiters of 0 that distinguish different routes. For example, in Fig. 1, \( s = (0, x_1, x_8, 0, x_3, x_6, 0) \).

Then, CARP can be represented as follows:

\[
\min_{x_{kl}} tc(s) = \sum_{k=1}^{m} \sum_{l=0}^{n_k} \left( sc(x_{kl}) + \Delta(x_{kl}, x_{k(l+1)}) \right) \quad (1)
\]

s.t. \( x_{kl} \in ID(Z), \; \forall 1 \leq k \leq m, 1 \leq l \leq n_k \)

\( x_{k1l_1} \neq x_{k2l_2}, \; \forall (k_1, l_1) \neq (k_2, l_2) \) \; (3)

\( x_{k1l_1} \neq inv(x_{k2l_2}), \; \forall (k_1, l_1) \neq (k_2, l_2) \) \; (4)

\[
\sum_{l=1}^{n_k} d(x_{kl}) \leq Q, \; \forall 1 \leq k \leq m \quad (5)
\]

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. [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 \( x_1, \ldots, x_g \) \((g \cdot l = n)\) 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 \( x_i \) 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 \( \mathcal{P}(Z_i) \) can be defined as follows:

\[
\min_{z_{kl}} tc\left(s(Z_i)\right) = \frac{m}{\sum_{k=1}^{g} \sum_{l=0}^{n_k} \left(se(x_{kl}) + \Delta(x_{kl}, x_{k(l+1)})\right)} \quad (6)
\]

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

\[x_{kl} \neq x_{k'l}, \quad (k_1, l_1) \neq (k_2, l_2) \quad (8)
\]

\[x_{kl} \neq inv(x_{k'l}), \quad (k_1, l_1) \neq (k_2, l_2) \quad (9)
\]

\[\sum_{i=1}^{n_k} d(x_{kl}) \leq Q, \quad \forall 1 \leq k \leq m \quad (10)
\]

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, \( \mathcal{P}(Z_i) \) is a smaller sized CARP with the task set of \( Z_i \) instead of \( Z \).

Given the above \( \mathcal{P}(Z_i) \)'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 \( s^* = (s_1^*, \ldots, s_n^*) \), one can simply define \( Z_i = \{z | z \in s_i^*\} \). In this way, \( s^*(Z_i) = s_i^* \) and thus \( tc(s^*) = \sum_{i=1}^{g} tc(s_i^*) = \sum_{i=1}^{g} tc(s^*(Z_i)) \). Therefore, the \( s^*(Z_i) \)'s obtained from solving Eqs. (6)–(10) lead to \( s^* \) in the original Eqs. (1)–(5).

Next at Step 3, an EA is used to optimize each \( Z_i \). To this end, a subpopulation \( p(Z_i) = \{p_1(Z_i), \ldots, p_{N_i}(Z_i)\} \) is maintained, where \( N_i \) is the predefined population size. Each individual \( p_j(Z_i) \in p(Z_i) \) is a candidate solution to \( \mathcal{P}(Z_i) \), 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 \( \tilde{s}(Z \setminus Z_i) \) be the best-so-far individual to the subcomponent \( Z \setminus Z_i \), then an individual with a smaller \( tc(\tilde{s}(Z_i)) \) must have a smaller \( tc(s(Z_i)) \). Since \( tc(\tilde{s}(Z_i)) = tc(\tilde{s}(Z \setminus Z_i)) + tc(s(Z \setminus Z_i)) \), the evaluation of \( s(Z_i) \) can be seen as a constant that is independent from \( s(Z_i) \).

