A Global Repair Operator for Capacitated Arc Routing Problem

Yi Mei, Ke Tang, *Member, IEEE*, and Xin Yao, *Fellow, IEEE*

***Abstract*—Capacitated arc routing problem (CARP) has attracted much attention during the last few years due to its wide applications in real life. Since CARP is NP-hard and exact methods are only applicable for small instances, heuristics and metaheuristic methods are widely adopted when solving CARP. This paper demonstrates one major disadvantage encountered by traditional search algorithms and proposes a novel operator named global repair operator (GRO) to address it. We further embed GRO in a recently proposed tabu search algorithm (TSA) and apply the resultant repair-based tabu search (RTS) algorithm to five well-known benchmark test sets. Empirical results suggest that RTS not only outperforms TSA in terms of quality of solutions but also converges to the solutions faster. Moreover, RTS is also competitive with a number of state-of-the-art approaches for CARP. The efficacy of GRO is thereby justified. More importantly, since GRO is not specifically designed for the referred TSA, it might be a potential tool for improving any existing method that adopts the same solution representation.**

***Index Terms*—Capacitated arc routing problem (CARP), global repair operator (GRO), heuristic search, tabu search.**

I. Introduction

THE CAPACITATED arc routing problem (CARP) is a classic problem with wide applications in real world, such as urban waste collection, post delivery, salting route optimization, winter gritting, etc. [1]. It involves determining a minimum cost routing plan for a set of vehicles, each of which is associated with a capacity constraint. Concretely, the CARP can be represented by a graph $G = (V, E, A)$, where the vertex set V, the edge set E, and the arc set A represent the set of intersections, two-way streets, and one-way streets, respectively. A vertex $dep \in V$ represents a central depot where a set of vehicles are based. Two subsets $E_R \subseteq E$ and $A_R \subseteq A$ are called task sets, consisting of tasks required to be served by the vehicles (e.g., the streets required to be cleaned in winter gritting problem). Each element $(t, h) \in E \cup A$ is associated with three costs, i.e., $d(t, h)$, $sc(t, h)$, and $dc(t, h)$, which indicates the demand, the cost of serving, and passing without serving from the tail vertex t to the head vertex h (i.e., deadheading), respectively. Note that, for nonrequired edges and arcs, $d(t, h) = 0$. The aim is to schedule the routes for each vehicle, so that the total cost of the routes is minimized. Each route starts and ends at the depot, and the total demand processed must not exceed vehicle capacity.

Since CARP is NP-hard [2], exact algorithms are only applicable for small instances. For this reason, various heuristics and metaheuristics have been proposed for CARP. To name a few, Golden and Wong proposed a constructive heuristic called augment–merge [2] in 1981. After that, Golden *et al.* proposed another constructive heuristic called path scanning [3]. Ulusoy improved path scanning to form a new heuristic called Ulusoy's heuristic [4]. Pearn proposed an approximate algorithm [5] and an augment–insert heuristic [6] for CARP. Mourao and Amado proposed a heuristic method for mixed CARP and demonstrated that it outperforms all the previous heuristics [7]. Amponsah and Salhi proposed an efficient constructive heuristic embedded with a look-ahead strategy and enhancement procedures [8]. Hertz *et al.* proposed a tabu search for CARP called CARPET [9] and a variable neighborhood descent (VND) algorithm [10]. Greistorfer proposed a tabu scatter search for arc routing problem [11]. Lacomme *et al.* proposed a competitive memetic algorithm (MA) for CARP [12] and a genetic algorithm for CARP and its extension [13]. Beullens *et al.* proposed a guided local search (GLS) method [14]. Handa *et al.* proposed an evolutionary algorithm for the salting route optimization, which is an application of CARP [15], [16]. Recently, Brandão and Eglese proposed a deterministic tabu search algorithm (TSA) [17].

The solution representation of a heuristic method is critical for conducting an effective search and thus has much influence on obtaining good solutions. In the literature of CARP, a solution is commonly encoded as a set of routes, each of which is an ordered list of vertices. Every vertex is associated with a zero–one variable, indicating whether the edge between this vertex and the successive vertex is served. TSA and CARPET directly use such *vertex encoding*. MA represents solutions as ordered lists of tasks. Since each task corresponds to a unique pair of vertices, MA can be viewed as using the vertex encoding as well.

In the literature, the vertex encoding has provided generally satisfactory results. However, as will be explained in detail in Section II-B, the existing algorithms adopting the vertex encoding are likely to overlook those promising infeasible solutions. Under this circumstance, the search will be ineffective. This paper presents a novel repair operator, namely, the global

Manuscript received April 21, 2008; revised August 5, 2008. This work was supported in part by the Fund for Foreign Scholars in University Research and Teaching Programs under Grant B07033 and in part by an Engineering and Physical Science Research Council (EPSRC) grant in U.K. under Grant EP/E058884/1. This paper was recommended by Associate Editor H. Takagi.

Y. Mei and K. Tang are with the Nature Inspired Computation and Applications Laboratory, Department of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China (e-mail: meiyi@mail.ustc.edu.cn; ketang@ustc.edu.cn).

X. Yao is with the Nature Inspired Computation and Applications Laboratory, Department of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China, and also with the Centre of Excellence for Research in Computational Intelligence and Applications, School of Computer Science, University of Birmingham, B15 2TT Birmingham, U.K. (e-mail: x.yao@cs.bham.ac.uk).

Digital Object Identifier 10.1109/TSMCB.2008.2008906

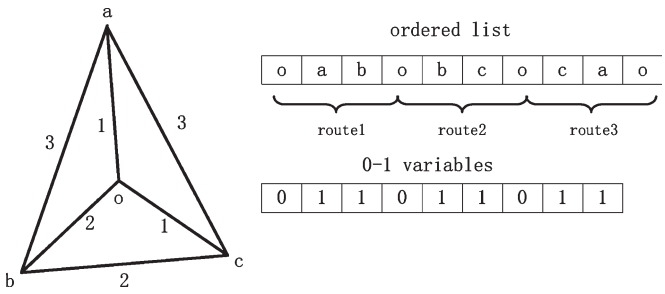


Fig. 1. Vertex encoding for a CARP solution.

repair operator (GRO), to handle such problems. The GRO can be easily embedded in many existing algorithms and improve them significantly in terms of the total cost of the achieved solution. Moreover, GRO requires little computational cost while can accelerate convergence of the original algorithm and, thereby, even shortens the time required to obtain high quality solutions. The advantages of GRO are justified by embedding it in the TSA and comparing with six existing algorithms, i.e., the CARPET, VND, GLS, MA, and two versions of TSA, on five benchmark test sets.

The rest of this paper is organized as follows. First, the mathematical formulation of CARP is presented in Section II, followed by the introduction of GRO. Section III presents the repair-based tabu search (RTS) algorithm, which is obtained by embedding GRO in TSA. Section IV is dedicated to the empirical studies. Finally, the conclusion and discussion are presented in Section V.

II. Repair Operator for CARP

This section describes the proposed GRO. We start from describing the mathematical formulation of CARP. After that, motivation and detailed steps of GRO are presented.

A. Mathematical Formulation of CARP

Suppose a CARP represented by a graph $G = (V, E, A)$ is given. Let $V = \{v_1, \ldots, v_n\}$ and v_1 denotes the depot; each $(v_i, v_j) \in E \cup A$ is associated with the demand $d(v_i, v_j)$, serving cost $sc(v_i, v_j)$, and deadheading cost $dc(v_i, v_j)$. Furthermore, a set of vehicles with identical capacity Q is based at the depot.

As mentioned before, the vertex encoding, which is widely used in heuristic approaches to CARP, consists of an ordered list and a vector of zero–one variables. The ordered list is a permutation of vertices. More specifically, an ordered list consists of several routes, each of which starts and ends with v_1. Every element of the ordered list is associated with a zero–one variable y. y takes one if the edge between the vertex and its successor is served at this stage of the route. Fig. 1 shows the vertex encoding for a solution of a CARP. a, b, c, and o are vertices, where o denotes the depot. The first permutation is the ordered list, which can be divided into three single routes named route1, route2, and route3, respectively. Each route starts and ends at the depot o, and traverses the vertices in the order given by the ordered list, e.g., route1 leaves o, visits a and b consequently, and then returns o. The second vector is the associated zero–one variable vector. The solution corresponding to such a vertex encoding is as follows: Route1 traverses (o, a) deadheadingly and then serves (a, b) and (b, o); route2 traverses (o, b) deadheadingly and then serves (b, c) and (c, o); route3 traverses (o, c) deadheadingly and then serves (c, a) and (a, o).

An ordered list can be formally presented as $OL = \langle l_1, l_2, \ldots, l_K \rangle$, where $l_k \in V$, with $k = 1, 2, \ldots, K$. Suppose there are m routes in the ordered list, then the depot v_1 appears $(m + 1)$ times in the ordered list. Without loss of generality, assume these v_1's lie at the position $r_0, r_1, \ldots, r_m$, where $r_0 = 1$, $r_m = K$, and $r_i < r_j$, $\forall 1 \leqslant i < j \leqslant m$. Accordingly, the zero–one variable vector is denoted as $\langle y_1, y_2, \ldots, y_{K-1} \rangle$. y_k takes one if (l_k, l_{k+1}) is served at position k of the ordered list and takes zero if otherwise.

With the aforementioned definitions, the CARP can be formulated as follows:

$$\min \sum_{k=1}^{K-1} (sc(l_k, l_{k+1}) \times y_k + dc(l_k, l_{k+1}) \times (1 - y_k)) \quad (1)$$

$$s.t.: \sum_{k=1}^{K-1} y_k = N \quad (2)$$

$$(l_k, l_{k+1}) \in E_R \cup A_R \quad \forall y_k = 1 \quad (3)$$

$$(l_k, l_{k+1}) \neq (l_j, l_{j+1}) \quad \forall y_k = 1; \quad y_j = 1 \quad k \neq j \quad (4)$$

$$\sum_{k=r_i}^{r_{i+1}-1} d(l_k, l_{k+1}) \times y_k \leqslant Q, \quad i = 0, 1, 2, \ldots, m-1 \quad (5)$$

$$l_k \in V, \quad k = 1, 2, \ldots, K \quad (6)$$

$$(l_k, l_{k+1}) \in E \cup A, \quad k = 1, 2, \ldots, K-1 \quad (7)$$

$$y_k = 0 \text{ or } 1, \quad k = 1, 2, \ldots, K-1 \quad (8)$$

where N is the number of tasks. Constraints (2)–(4) indicate that each task is served once and only once. Constraint (5) bounds the total demand of each single route with the vehicle capacity. Constraints (6)–(8) define the domains of the variables.

B. Motivation

Substituting constraints (2)–(4) into the objective function (1) in the former section, we can easily modify function (1) to the following form:

$$\min \sum_{k=1}^{K-1} (dc(l_k, l_{k+1})) + \sum_{(i,j) \in E_R \cup A_R} (sc(i, j) - dc(i, j)). \quad (9)$$

From the modified objective function, it can be observed that the two components of vertex encoding play different roles in the solution. The ordered list determines the total cost of a solution, while the zero–one variables determine whether and how much the solution violates the capacity constraint. Since an edge can be traversed multiple times in CARP, one ordered

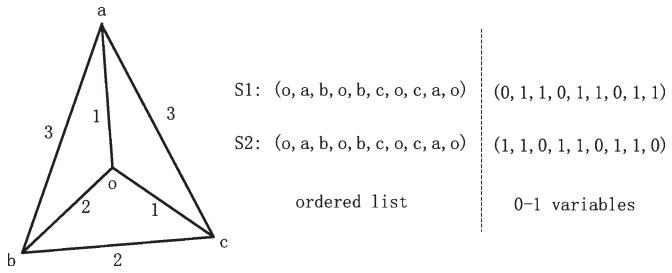


Fig. 2. Example of CARP.

list may correspond to many solutions with different values of the zero–one variables.

The optimal solution of CARP corresponds to both an optimal ordered list and a feasible assignment of zero–one variables on the ordered list. In the existing methods, various search operators have been employed, such as single insertion (SI), double insertion (DI), swap (employed in MA [12] and TSA [17]), and 2-opt (employed in MA [12]). Given the current solution, all these operators generate a new solution by modifying both the ordered list and the corresponding zero–one variables. However, the simultaneous optimization of the two components is not an easy task, and modifying them simultaneously may even lead to an ineffective search in the solution space. An example is given in Fig. 2 to demonstrate this problem. In Fig. 2, there are four vertices in the graph, denoted as a, b, c, and o, of which vertex o represents the depot. All the six edges, i.e., $\{(o,a),(o,b),(o,c),(a,b),(a,c),(b,c)\}$, are required to be served. Suppose that there are three vehicles available, with the same capacity of four. The number on each edge denotes its demand, e.g., $d(a,b)=3$. The right part in Fig. 2 shows the vertex representations of two different solutions S_1 and S_2. S_1 and S_2 share the same ordered list but are different on the second component. As a result, S_1 is infeasible with one unit capacity violation, i.e., $d(a,b)+d(b,o)=3+2=5>4$, whereas S_2 is the global optimum. Hence, given S_1, the optimal solution can be easily achieved by modifying the associated zero–one variables only, while modifying both components of the solution will discard the optimal ordered list and lead to an ineffective search.

The aforementioned example demonstrates that permuting only the zero–one variable of an infeasible solution might be more effective than modifying both components of it. In particular, those low cost infeasible solutions should be focused more during the search process, because a solution with low cost is more likely to contain the optimal ordered list (so that only the zero–one variables need to be readjusted). However, few search operators have been designed for this purpose. Instead, most existing approaches handle infeasible solutions using relatively standard methods that are available in the constrained optimization literature and tend to overlook the promising infeasible solutions. For example, the CARPET [9], VND [10], and TSA [17] simply define the objective function as a weighted sum of cost and constraint violation. By tuning the weights, the search process biases more toward the feasible solutions, and no effort is spent to examine the infeasible solutions. In [12] and [13], a genotype representation scheme is adopted. To obtain the final solution, a split operator, which does not generate infeasible solutions at all, needs to be applied. Thus, only feasible solutions are considered in this case, and all the infeasible solutions are discarded even without calculating the cost. In [16], Handa *et al.* also adopt the weighted sum of cost and constraint violation as the objective function. In addition, a repair operator is proposed to handle infeasible solutions. Given an infeasible solution, the repair operator first figures out the route that violates the capacity constraint the most in the solution. Then, a task is randomly chosen from those tasks served in this route, and it will be moved to another route that traverses the selected task by deadheading path. Otherwise, no change will be made. The repair process terminates when the aforementioned procedure has been repeated for a predefined number of times or the constraint violation has been reduced to zero. To the best of our knowledge, this is the only existing repair operator that is specifically designed in the context of CARP. Nevertheless, this repair operator always works on a part (a single route) of the solution rather than the whole solution itself. Hence, it searches in a rather *local* region around the infeasible solution, and it might not be able to fully exploit the useful information in the infeasible solution. Based on the consideration that the low cost infeasible solutions deserve to be examined more carefully, we propose our GRO.

C. GRO

As mentioned earlier, GRO is specifically designed to deal with the low cost infeasible solutions. Given such a solution, GRO preserves its ordered list and reassigns the zero–one variables to minimize the constraint violation. In other words, GRO seeks the optimal assignment of zero–one variables for a given ordered list. Such a repair process takes into account all routes involved in the solution, and thus, GRO can be viewed as a global operator. Suppose that we have an infeasible solution with m routes, reassigning the zero–one variables can be formulated as the following problem:

$$\min \sum_{i=1}^{m}\left(\max\left[\sum_{j=1}^{N} s_j \times x_{ij} - Q, 0\right]\right) \tag{10}$$

$$s.t.: \sum_{i\in\Omega(j)} x_{ij} = 1, \quad j=1,2,\ldots,N \tag{11}$$

$$x_{ij}=0 \text{ or } 1, \quad i=1,2,\ldots,m; \quad j=1,2,\ldots,N \tag{12}$$

where N is the total number of tasks, and s_j denotes the serving cost for task j. x_{ij} is set to one if task j is served in the route i and is set to zero if otherwise. $\Omega(j)$ is defined as

$$\Omega(j)=\{i|\text{task } j \text{ is traversed in route } i \text{ in } S\}. \tag{13}$$

Given the ordered list, constraints (11) and (12) guarantee that each task is served only once among the routes it is traversed.

Let $\{a_1,a_2,\ldots,a_N\}$ and $\{b_1,b_2,\ldots,b_m\}$ be sets of items and bins, respectively. The aforementioned problem can be viewed as a bin-packing problem, where the size of the item

```
   Input: a solution S_0
   Output: an archive A
1  Set current solution S = S_0, best solution S_b = S;
2  Set the archive A = ∅;
3  while stopping criterion is not met do
4      Set f(S') = ∞;
5      foreach S_N ∈ N(S) do
6          if S_N is non-tabu and f(S_N) < f(S') then
7              S' = S_N, f(S') = f(S_N);
8          end
9      end
10     if f(S') = 0 then A = A ∪ S';
11     if f(S') < f(S_b) then S_b = S', f(S_b) = f(S');
12     Update tabu list;
13     S = S';
14 end
15 if A = ∅ then A = {S_b};
16 return A;
```

Fig. 3. Tabu search process.

j is s_j, and all the bins share an identical capacity Q. GRO employs an insertion heuristic followed by a short-term tabu search to solve the bin-packing problem. The general idea of the insertion heuristic is straightforward. We sequentially pick an item out of the whole set and insert it into a bin, until all the items have been inserted. Such procedure can be described as follows.

Step 1) Initialize $x_{ij} = 0, \forall i, j$. Let $A = \{1, 2, \ldots, N\}$ and $cl(b_i) = 0, \forall i$. Here, $cl(b_i)$ is the current load of b_i. Then, repeat steps 2) to 4) until $A = \emptyset$.

Step 2) For each $j \in A$, identify the set $\bar{\Omega}(j)$ satisfying $\bar{\Omega}(j) = \{i \in \Omega(j) | cl(b_i) + s_j \leqslant Q\}$. Select the item corresponding to the smallest $|\bar{\Omega}(j)|$ as the one to be inserted. If multiple items share the smallest $|\bar{\Omega}(j)|$, the one with the largest s_j will be selected. Then, ties are broken by selecting the item with the smallest index (j). Selecting the items in this way guarantees that the item with the least choice of insertion without violating the constraints is chosen first.

Step 3) Identify the b_i with the smallest $cl(b_i)$. If more than one bin has the smallest $cl(b_i)$, the one with the smallest

$$\sum_{j \in A} I_{\Omega(j)}(i) \times s_j$$

is selected, where $I_{\Omega(j)}(i)$ is an indicator function. $I_{\Omega(j)}(i) = 1$ if $i \in \Omega(j)$ and zero if otherwise. The aforementioned equation indicates that the bin available for the least untreated items is considered first. After that, ties are broken by selecting the bin with the smallest index (i).

Step 4) Insert the selected item $a_{j'}$ in the chosen bin $b_{i'}$. Set $x_{i'j'} = 1$, remove j' from A, and update $cl(b_{i'})$ with $s_{j'}$.

After obtaining the initial solution with the insertion heuristic, a standard tabu search is employed to further improve it, as shown in Fig. 3.