Finally, after all the \( p(Z_i) \)'s have been evolved by the EA, they are assembled to form an archive \( A(Z) \). That is, \( A(Z) = \{p_1(Z_1), \ldots, p_g(Z_g)\} \), \( j = 1, \ldots, N \). Then, in the next cycle with the new decomposition \( (Z_1, \ldots, Z_g) \), the corresponding subpopulations \( p(Z_i)' \)s are obtained as follows: Given \( (Z_1, \ldots, Z_g) \), for each \( j = 1, \ldots, N \), the individual \( p_j(Z_i) \) is obtained by scanning the sequence of \( A(Z) \) 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 \( p_1(Z_1) \in p(Z_1) \) and \( p_1(Z_2) \in p(Z_2) \) are combined to obtain \( A_1(Z) \). 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 \( p_1(Z_1') \) (\( p_2(Z_2') \), resp.) is obtained by scanning the sequence of \( A_1(Z) \) and removing the tasks in \( Z_1' \) (\( Z_2' \), resp.). The other individuals of \( p(Z_1) \) and \( p(Z_2) \) 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 is assigned to any of the subcomponents with equal probability. Hence, even under the ideal decomposition, one
removing the tasks not belonging to \( \bar{Z} \) can be seen as
\[
\sum_{i=1}^{g} \text{tc}(s(Z_i)) = tc(s(\bar{Z})) = \sum_{i=1}^{g} \text{tc}(s(\bar{Z})) = \sum_{i=1}^{g} \text{tc}(s(Z_i)) 
\]
In other words, \( \sum_{i=1}^{g} \text{tc}(s(Z_i)) \) is an upper bound of \( \sum_{i=1}^{g} \text{tc}(s(Z_i)) \). 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 \( \bar{s} \) to the original problem keeps improving during the search process. If the \( Z_i \)'s are set in such a way that \( \sum_{i=1}^{g} \text{tc}(s(Z_i)) = tc(s(\bar{Z})) \), then the decomposition can be improved along with the improvement of \( \bar{s} \). Similar to generating \( p_i(Z_i) \)'s from \( A_i(Z) \), each \( s(Z_i) \) is obtained by scanning the sequence of \( \bar{s} \) and removing the tasks not belonging to \( Z_i \). For example, in Fig. 2, one can assume that \( \bar{s} = A_i(Z) \). Then, \( p_i(Z_i) \) and \( p_i(\bar{s}) \) can be seen as \( s(\bar{Z}) \) and \( \bar{s}(\bar{Z}) \) corresponding to \( Z_i = \{1, 2, 4, 6\} \) and \( Z_i = \{3, 5, 7, 8\} \), respectively. It can be seen that the constraints affect the sequences of \( s(Z_i) \)'s (e.g., if \( x_k(t_{i+1}) \notin Z_i \), then \( (x_k(t_{i+1}), x_k(t_{i+1}+1)) \) cannot occur in \( s(Z_i) \)). Thus, not all the decompositions satisfy that \( \sum_{i=1}^{g} \text{tc}(s(Z_i)) = tc(s) \). A natural way to satisfy the above condition is to divide the routes of \( \bar{s} \) into non-overlapping groups of routes. For example, in Fig. 2, we can set \( Z_1 = \{1, 5, 6, 8\} \) and \( Z_2 = \{2, 7, 4, 3\} \) according to the routes of \( \bar{s} = A_i(Z) \). In this way, \( \sum_{i=1}^{g} \text{tc}(s(Z_i)) = tc(s) \) and thus \( \sum_{i=1}^{g} \text{tc}(s(Z_i)) = tc(s) \).

In summary, to guarantee that the decomposition keeps \( \bar{s} \) on improving in terms of the approximated measure \( \sum_{i=1}^{g} \text{tc}(s(Z_i)) \), one can divide the routes of the best-so-far solution \( \bar{s} \) into groups. Similar to random grouping, we adopt the Random Route Grouping (RRG) strategy here, which means that each route of \( \bar{s} \) 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 \( g = 5 \), then the groups have 7, 7 and 6 routes, respectively.

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 Alg. 1. In line 2, \( \Delta_{|Z_i|} \) is the distance matrix between each pair of tasks. In line 13, the function \( \text{RandPerm}(m) \) is a random permutation of the vector \( (1, \ldots, m) \).

Finally, the CC framework with RRG is described in Alg. 2. In line 8, each individual \( p_i(Z_i) \in p(Z_i) \) is obtained by scanning the sequence of \( A_i(Z) \) and removing the tasks belonging to \( Z_i \). In line 10, \( s^{(i)}(Z_i) \) is obtained from \( s \) in the same way.

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

\begin{algorithm}
\begin{algorithmic}
\State if \( t = 1 \) then
\State \( (Z_1, \ldots, Z_g) = \text{k-medoids}(\Delta_{|Z|x|Z|}, g) \);
\EndIf
\State for \( i = 1 \rightarrow g \) do
\State \( m_i = |m/g| \);
\EndFor
\State for \( i = 1 \rightarrow g \) do
\State \( m_i \leftarrow m_i + 1 \);
\EndFor
\State for \( i = 1 \rightarrow g \) do
\State \( Z_i = \{\} \);
\EndFor
\State \( (p_1, \ldots, p_m) = \text{RandPerm}(m) \);
\State for \( i = 1 \rightarrow g \) do
\State \( j_1 = \sum_{k=1}^{i} m_k + 1, j_2 = \sum_{k=1}^{i-1} m_k + m_i \);
\EndFor
\State for all \( z \in s^{(i)} \) do
\State \( Z_i \leftarrow Z_i \cup z \);
\EndFor
\State return \( (Z_1, \ldots, Z_g) \);
\end{algorithmic}
\end{algorithm}

\[\text{V. EXPERIMENTAL STUDIES}\]