In Fig. 3, S_0 is the solution obtained by the insertion heuristic, and $f(S)$ denotes the objective function (10). The neighborhood $N(S)$ is a set of neighbors generated by moving one item to another admissible bin. The tabu list and aspiration criterion employed here are conventional. That is, an item that just leaves a bin is not allowed to return to that bin within a certain number of iterations (i.e., the tabu tenure), unless the resultant solution is better than the best solution found so far. The tabu tenure is set to $F/2$, where F is the number of items having more than one admissible bins. The tabu search terminates after N iterations or consecutive $N/3$ iterations without improvement.

Based on a solution of the bin-packing problem, a new solution of CARP can be directly obtained by updating the zero–one variables according to x_{ij}'s. Since the ordered list remains unchanged when solving the bin-packing problem, the issue of cost is actually not considered at this stage. However, the tabu search process might generate different assignments of zero–one variables that all correspond to feasible solutions of the CARP, and it will be difficult to select them if the cost is not considered. Hence, we further define an archive in the tabu search process. As can be seen from line 10 in Fig. 3, at each iteration of the tabu search process, if the obtained best assignment of zero–one variables corresponds to a feasible solution to CARP, it will be included into the archive. After the termination of the tabu search process, all candidates in the archive will be fed into the following *further refinement* procedure to complete the repair process.

In the vertex encoding, two adjacent services of tasks are connected by the shortest path between them. When a new solution is obtained by changing the zero–one variables only, the path between two adjacent services might no longer be the shortest path. The new solution can be easily improved by a *further refinement* procedure, which modifies the ordered list of the solution without changing the zero–one variables so that any two adjacent services are connected with the shortest path between them. Hence, as the final step of our repair process, all the archived assignments of zero–one variables are transformed to the corresponding solutions of CARP; then, the ordered lists of these solutions are refined by updating the vertices between each pair of adjacent services with the shortest path. Finally, the solution with the lowest cost is chosen as the output of the repair process.

To summarize, the major steps of GRO are listed as follows.

1) Formulate the problem given by (10) as a bin-packing problem, and get a solution via the insertion heuristic.
2) Utilize a tabu search process to further improve the solution obtained in step 1), and get an archive of candidate assignments of zero–one variables.
3) Obtain new solutions of CARP based on the archived assignments of zero–one variables, and update these solutions with the further refinement procedure. The solution with the lowest cost is chosen as the output of GRO.

III. Embedding GRO in TSA

It can be observed that GRO is solely based on the vertex encoding. Hence, it can be embedded in any search algorithm that adopts the same encoding scheme. In this section, we demonstrate how to combine GRO with a recently proposed tabu search method. Belonging to the family of local search

techniques, tabu search has been widely used to solve various real-world problems [18], [19]. The TSA employed here was specifically proposed for CARP [17].

In TSA, the function to be optimized is defined as a weighted sum of the cost and capacity violation, i.e., $f(S) = c(S) + p \times cv(S)$, where $c(S)$ is the cost of S and $cv(S)$ is the capacity violation of S. The self-adaptive parameter p controls the tradeoff between the cost and feasibility. p is set to one first and is halved (doubled) if all the solutions are feasible (infeasible) for ten consecutive iterations. TSA employs three move operators, namely, SI, DI, and swap1. During the search process, TSA realizes different search biases by adjusting the frequencies of applying the three operators (F_{SI}, F_{DI}, and F_{swap}). Moreover, the Frederickson's heuristic [20] is also used to obtain further improvements. Since TSA is only used to demonstrate the efficacy of GRO and no modification is made to it throughout this work, we do not present full details of TSA in this paper. Interested readers are referred to the original publication [17].

In [17], the aforementioned TSA is named TSA version 1 (TSA1). Furthermore, a TSA version 2 (TSA2) has also been proposed. Generally speaking, TSA2 applies TSA1 to five different initial solutions simultaneously. After that, TSA1 is further applied to the best solution that has been achieved but with different parameter values. Experimental studies showed that TSA1 is much faster than three compared methods, namely, CARPET, MA, and TSA2, while still obtained acceptable solutions (better than CARPET and slightly worse than MA). The runtime of TSA2 is about five times more than TSA1, while it performs much better in terms of the quality of solutions.

The GRO is embedded in TSA with little effort. That is, every time an infeasible solution with the lowest cost so far is obtained, GRO is applied to it. We name such a simple combination of GRO and TSA1 as the RTS algorithm. Since both GRO and TSA1 are deterministic, RTS is a deterministic algorithm.

IV. Experimental Studies

To evaluate the efficacy of GRO, we experimentally compare RTS to a number of state-of-the-art approaches in this section.

A. Experimental Setup

The experiments were carried out on five benchmark test sets of CARP problems, referred to as the *gdb*, *val*, and *egl* sets, the set of Beullens *et al.*, and Brandão and Eglese's set. All these test sets have been studied in the literature. The *gdb* set was generated by DeArmon in [21] and consisted of 23 instances. The *val* set was generated by Benavent *et al.* in [22]. It contains 34 instances based on ten different graphs. Different instances based on each graph were generated by changing the capacity of the vehicles. The *egl* set was generated by Eglese based on data from a winter gritting application in Lancashire [23]–[25]. It consists of 24 instances based on two graphs, each with a distinct set of required edges and capacity constraints. The test set generated by Beullens *et al.* in [14] is based on the intercity road network in Flanders. It further contains four subsets, namely, the sets C, D, E, and F, each of which contains 25 instances. The instances of sets D and F share the same networks with the instances of sets C and E, respectively, but with larger capacity of vehicles. Generated by Brandão and Eglese in [17], the final test set consists of ten large instances defined on a road network with 255 vertices and 375 edges in Lancashire. Different instances in this set were created by changing the set of edges required for service and the capacity of the vehicles.

We considered six existing approaches in the comparative study. That is, the CARPET [9], VND [10], GLS [14], MA [12], TSA1, and TSA2 [17]. To facilitate the comparison between RTS and TSA1, RTS was implemented using the same parameters suggested for TSA1 in [17]. Since TSA2 applies TSA1 for six times, it might be inappropriate to directly compare RTS to TSA2. Instead, we further consider an extended version of RTS, represented by RTS* hereinafter. The RTS* was obtained by adjusting the stopping criterion of RTS, so that its runtime is increased by about four times. Except for the stopping criterion, RTS* adopts exactly the same parameters as RTS. In this way, we are able to conduct a more comprehensive comparison between RTS and the other approaches.

B. Experimental Results

Tables I–VIII present the experimental results of the compared approaches on all the five test sets. It should be noted that few existing algorithms have been tested on all the five benchmark test sets. On the *gdb* and *val* sets, the results of CARPET, VND, MA, TSA1, TSA2, RTS, and RTS* are available (Tables I and II). As shown in Table III, MA, TSA1, TSA2, RTS, and RTS* had been applied to the *egl* set. Comparison between GLS, TSA1, TSA2, RTS, and RTS* can be made based on the set of Beullens *et al.*; results for the four subsets are presented in Tables IV–VII. Finally, RTS and RTS* are only compared with TSA1 on the Brandão and Eglese's set (Table VIII), since no other algorithm has been applied to this set before. A brief description of the contents in each table is listed as follows.

1) The columns headed "$|\mathrm{V}|$," "$|\mathrm{R}|$," and "$|\mathrm{E}|$" indicate the number of vertices, required edges, and total edges, respectively. Since all edges are required to be served in the *gdb* set, the column $|\mathrm{R}|$ is omitted from Table I.
2) The column headed "LB" gives the lower bounds found so far for the problems, which are available in [14], [17], and [26]–[29]. Note that this column is absent in Table VIII because the lower bounds of the problems in the Brandão and Eglese's set are not available in the literature.
3) The columns headed "Cost" present the cost of the final solutions obtained by the corresponding algorithm. The columns headed "CPU(s)" provide the runtime (in CPU seconds) needed to obtain the solution.
4) For Tables I–VII, three additional rows are included at the bottom of the tables. The first row presents the average costs of solutions and runtimes calculated for each approach over all the instances in each set. The average values of lower bounds have also been calculated

TABLE I
RESULTS FOR *gdb* BENCHMARK TEST SET. "mean", "No.opt", AND "APD" STAND FOR AVERAGE COST AND RUNTIMES, NUMBER OF OPTIMAL SOLUTIONS, AND APD TO THE LOWER BOUNDS, RESPECTIVELY

Name	|V|	|E|	LB	CARPET		VND		MA		TSA1		TSA2		RTS		RTS*	
				Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)
1	12	22	316	316	1.7	316	0.0	316	0.0	316	0.0	316	0.0	**316**	0.0	**316**	0.0
2	12	26	339	339	2.8	339	0.0	339	0.2	345	0.1	339	0.1	**339**	0.0	**339**	0.0
3	12	22	275	275	0.0	275	0.0	275	0.0	275	0.0	275	0.0	**275**	0.1	**275**	0.1
4	11	19	287	287	0.1	287	0.0	287	0.0	287	0.0	287	0.0	**287**	0.0	**287**	0.0
5	13	26	377	377	3.0	377	0.2	377	0.1	377	0.0	377	0.1	**377**	0.0	**377**	0.0
6	12	22	298	298	0.5	298	0.0	298	0.1	298	0.0	298	0.0	**298**	0.0	**298**	0.0
7	12	22	325	325	0.0	325	0.0	325	0.1	325	0.0	325	0.0	**325**	0.0	**325**	0.0
8	27	46	348	352	33.0	350	5.2	350	18.6	352	1.8	348	1.1	**348**	2.0	**348**	9.4
9	27	51	303	317	29.2	315	8.6	303	3.5	307	0.7	303	18.3	**303**	1.3	**303**	1.3
10	12	25	275	275	0.8	275	0.0	275	0.0	275	0.0	275	0.0	**275**	0.0	**275**	0.0
11	22	45	395	395	1.2	395	0.0	395	0.6	395	0.1	395	0.1	**395**	0.2	**395**	0.2
12	13	23	458	458	11.2	458	1.6	458	4.9	462	0.1	458	0.6	**458**	1.7	**458**	1.7
13	10	28	536	544	1.3	544	1.1	536	3.7	544	0.1	540	3.4	544	1.0	**536**	1.0
14	7	21	100	100	0.3	100	0.0	100	0.0	100	0.0	100	0.1	**100**	0.0	**100**	0.0
15	7	21	58	58	0.0	58	0.0	58	0.0	58	0.0	58	0.0	**58**	0.0	**58**	0.0
16	8	28	127	127	0.9	127	0.2	127	0.0	129	0.1	127	0.1	**127**	0.0	**127**	0.0
17	8	28	91	91	0.0	91	0.0	91	0.0	91	0.0	91	0.0	**91**	0.0	**91**	0.0
18	9	36	164	164	0.1	164	0.0	164	0.1	164	0.0	164	0.0	**164**	0.3	**164**	0.3
19	8	11	55	55	0.1	55	0.0	55	0.0	55	0.0	55	0.0	**55**	0.0	**55**	0.0
20	11	22	121	121	5.2	121	0.1	121	0.2	123	0.1	121	0.1	**121**	0.0	**121**	0.0
21	11	33	156	156	0.6	156	0.1	156	0.1	156	0.0	156	0.0	**156**	0.1	**156**	0.1
22	11	44	200	200	1.8	200	0.8	200	1.7	200	0.0	200	0.1	**200**	0.0	**200**	0.0
23	11	55	233	235	18.6	235	2.8	233	25.6	235	0.5	235	15.6	**233**	0.1	**233**	0.1
mean			253.8	255.0	4.9	254.8	0.5	253.9	2.6	255.2	0.2	254.0	1.7	254.1	0.2	253.8	0.6
No.opt			23	19	-	19	-	22	-	15	-	21	-	22	-	23	-
APD			-	0.35	-	0.30	-	0.03	-	0.46	-	0.07	-	0.06	-	0.00	-

for reference. The second row summarizes the number of instances on which the approach has achieved the optimal solutions (i.e., reaches the lower bounds). The third row calculates for each approach the average percentage deviation (APD) to the lower bounds. Because lower bounds are not available for the instances of the Brandão and Eglese's test set, only the average costs and runtimes are provided for Table VIII.

5) In all the tables, results are highlighted in bold for the instances on which RTS or RTS* achieved the optimal solutions.

In our experiments, RTS and RTS* were coded with C language and run using an Intel(R) Xeon(R) E5335 2.00 GHz. The results of the six existing approaches were directly obtained from the original publications. Since the compared approaches were implemented on different computers, normalization has been carried out to make fair comparisons on the runtimes. That is, all the runtimes presented in this paper were obtained via dividing the runtimes in the original publications by some factors. To be specific, CARPET and VND were implemented on the Graphics Indigo2 (195 MHz); thereby, the runtimes presented in [9] and [10] have been divided by ten. The results of GLS in [14] were obtained using a Pentium II 500 MHz; thus, we divided the runtimes there by four. The MA was implemented on a Pentium III 1 GHz and the TSA on a Pentium Mobile 1.4 GHz. Hence, the runtimes of MA, TSA1, and TSA2 given in the corresponding papers have been divided by 2, 10/7, and 10/7, respectively.

The efficacy of GRO can be evaluated from two perspectives, i.e., the quality of solution and the computational time. The average cost, the number of optimal solutions achieved, and the APD are all examined to get a more comprehensive comparison on the quality of solution. From Tables I and II, it can be seen that MA, TSA2, RTS, and RTS* performed comparably on the *gdb* and *val* sets, while the results of CARPET, VND, and TSA1 are inferior. Among the former four methods, RTS requires the least runtime, and RTS* is more efficient than MA and TSA2.

On the *egl* set, RTS* significantly outperformed the other methods in terms of solution qualities. MA, TSA2, and RTS achieved comparable results, and TSA1 performed the worst. Furthermore, RTS* is less time consuming than MA and TSA2, not to mention RTS.

In [14] and [17], results on the four test subsets of Beullens *et al.* were reported in terms of the cost of deadheading only. Hence, we present the results of RTS and RTS* in the same form in Tables IV–VII. In general, GLS exhibited the best overall performance, followed by RTS*, TSA2, and RTS. However, we can further find that GLS outperformed the RTS* marginally, while the runtime of RTS* is always shorter than GLS. This is particularly obvious on sets C and E. RTS* appears to be superior to TSA2 on set F, and the two achieved similar performance on sets C, D, and E. On the other hand, RTS* requires about half of the runtime of TSA2 on all the four sets. Unsurprisingly, TSA1 did not perform as good as the others.

Finally, as shown in Table VIII, RTS not only achieved solutions of higher qualities than TSA1 but also was computationally more efficient. Since only the results of TSA1 are available in the literature, the results of RTS* are presented for reference only.

To summarize, RTS and RTS* are competitive with a number of state-of-the-art approaches for CARP. In particular, RTS is the most efficient one among all the compared approaches, and it also managed to achieve high quality solutions on many

TABLE II
RESULTS FOR *val* BENCHMARK TEST SET. "mean", "No.opt", AND "APD" STAND FOR AVERAGE COST AND RUNTIMES, NUMBER OF OPTIMAL SOLUTIONS, AND APD TO THE LOWER BOUNDS, RESPECTIVELY

Name	|V|	|E|	LB	CARPET		VND		MA		TSA1		TSA2		RTS		RTS*	
				Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)
1A	24	39	173	173	0.0	173	0.0	173	0.0	173	0.0	173	0.0	**173**	0.0	**173**	0.0
1B	24	39	173	173	5.0	173	0.4	173	4.0	173	0.1	173	0.6	**173**	0.5	**173**	0.5
1C	24	39	245	245	50.6	245	1.6	245	14.3	245	0.6	245	8.5	**245**	0.0	**245**	0.0
2A	24	34	227	227	0.1	227	0.0	227	0.0	227	0.0	227	0.0	**227**	0.0	**227**	0.0
2B	24	34	259	260	7.1	259	0.2	259	0.1	259	0.1	259	0.2	**259**	0.1	**259**	0.1
2C	24	34	457	494	17.2	457	3.4	457	10.9	457	1.0	457	5.5	**457**	0.3	**457**	0.3
3A	24	35	81	81	0.4	81	0.0	81	0.0	81	0.0	81	0.0	**81**	0.0	**81**	0.0
3B	24	35	87	87	1.5	87	0.0	87	0.0	87	0.0	87	0.0	**87**	0.0	**87**	0.0
3C	24	35	138	138	22.6	140	2.7	138	14.1	138	0.4	138	0.9	**138**	0.2	**138**	0.2
4A	41	69	400	400	15.4	400	0.1	400	0.4	400	0.1	400	0.3	**400**	0.0	**400**	0.0
4B	41	69	412	416	41.0	414	2.0	412	0.6	414	1.2	412	3.9	**412**	0.9	**412**	0.9
4C	41	69	428	453	38.0	428	3.9	428	9.6	444	1.2	428	26.6	**428**	4.6	**428**	4.6
4D	41	69	526	556	126.6	544	6.6	541	51.6	538	7.2	530	77.0	530	1.3	530	21.0
5A	34	65	423	423	2.1	423	0.1	423	0.9	423	0.2	423	0.2	**423**	0.4	**423**	0.4
5B	34	65	446	448	22.4	449	1.4	446	0.1	446	0.1	446	0.1	**446**	0.0	**446**	0.0
5C	34	65	473	476	28.9	474	2.4	474	50.5	474	0.8	474	7.4	474	1.0	474	21.0
5D	34	65	573	607	121.5	599	4.2	583	45.4	583	4.6	583	51.3	583	4.8	583	24.3
6A	31	50	223	223	2.11	223	0.0	223	0.1	223	0.1	223	1.1	**223**	0.0	**223**	0.0
6B	31	50	233	241	14.6	233	1.2	233	33.7	233	1.8	233	8.9	**233**	0.5	**233**	0.5
6C	31	50	317	329	46.2	325	4.2	317	26.1	323	2.2	317	16.0	**317**	0.0	**317**	0.0
7A	40	66	279	279	3.6	279	0.0	279	0.9	283	0.6	279	0.7	**279**	1.2	**279**	1.2
7B	40	66	283	283	0.0	283	0.0	283	0.2	283	0.1	283	0.4	**283**	0.9	**283**	0.9
7C	40	66	334	343	65.8	335	4.5	334	50.6	335	2.8	334	25.9	**334**	3.1	**334**	3.1
8A	30	63	386	386	2.1	386	0.2	386	0.3	386	0.4	386	0.2	**386**	0.1	**386**	0.1
8B	30	63	395	401	44.2	403	1.1	395	5.0	407	0.7	395	1.3	**395**	0.0	**395**	0.0
8C	30	63	518	533	79.9	543	5.6	527	35.7	545	1.3	529	39.0	545	1.5	524	20.0
9A	50	92	323	323	15.5	323	1.4	323	9.1	323	0.5	323	0.0	**323**	1.5	**323**	1.5
9B	50	92	326	329	32.5	326	2.6	326	14.7	326	0.9	326	0.4	**326**	2.5	**326**	2.5
9C	50	92	332	332	30.6	336	2.8	332	35.6	332	0.5	332	0.3	**332**	4.0	**332**	4.0
9D	50	92	385	409	191.5	399	6.2	391	105.6	404	5.1	391	42.3	391	9.1	391	54.7
10A	50	97	428	428	3.0	428	0.2	428	12.7	430	2.5	428	2.2	**428**	3.6	**428**	3.6
10B	50	97	436	436	10.0	436	0.8	436	2.3	438	2.5	436	1.3	**436**	0.4	**436**	0.4
10C	50	97	446	451	50.7	446	3.3	446	8.7	447	3.2	446	5.3	**446**	5.0	**446**	5.0
10D	50	97	525	544	84.7	538	7.7	530	107.5	534	7.6	530	152.7	534	12.4	534	61.2
mean			343.2	350.8	34.6	347.5	2.1	345.2	21.6	347.5	1.5	344.9	14.1	345.5	1.8	344.9	6.8
No.opt				17	-	21	-	28	-	19	-	28	-	28	-	28	-
APD			-	1.60	-	0.82	-	0.27	-	0.77	-	0.22	-	0.33	-	0.21	-

problems. Although RTS* is computationally more expensive than RTS, it provided significantly better solutions. Furthermore, RTS* is still much less time consuming than those compared methods that perform similarly to RTS* in terms of quality of solutions. Hence, RTS* can be said to give a good tradeoff between quality and time.