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 \( (O(g^q)) \), where \( m \) is the number of routes in \( s \). 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 \( s = \text{RRGCC}(P(Z), g) \)

1: Set \( tc(s) = \infty \) and \( t = 1 \);
2: repeat
3: \((Z_1, \ldots, Z_g) = \text{RRG}(s, g, t)\);
4: for \( i = 1 \rightarrow g \) do
5: if \( t = 1 \) then
6: Initialize \( p(Z_i) \);
7: else
8: \( \text{Obtain } p(Z_i) \text{ from the archive } A(Z) ; \)
9: end if
10: Obtain \( s^{(i)}(Z_i) \) from \( s \);
11: \((s^{(i)}(Z_i), p(Z_i)) = EA(s^{(i)}(Z_i), p(Z_i)) ; \)
12: end for
13: \( s \leftarrow \bigcup_{i=1}^{g} s^{(i)}(Z_i) \);  
14: if \( tc(s^{(i)}) < tc(s) \) then
15: \( s = s^{(i)} ; \)
16: end if
17: \( t \leftarrow t + 1 ; \)
18: until Stopping criteria are met
19: return \( s \);

In general, a larger \( \tau \) 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 of

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g )</td>
<td>Number of subcomponents</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>( psize )</td>
<td>Population size</td>
<td>30</td>
</tr>
<tr>
<td>( offsize )</td>
<td>Offspring population size</td>
<td>6 ( psize )</td>
</tr>
<tr>
<td>( P_s )</td>
<td>Probability of local search</td>
<td>0.2</td>
</tr>
<tr>
<td>( gen )</td>
<td>Maximal generations</td>
<td>500</td>
</tr>
<tr>
<td>( cycle )</td>
<td>Number of cycles</td>
<td>25, 50, 100</td>
</tr>
</tbody>
</table>

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

\[
\tau = \frac{\sum_{z \in Z} d(z)}{Q} \tag{13}
\]
TABLE II. THE RESULTS OF THE COMPARED ALGORITHMS ON THE EGL-G SET. FOR EACH INSTANCE, THE MINIMAL MEAN VALUE OF THE COMPARED ALGORITHMS WITH \( g = 1 \) IS MARKED WITH \( \dagger \), 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     | \(|V|\) | \(|E|\) | \(|Z|\) | \(\tau\) | \(g = 1\) | \(g = 2\) | \(g = 3\) |
|----------|--------|--------|--------|------|----------|----------|----------|
|          | cycle 25 | cycle 50 | cycle 100 | cycle 25 | cycle 50 | cycle 100 | cycle 25 | cycle 50 | cycle 100 |
| G1-A     | 255     | 375    | 347    | 20 | 1099302(24210) | 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(6628) | 1136521(7197) | 1136291(6845) | 1135963(7206) |
| G1-C     | 255     | 375    | 347    | 30 | 1255799(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) | 1386441(5147) | 1388179(4637) | 1388315(4376) |
| G1-E     | 255     | 375    | 347    | 40 | 1534496(6798) | 1535000(6202) | 1535933(6549) | 1536948(8173) | 1535854(7478) | 1538175(6661) | 1538175(6661) |
| G2-A     | 255     | 375    | 372    | 23 | 1119376(4632) | 1121599(7223) | 1131518(5801) | 1139595(5442) | 1117274(9514) | 1111627(7645) | 1116277(7480) |
| G2-B     | 255     | 375    | 375    | 27 | 1225361(5048) | 1226477(6030) | 1228069(4777) | 1229522(6206) | 1223541(4605) | 1226656(5264) | 1227541(8833) |
| G2-C     | 255     | 375    | 372    | 46 | 1388342(4707) | 1389523(7574) | 1396927(7399) | 1399973(8140) | 1392577(6423) | 1395260(7270) | 1396226(7064) |
| 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 | 1535793(6549) | 1536948(8173) | 1535933(6549) | 1536948(8173) | 1535854(7478) | 1538175(6661) | 1538175(6661) |

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.

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. 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