Since we are proposing a repair operator (i.e., the GRO) rather than an algorithm (i.e., the RTS) in this paper, it is worthwhile to study the role that the GRO plays in RTS. This can be done by comparing RTS and RTS* to TSA1 and TSA2, respectively. From the perspective of solution qualities, RTS outperformed TSA on 120 problem instances among a total of 191 problem instances investigated in this paper. RTS* outperformed TSA2 on 42 problem instances, while was inferior on eight only. The statistics presented in the previous tables also evidence the superiority of RTS and RTS*. Hence, the incorporation of GRO undoubtedly improves TSA1's performance.

Given the improvement brought by GRO, we further investigate the computational time required by GRO during the optimization process. For this purpose, we recorded the total runtime of RTS* and the time occupied by GRO in it. The average runtime on each test set and all the sets are presented in Table IX. We can see that GRO accounted for no more than 4% runtime of RTS*. Hence, it is reasonable to expect that the runtimes of RTS are slightly longer than that of TSA1. Nevertheless, as shown in Tables I–VIII, RTS is actually much less time consuming than TSA1. The reason is that the RTS converged faster than the TSA1 and thereby stopped earlier. Moreover, we also noticed that the difference between RTS and RTS* is marginal on quite a lot of problem instances. For example, the APDs of the two algorithms are very close to each other for the *gdb* set, *val* set, and set E of the set of Beullens et als. Such an observation further demonstrates the fast convergence of RTS.

C. Further Analysis

Since RTS* was outperformed by the compared methods on some instances, we attempt to further analyze when RTS* is preferable. The general idea is to find the correlation (if any) between the performance of RTS* and some measurable characteristics of the problem instances. Since there is no work regarding this issue so far, we define the following metric, namely, proportion of free tasks (*PFT*), for our analysis. It is defined as follows:

$$PFT = \frac{\sum_{(i,j)\in E_R \cup A_R} I_{(0,AR]} d(i,j)}{N} \tag{14}$$

TABLE III
RESULTS FOR *egl* BENCHMARK TEST SET. "mean", "No.opt", AND "APD" STAND FOR AVERAGE COST AND RUNTIMES, NUMBER OF OPTIMAL SOLUTIONS, AND APD TO THE LOWER BOUNDS, RESPECTIVELY

Name	|V|	|R|	|E|	LB	MA		TSA1		TSA2		RTS		RTS*	
					Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)
E1-A	77	51	98	3548	3548	37.1	3548	1.5	3548	15.5	**3548**	0.0	**3548**	0.0
E1-B	77	51	98	4498	4498	34.7	4533	3.4	4533	19.6	4525	2.5	**4498**	8.8
E1-C	77	51	98	5566	5595	35.6	5659	3.6	5595	16.9	5595	2.8	5595	13.5
E2-A	77	72	98	5018	5018	76.3	5018	5.4	5018	44.4	**5018**	0.0	**5018**	0.0
E2-B	77	72	98	6305	6340	76.7	6385	8.1	6343	46.7	6338	6.6	6317	29.2
E2-C	77	72	98	8243	8415	64.8	8400	8.4	8347	55.1	8356	6.8	8335	32.6
E3-A	77	87	98	5898	5898	121.0	6044	12.5	5902	54.1	5898	0.9	5898	0.9
E3-B	77	87	98	7704	7822	127.7	7916	12.5	7816	79.4	7799	10.9	7787	47.9
E3-C	77	87	98	10163	10433	103.2	10309	16.2	10309	94.0	10326	10.9	10305	52.8
E4-A	77	98	98	6408	6461	145.9	6476	9.8	6473	94.9	6464	11.7	6461	56.8
E4-B	77	98	98	8884	9021	156.4	9134	18.8	9063	117.3	9026	12.7	9026	61.5
E4-C	77	98	98	11427	11779	126.2	11627	22.3	11627	132.0	11598	16.9	11598	70.1
S1-A	140	75	190	5018	5018	104.3	5171	7.2	5072	46.6	**5018**	1.7	**5018**	1.7
S1-B	140	75	190	6384	6435	104.4	6388	9.2	6388	56.6	6488	6.8	6394	32.5
S1-C	140	75	190	8493	8518	82.8	8739	4.8	8535	55.4	8518	7.4	8518	35.4
S2-A	140	147	190	9824	9995	437.2	10190	49.1	10038	276.6	10087	37.8	9970	212.1
S2-B	140	147	190	12968	13174	380.3	13284	54.7	13178	313.8	13345	42.8	13345	207.7
S2-C	140	147	190	16353	16795	373.5	16709	37.5	16505	361.1	16600	47.2	16600	227.8
S3-A	140	159	190	10143	10296	535.3	10508	55.5	10451	387.9	10284	46.4	10284	221.5
S3-B	140	159	190	13616	14053	532.0	13981	58.9	13981	399.4	13979	50.5	13857	242.6
S3-C	140	159	190	17100	17297	437.2	17346	69.4	17346	417.5	17497	53.9	17316	264.4
S4-A	140	190	190	12143	12442	768.8	12546	90.9	12462	487.8	12388	74.5	12348	355.0
S4-B	140	190	190	16093	16531	715.1	16695	99.0	16490	668.2	16612	91.3	16442	406.9
S4-C	140	190	190	20375	20832	747.5	20981	101.4	20733	654.2	21156	91.2	20821	437.2
mean				9673.8	9842.3	263.5	9899.5	31.7	9823.0	204.0	9852.6	26.4	9804.1	125.8
No.opt					5	-	2	-	2	-	4	-	5	-
APD				-	1.38	-	2.08	-	1.30	-	1.43	-	1.04	-

TABLE IV
RESULTS FOR BEULLENS *et al.* SET C. "mean", "No.opt", AND "APD" STAND FOR AVERAGE COST AND RUNTIMES, NUMBER OF OPTIMAL SOLUTIONS, AND APD TO THE LOWER BOUNDS, RESPECTIVELY

Name	|V|	|R|	|E|	LB	GLS		TSA1		TSA2		RTS		RTS*	
					Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)
C1	69	79	98	1590	1660	81.4	1700	8.9	1660	89.0	1660	7.7	1660	33.1
C2	48	53	66	1095	1095	2.0	1165	1.1	1095	18.6	**1095**	1.2	**1095**	1.2
C3	46	51	64	875	925	43.2	935	2.7	925	13.6	925	2.3	925	11.3
C4	60	72	84	1285	1340	70.9	1515	7.7	1340	45.8	1340	5.8	1340	28.6
C5	56	65	79	2410	2475	58.6	2630	4.8	2470	33.5	2540	4.8	2470	23.8
C6	38	51	55	855	895	40.1	910	1.9	895	6.5	895	2.4	895	11.1
C7	54	52	70	1735	1795	41.7	1795	3.8	1795	20.9	1795	3.0	1795	15.2
C8	66	63	88	1640	1730	57.5	1740	5.0	1730	31.1	1730	3.9	1730	18.8
C9	76	97	117	1775	1825	111.3	1880	17.2	1830	172.1	1830	11.9	1830	59.1
C10	60	55	82	2190	2290	45.3	2370	3.9	2270	21.4	2270	3.0	2270	15.6
C11	83	94	118	1725	1815	105.9	1940	16.0	1815	146.6	1805	11.0	1805	54.3
C12	62	72	88	1510	1610	71.3	1760	7.6	1610	32.3	1675	6.0	1610	30.6
C13	40	52	60	1050	1110	43.4	1115	3.1	1110	16.7	1110	2.8	1110	13.1
C14	58	57	79	1620	1680	49.7	1710	4.5	1680	38.2	1680	3.2	1680	15.7
C15	97	107	140	1765	1860	138.1	1910	21.6	1865	234.7	1870	15.6	1860	82.8
C16	32	32	42	580	585	30.7	585	0.6	585	3.6	585	0.7	585	3.5
C17	43	42	56	1590	1610	34.4	1630	2.2	1610	9.8	1610	1.84	1610	8.7
C18	93	121	133	2315	2410	141.4	2460	27.2	2415	364.6	2430	22.0	2425	107.2
C19	62	61	84	1345	1395	52.6	1425	4.9	1400	53.3	1395	3.5	1395	17.7
C20	45	53	64	665	665	0.3	795	0.7	665	1.8	**665**	0.4	**665**	0.4
C21	60	76	84	1705	1725	81.7	1725	2.6	1725	22.4	1725	4.6	1725	33.2
C22	56	43	76	1070	1070	0.7	1070	0.1	1070	0.5	**1070**	0.6	**1070**	0.6
C23	78	92	109	1620	1690	95.4	1775	13.2	1700	69.7	1700	10.6	1700	51.5
C24	77	84	115	1330	1360	77.9	1405	4.2	1360	55.2	1385	4.8	1360	41.3
C25	37	38	50	905	905	0.1	935	1.3	905	0.5	**905**	0.1	**905**	0.1
mean				1449.8	1500.8	59.0	1555.2	6.7	1501.0	60.1	1507.6	5.3	1500.6	27.1
No.opt					4	-	1	-	4	-	4	-	4	-
APD				-	3.30	-	7.25	-	3.32	-	3.69	-	3.29	-

where

$$AR = Q - \frac{\sum_{(i,j)\in E_R \cup A_R} d(i,j)}{\left\lceil \sum_{(i,j)\in E_R \cup A_R} d(i,j)/Q \right\rceil} \tag{15}$$

$$I_{(0,AR]} d(i,j) = \begin{cases} 1, & \text{if } d(i,j) \leqslant AR \\ 0, & \text{otherwise.} \end{cases} \tag{16}$$

TABLE V
RESULTS FOR BEULLENS *et al.* SET D. "mean", "No.opt", AND "APD" STAND FOR AVERAGE COST AND RUNTIMES, NUMBER OF OPTIMAL SOLUTIONS, AND APD TO THE LOWER BOUNDS, RESPECTIVELY

Name	|V|	|R|	|E|	LB	GLS		TSA1		TSA2		RTS		RTS*	
					Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)
D1	69	79	98	725	725	3.0	865	2.0	740	43.9	740	3.1	740	28.7
D2	48	53	66	480	480	0.3	480	1.1	480	0.6	**480**	0.3	**480**	0.3
D3	46	51	64	415	415	0.1	415	0.1	415	0.1	**415**	0.0	**415**	0.0
D4	60	72	84	615	615	0.1	630	3.1	615	1.2	**615**	0.1	**615**	0.1
D5	56	65	79	1040	1040	0.4	1080	1.7	1040	2.0	**1040**	1.1	**1040**	1.1
D6	38	51	55	485	485	0.1	525	0.5	485	5.5	**485**	0.7	**485**	0.7
D7	54	52	70	735	835	34.7	915	2.4	835	16.0	835	2.5	835	11.8
D8	66	63	88	615	685	48.8	685	3.2	685	34.4	685	3.3	685	16.5
D9	76	97	117	680	680	0.3	810	4.3	680	21.1	**680**	3.0	**680**	3.0
D10	60	55	82	900	910	37.3	910	0.9	910	11.6	910	1.4	910	10.1
D11	83	94	118	920	930	92.1	990	5.6	960	78.4	1000	3.2	930	41.3
D12	62	72	88	680	680	0.1	735	1.8	680	18.2	730	5.1	**680**	8.6
D13	40	52	60	690	690	0.2	695	0.8	695	9.7	**690**	0.3	**690**	0.3
D14	58	57	79	920	930	45.1	950	0.7	940	14.8	930	0.6	930	9.9
D15	97	107	140	910	910	92.2	1100	16.5	950	71.6	920	14.1	920	72.8
D16	32	32	42	170	170	0.0	175	0.1	170	0.3	**170**	0.0	**170**	0.0
D17	43	42	56	675	675	0.1	675	0.0	675	0.0	**675**	0.2	**675**	0.2
D18	93	121	133	930	930	0.6	1075	9.7	930	133.4	950	9.2	950	101.1
D19	62	61	84	650	680	38.7	690	1.9	690	14.1	690	0.6	680	13.1
D20	45	53	64	415	415	0.0	420	1.8	415	0.8	**415**	0.0	**415**	0.0
D21	60	76	84	695	805	63.0	865	1.6	825	42.6	815	6.2	810	27.0
D22	56	43	76	690	690	0.1	690	0.0	690	0.0	**690**	0.1	**690**	0.1
D23	78	92	109	715	735	83.8	780	3.7	735	77.8	755	3.8	735	45.0
D24	77	84	115	620	670	62.0	670	7.2	670	73.6	765	1.7	670	35.0
D25	37	38	50	410	410	0.0	510	0.1	410	0.4	**410**	0.0	**410**	0.0
mean				671.2	687.6	24.1	733.4	2.8	693.2	26.9	699.6	2.4	689.6	17.3
No.opt					15	-	4	-	13	-	12	-	13	-
APD				-	2.38	-	8.90	-	3.02	-	4.04	-	2.62	-

TABLE VI
RESULTS FOR BEULLENS *et al.* SET E. "mean", "No.opt", AND "APD" STAND FOR AVERAGE COST AND RUNTIMES, NUMBER OF OPTIMAL SOLUTIONS, AND APD TO THE LOWER BOUNDS, RESPECTIVELY

Name	|V|	|R|	|E|	LB	GLS		TSA1		TSA2		RTS		RTS*	
					Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)
E1	73	85	105	1855	1940	85.5	2160	9.4	1935	142.2	1935	9.1	1935	45.4
E2	58	58	81	1580	1610	47.2	1700	3.9	1610	37.2	1610	3.4	1610	16.3
E3	46	47	61	750	750	0.2	750	0.1	750	5.4	**750**	1.2	**750**	1.2
E4	70	77	99	1580	1610	82.1	1760	7.8	1615	92.6	1610	7.1	1610	36.6
E5	68	61	94	2130	2170	53.5	2235	6.0	2160	57.3	2185	3.6	2185	18.0
E6	49	43	66	670	670	0.0	670	0.0	670	0.5	**670**	0.4	**670**	0.4
E7	73	50	94	1780	1900	40.4	1900	0.6	1900	27.4	1900	0.5	1900	12.6
E8	74	59	98	2080	2150	55.4	2180	4.6	2155	58.6	2150	3.6	2150	17.7
E9	93	103	141	2160	2250	110.1	2440	17.0	2300	180.1	2325	15.0	2285	78.7
E10	56	49	76	1690	1690	0.1	1690	0.9	1690	2.6	**1690**	2.0	**1690**	2.0
E11	80	94	113	1810	1850	105.1	1980	15.7	1855	122.0	1850	12.7	1850	56.7
E12	74	67	103	1580	1710	66.2	1735	5.0	1730	62.8	1735	5.5	1715	25.9
E13	49	52	73	1300	1325	44.6	1415	0.7	1325	27.3	1325	2.6	1325	12.8
E14	53	55	72	1780	1810	47.6	1840	2.7	1810	31.8	1810	3.0	1810	14.7
E15	85	107	126	1555	1610	125.9	1645	20.4	1610	212.3	1625	12.0	1610	81.2
E16	60	54	80	1785	1825	49.8	1870	3.9	1825	49.5	1825	2.8	1825	13.9
E17	38	36	50	1290	1290	1.6	1325	1.1	1290	0.0	**1290**	0.7	**1290**	0.7
E18	78	88	110	1600	1610	90.8	1660	3.5	1610	86.2	1610	8.2	1610	49.9
E19	77	66	103	1400	1435	52.9	1475	5.5	1435	71.1	1435	4.8	1435	22.9
E20	56	63	80	950	990	58.1	1020	5.1	990	55.5	990	3.8	990	18.5
E21	57	72	82	1700	1705	73.4	1790	2.7	1705	54.9	1705	5.4	1705	26.8
E22	54	44	73	1155	1185	32.3	1215	2.0	1185	21.5	1185	1.5	1185	7.6
E23	93	89	130	1395	1435	98.7	1530	12.5	1445	132.7	1440	9.9	1430	49.4
E24	97	86	142	1695	1785	90.5	1850	9.4	1785	78.8	1785	2.8	1785	41.8
E25	26	28	35	655	655	0.0	655	0.1	655	0.1	**655**	0.1	**655**	0.1
mean				1517.0	1558.2	56.5	1619.6	5.6	1561.6	64.4	1563.6	4.9	1560.2	26.1
No.opt					5	-	4	-	5	-	5	-	5	-
APD				-	2.51	-	6.21	-	2.69	-	2.78	-	2.59	-

In (15), $\sum_{(i,j)\in E_R\cup A_R} d(i,j)$ is the overall demand of an instance. Correspondingly, $\lceil \sum_{(i,j)\in E_R\cup A_R} d(i,j)/Q \rceil$ is the minimal number of routes required for the instance, because the load of a single route must not exceed Q. Hence, the second term of (15) is the average load of each route, and thereby, AR is the average residual. A task is defined as a *free task* if its

TABLE VII
RESULTS FOR BEULLENS *et al.* SET F. "mean", "No.opt", AND "APD" STAND FOR AVERAGE COST AND RUNTIMES, NUMBER OF OPTIMAL SOLUTIONS, AND APD TO THE LOWER BOUNDS, RESPECTIVELY

Name	|V|	|R|	|E|	LB	GLS		TSA1		TSA2		RTS		RTS*	
					Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)
F1	73	85	105	1065	1065	0.9	1090	8.1	1085	62.0	1155	3.3	1065	30.8
F2	58	58	81	920	920	0.6	940	1.8	920	0.2	**920**	0.8	**920**	0.8
F3	46	47	61	400	400	0.0	480	1.3	400	0.9	**400**	0.4	**400**	0.4
F4	70	77	99	930	940	62.0	970	1.9	960	44.6	950	5.8	950	28.9
F5	68	61	94	1180	1180	0.1	1185	1.3	1180	12.0	**1180**	0.5	**1180**	0.5
F6	49	43	66	490	490	0.0	540	0.3	490	0.0	**490**	0.4	**490**	0.4
F7	73	50	94	1080	1080	0.0	1110	2.5	1080	1.2	**1080**	0.0	**1080**	0.0
F8	74	59	98	1135	1145	46.4	1155	0.8	1145	25.0	1145	1.0	1145	11.6
F9	93	103	141	1145	1145	0.9	1520	5.2	1170	102.1	1215	11.3	**1145**	29.6
F10	56	49	76	1010	1010	0.0	1010	0.1	1010	0.1	**1010**	0.0	**1010**	0.0
F11	80	94	113	1015	1015	1.1	1100	6.0	1015	88.1	**1015**	7.6	**1015**	7.6
F12	74	67	103	900	910	57.2	1000	3.4	910	33.5	910	3.0	910	22.5
F13	49	52	73	835	835	0.1	855	0.6	835	0.1	**835**	0.1	**835**	0.1
F14	53	55	72	1025	1025	4.7	1085	1.8	1035	20.8	1125	2.0	1035	12.8
F15	85	107	126	945	945	0.3	1315	15.5	990	101.8	975	6.6	**945**	69.3
F16	60	54	80	775	775	0.0	945	1.3	775	2.4	**775**	0.4	**775**	0.4
F17	38	36	50	605	605	0.0	660	0.2	630	3.3	**605**	0.1	**605**	0.1
F18	78	88	110	835	850	68.6	945	9.7	850	65.0	850	4.6	850	38.0
F19	77	66	103	685	725	39.5	740	1.1	740	24.4	725	2.0	725	19.3
F20	56	63	80	610	610	0.3	610	0.2	610	6.9	**610**	0.6	**610**	0.6
F21	57	72	82	905	905	1.0	940	1.5	905	31.9	**905**	1.0	**905**	1.0
F22	54	44	73	790	790	0.1	790	0.1	790	0.3	**790**	0.1	**790**	0.1
F23	93	89	130	705	725	79.9	895	3.2	730	56.1	775	7.8	725	41.8
F24	97	86	142	975	975	6.7	1040	7.2	1010	55.4	1005	7.4	1005	37.6
F25	26	28	35	430	430	0.0	430	0.0	430	0.1	**430**	0.0	**430**	0.0
mean				855.6	859.8	14.8	934.0	3.0	867.8	29.5	875	2.7	861.8	14.2
No.opt					19	-	4	-	13	-	14	-	16	-
APD				-	0.54	-	8.71	-	1.44	-	2.09	-	0.75	-

TABLE VIII
RESULTS FOR BRANDÃO AND EGLESE'S BENCHMARK TEST SET. "mean" STANDS FOR AVERAGE COST AND RUNTIMES

Name	|V|	|R|	|E|	TSA1		RTS		RTS*	
				Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)
G1-A	255	347	375	1049708	552.6	1042411	348.1	1025765	1778.4
G1-B	255	347	375	1140692	607.1	1137402	409.3	1135873	1905.2
G1-C	255	347	375	1282270	643.4	1278700	401.7	1271894	1903.5
G1-D	255	347	375	1420126	595.6	1417963	444.8	1402433	2230.6
G1-E	255	347	375	1583133	470.5	1573141	534.1	1558548	2279.9
G2-A	255	375	375	1129229	1018.9	1126112	510.7	1125602	2225.5
G2-B	255	375	375	1255907	785.6	1245141	454.4	1242542	2242.6
G2-C	255	375	375	1418145	594.3	1416282	473.0	1401583	2530.4
G2-D	255	375	375	1516103	1263.7	1516072	524.1	1516072	2336.1
G2-E	255	375	375	1701681	615.9	1684472	525.6	1668348	2492.1
mean				1349699.1	714.8	1343769.6	462.6	1334866.0	2192.4

TABLE IX
AVERAGE RUNTIMES OF RTS* AND GRO (IN CPU SECONDS)

Name	Average Runtime of RTS*	Average Runtime of GRO	Prop of GRO
gdb	0.6188	0.0095	1.5317%
val	6.8253	0.0092	0.1349%
egl	125.7910	2.9417	2.3386%
Beullens et al.'s Set C	27.1307	0.6334	2.3348%
Beullens et al.'s Set D	17.3051	0.0295	0.1704%
Beullens et al.'s Set E	26.0732	0.2419	0.9278%
Beullens et al.'s Set F	14.1632	0.0155	0.1093%
Brandão&Eglese	462.5752	18.1386	3.9212%
Overall	52.3970	1.4425	2.7531%

demand is no more than the AR. Based on such definition, the numerator of PFT stands for the number of free tasks in an instance. A smaller PFT implies a tighter capacity constraint.

Intuitively speaking, the tighter the capacity constraints, the harder the GRO can successfully repair an infeasible solution. If the constraints are so tight that the solutions obtained by GRO are not even close to the feasible region (i.e., the violations are still large after the repair process), GRO is unlikely to benefit the search process much. Hence, we expect GRO to be more preferable for problem instances whose capacity

constraints are not too tight, i.e., with large PFTs. In order to verify this hypothesis, we calculated the correlation coefficient between the solution quality and PFT based on the total 191 problem instances involved in this paper. First, the costs of the solutions are divided by the lower bounds for normalization. The correlation coefficient between the normalized results and PFT is -0.2463 for RTS and -0.2306 for RTS*. This clearly implies negative correlation between PFT and the performance of RTS, and thereby, our hypothesis is validated.

V. Conclusion and Discussion

In this paper, we proposed a search operator called GRO for CARP based on the vertex encoding. GRO is motivated by the observation that, in the vertex encoding, an ordered list may correspond to many solutions with different zero–one variables, and the traditional move operators may not be able to focus on the low cost ordered lists and lead to an ineffective search. GRO tries to minimize the capacity violation of a low cost infeasible solution while retaining its ordered list, which is formulated as a bin-packing problem and is NP-hard [30]. GRO employs an insertion heuristic and a tabu search to find near-optimal solutions for the bin-packing problem. To verify the efficacy of GRO, we embedded it in TSA1 and tested the resultant RTS algorithm on five previously studied benchmark test sets, which contain 191 CARP instances in total. Experimental results showed that GRO enhanced TSA1 significantly both in terms of solution quality and computational time, and the RTS* algorithm provided very competitive results in comparison to a number of state-of-the-art approaches for CARP. Furthermore, GRO is independent of the method generating the infeasible solutions and can be easily embedded in any method utilizing the vertex encoding (e.g., the most straightforward way is to call the GRO whenever an infeasible solution is reached.). Therefore, we believe that GRO would be an effective supplementary operator to the existing algorithms.

Two parameters need to be predefined for GRO: the tabu tenure and the number of iterations. In this paper, they are arbitrarily set to $F/2$ and N. Although these values are not meant to be optimal, experimental study showed that GRO is not very sensitive to the parameters. Hence, $F/2$ and N can be used as a rule of thumb.

As mentioned in Section II-B, few repair operators have been specifically designed for CARP. However, in the broader context of heuristic search, lots of repair operators have been proposed for various optimization problems [31]–[38]. Moreover, it is usually the case that a repair operator for one combinatorial optimization problem may not be directly applied to another problem, and some modifications are required. Hence, at the first sight, GRO merely looks like a new repair operator whose usage is restricted within a specific domain. However, the underlying mechanism of GRO is essentially different from many existing repair operators. When applied to an infeasible solution, GRO does not search in the original solution space (i.e., the space represented by the vertex encoding) for the alternative feasible solution. Instead, the search is carried out in the space represented by the zero–one variables. That means that GRO first switches the search space, then conducts the search, and finally transforms the obtained solution back to the original solution space. In consequence, after the repair process, we are likely to obtain a solution that is "distant" from the original solution in the solution space. Therefore, unlike many existing repair operators for combinatorial optimization (e.g., [34], [35], and [37]), which mainly conduct local searches in the original solution space, GRO actually implements a relatively global search. Such a characteristic, as well as the promising performance exhibited by GRO, may hopefully shed some light on the design of repair operators for other combinatorial optimization problems.

Last but not least, the promising results obtained via GRO imply that the assignment of tasks in different routes (i.e., zero–one variables) might have been overlooked in previous studies. Hence, the following question arises: Can an encoding scheme which focuses more on the assignment of tasks be more effective than the vertex encoding? This issue deserves in-depth research in the future.

Acknowledgment

The authors would like to thank Dr. J. Brandão and Prof. R. Eglese for the kind sharing of the test sets used in their work and the anonymous reviewers for the helpful comments and criticisms.

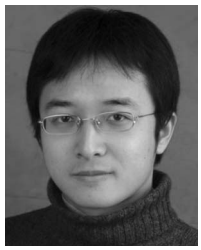
References

[1] M. Dror, Ed., *Arc Routing: Theory, Solutions and Applications.* Boston, MA: Kluwer, 2000.
[2] B. L. Golden and R. T. Wong, "Capacitated arc routing problems," *Networks*, vol. 11, no. 3, pp. 305–315, 1981.
[3] B. L. Golden, J. S. DeArmon, and E. K. Baker, "Computational experiments with algorithms for a class of routing problems," *Comput. Oper. Res.*, vol. 10, no. 1, pp. 47–59, 1983.
[4] G. Ulusoy, "The fleet size and mix problem for capacitated arc routing," *Eur. J. Oper. Res.*, vol. 22, no. 3, pp. 329–337, Dec. 1985.
[5] W. L. Pearn, "Approximate solutions for the capacitated arc routing problem," *Comput. Oper. Res.*, vol. 16, no. 6, pp. 589–600, 1989.
[6] W. L. Pearn, "Augment–insert algorithms for the capacitated arc routing problem," *Comput. Oper. Res.*, vol. 18, no. 2, pp. 189–198, Feb. 1991.
[7] M. C. Mourao and L. Amado, "Heuristic method for a mixed capacitated arc routing problem: A refuse collection application," *Eur. J. Oper. Res.*, vol. 160, no. 1, pp. 139–153, Jan. 2005.
[8] S. K. Amponsah and S. Salhi, "The investigation of a class of capacitated arc routing problems: The collection of garbage in developing countries," *Waste Manage.*, vol. 24, no. 7, pp. 711–721, 2004.
[9] A. Hertz, G. Laporte, and M. Mittaz, "A tabu search heuristic for the capacitated arc routing problem," *Oper. Res.*, vol. 48, no. 1, pp. 129–135, Jan./Feb. 2000.
[10] A. Hertz and M. Mittaz, "A variable neighborhood descent algorithm for the undirected capacitated arc routing problem," *Transp. Sci.*, vol. 35, no. 4, pp. 425–434, Nov. 2001.
[11] P. Greistorfer, "A tabu scatter search metaheuristic for the arc routing problem," *Comput. Ind. Eng.*, vol. 44, no. 2, pp. 249–266, Feb. 2003.
[12] P. Lacomme, C. Prins, and W. Ramdane-Cherif, "Competitive memetic algorithms for arc routing problem," *Ann. Oper. Res.*, vol. 131, no. 1–4, pp. 159–185, Oct. 2004.
[13] P. Lacomme, C. Prins, and W. Ramdane-Cherif, "A genetic algorithm for the capacitated arc routing problem and its extensions," in *Proc. EvoWorkshops Appl. Evol. Comput.*, Como, Italy, 2001, pp. 473–483.
[14] P. Beullens, L. Muyldermans, D. Cattrysse, and D. V. Oudheusden, "A guided local search heuristic for the capacitated arc routing problem," *Eur. J. Oper. Res.*, vol. 147, no. 3, pp. 629–643, Jun. 2003.
[15] H. Handa, D. Lin, L. Chapman, and X. Yao, "Robust solution of salting route optimization using evolutionary algorithms," in *Proc. IEEE Congr. Evol. Comput.*, Vancouver, BC, Canada, 2006, pp. 3098–3105.

[16] H. Handa, L. Chapman, and X. Yao, "Robust route optimization for gritting/salting trucks: A CERCIA experience," *IEEE Comput. Intell. Mag.*, vol. 1, no. 1, pp. 6–9, Feb. 2006.
[17] J. Brandão and R. Eglese, "A deterministic tabu search algorithm for the capacitated arc routing problem," *Comput. Oper. Res.*, vol. 35, no. 4, pp. 1112–1126, Apr. 2008.
[18] T. Z. Jiang and F. G. Yang, "An evolutionary tabu search for cell image segmentation," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 32, no. 5, pp. 675–678, Oct. 2002.
[19] S. Pierre and F. Houeto, "Assigning cells to switches in cellular mobile networks using taboo search," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 32, no. 3, pp. 351–356, Jun. 2002.
[20] G. N. Frederickson, "Approximation algorithms for some postman problems," *J. ACM*, vol. 26, no. 3, pp. 538–554, Jul. 1979.
[21] J. S. DeArmon, "A comparison of heuristics for the capacitated Chinese postman problems," M.S. thesis, Univ. Maryland, College Park, MD, 1981.
[22] E. Benavent, V. Campos, E. Corberán, and E. Mota, "The capacitated arc routing problem: Lower bounds," *Networks*, vol. 22, no. 7, pp. 669–690, Dec. 1992.
[23] R. W. Eglese, "Routing winter gritting vehicles," *Discrete Appl. Math.*, vol. 48, no. 3, pp. 231–244, Feb. 1994.
[24] 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, MA: Kluwer, 1996, pp. 633–650.
[25] L. Y. O. Li and R. W. Eglese, "An interactive algorithm for vehicle routing for winter-gritting," *J. Oper. Res. Soc.*, vol. 47, no. 2, pp. 217–228, 1996.
[26] J. M. Belenguer and E. Benavent, "A cutting plane algorithm for the capacitated arc routing problem," *Comput. Oper. Res.*, vol. 30, no. 5, pp. 705–728, Apr. 2003.
[27] D. Ahr, "Contributions to multiple postmen problems," Ph.D. dissertation, Rupercht-Karls-Universitat, Heidelberg, Germany, 2004.
[28] 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.
[29] H. Longo, D. A. M. Poggi, and E. Uchoa, "Solving capacitated arc routing problems using a transformation to the CVRP," *Comput. Oper. Res.*, vol. 33, no. 6, pp. 1823–1837, Jun. 2006.
[30] E. G. Coffman, M. R. Garey, and D. S. Johnson, "Approximation algorithms for bin-packing—An updated survey," in *Algorithm Design for Computer System Design*, G. Ausiello, M. Lucertini, and P. Serafini, Eds. Berlin, Germany: Springer-Verlag, 1984, pp. 49–106.
[31] Z. Michalewicz and G. Nazhiyath, "Genocop III: A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints," in *Proc. IEEE Int. Conf. Evol. Comput.*, Perth, Australia, 1995, vol. 2, pp. 647–651.
[32] K. Harada, J. Sakuma, I. Ono, and S. Kobayashi, "Constraint-handling method for multi-objective function optimization: Pareto descent repair operator," in *Proc. 4th Int. Conf. Evol. Multi-Criterion Optimization*, Sendai, Japan, 2007, pp. 156–170.
[33] H. Handa, K. Watanabe, O. Katai, T. Konishi, and M. Baba, "Coevolutionary genetic algorithm for constraint satisfaction with a genetic repair operator for effective schemata formation," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Tokyo, Japan, 1999, vol. 3, pp. 616–621.
[34] P. C. Chu and J. E. Beasley, "A genetic algorithm for the multidimensional knapsack problem," *J. Heuristics*, vol. 4, no. 1, pp. 63–86, Jun. 1998.
[35] M. Gröbner and P. Wilke, "Optimizing employee schedules by a hybrid genetic algorithm," in *Proc. EvoWorkshops Appl. Evol. Comput.*, 2001, vol. 2037, pp. 463–472.
[36] Q. Zhang, J. Sun, and E. Tsang, "An evolutionary algorithm with guided mutation for the maximum clique problem," *IEEE Trans. Evol. Comput.*, vol. 9, no. 2, pp. 192–200, Apr. 2005.
[37] H. Ueda, D. Ouchi, K. Takahashi, and T. Miyahara, "A co-evolving timeslot/room assignment genetic algorithm technique for university timetabling," in *Proc. Practice Theory Automated Timetabling III*, 2001, vol. 2079, pp. 48–63.
[38] C. A. Coello Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art," *Comput. Methods Appl. Mech. Eng.*, vol. 191, no. 11/12, pp. 1245–1287, Jan. 2002.

Yi Mei received the B.S. degree in mathematics from the University of Science and Technology of China, Hefei, China, in 2005, where he is currently working toward the Ph.D. degree in the Nature Inspired Computation and Applications Laboratory, Department of Computer Science and Technology.

His current research interests include memetic algorithm, tabu search, and other metaheuristics for solving arc routing problem.



Ke Tang (M'07) received the B.Eng. degree from the Department of Control Science and Engineering, Huazhong University of Science and Technology, Wuhan, China, in 2002 and the Ph.D. degree from the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, in 2007.

He is currently an Associate Professor with the Department of Computer Science and Technology, University of Science and Technology of China, Hefei, China. His major research interests include machine learning, pattern analysis, evolutionary computation, data mining, metaheuristic algorithms, and real-world applications.

Dr. Tang is a member of the Computational Intelligence Society of IEEE and the Chair of IEEE Task Force on Large Scale Global Optimization.

Xin Yao (M'91–SM'96–F'03) received the B.Sc. degree from the University of Science and Technology of China (USTC), Hefei, China, in 1982, the M.Sc. degree from the North China Institute of Computing Technology, Beijing, China, in 1985, and the Ph.D. degree from USTC in 1990.

From 1985 to 1990, he was an Associate Lecturer and Lecturer with USTC, while working toward the Ph.D. degree on simulated annealing and evolutionary algorithms. He took up a Postdoctoral Fellowship in the Computer Sciences Laboratory, Australian National University, Canberra, A.C.T., Australia, in 1990, and continued his work on simulated annealing and evolutionary algorithms. He was with the Knowledge-Based Systems Group, Commonwealth Scientific and Industrial Research Organization Division of Building, Construction and Engineering, Melbourne, Vic., Australia, in 1991, working primarily on an industrial project on automatic inspection of sewage pipes. He returned to Canberra in 1992 to take up a lectureship in the School of Computer Science, University College [University of New South Wales at Australian Defense Force Academy (ADFA)], ADFA, Canberra, where he was later promoted to a Senior Lecturer and Associate Professor. Attracted by the English weather, he moved to the University of Birmingham, Birmingham, U.K., as a Professor of computer science in 1999. He is currently the Director of the Centre of Excellence for Research in Computational Intelligence and Applications, School of Computer Science, University of Birmingham, and a Changjiang (Visiting) Chair Professor (Cheung Kong Scholar) at USTC. He is an Associate Editor or Editorial Board Member of ten journals and the Editor of the World Scientific Book Series on Advances in Natural Computation. He has given more than 50 invited keynote and plenary speeches at conferences and workshops worldwide. His major research interests include evolutionary artificial neural networks, automatic modularization of machine learning systems, evolutionary optimization, constraint handling techniques, computational time complexity of evolutionary algorithms, coevolution, iterated prisoner's dilemma, data mining, and real-world applications. He has more than 250 refereed publications.

Dr. Yao was awarded the President's Award for Outstanding Thesis by the Chinese Academy of Sciences for his Ph.D. work on simulated annealing and evolutionary algorithms in 1989. He won the 2001 IEEE Donald G. Fink Prize Paper Award for his work on evolutionary artificial neural networks. He is the Editor-in-Chief of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION.