A Route Clustering and Search Heuristic for Large Scale Multi-Depot Capacitated Arc Routing Problem

Yuzhou Zhang, Yi Mei, *Senior Member, IEEE*, Shihua Huang, Xin Zheng and Cuijuan Zhang

***Abstract*—The capacitated arc routing problem (CARP) has attracted much attention for its many practical applications. The large scale multi-depot CARP (LSMDCARP) is an important CARP variant which is very challenging due to its vast search space. To solve LSMDCARP, we propose an iterative improvement heuristic named Route Clustering and Search Heuristic (RoCaSH). In each iteration, it first (re)decomposes the original LSMDCARP into a set of smaller single-depot CARP sub-problems using route cutting off and clustering techniques. Then, it solves each sub-problem using the effective Ulusoy's split operator and local search. On one hand, the route clustering helps the search for each sub-problem by focusing more on the promising areas. On the other hand, the sub-problem solving provides better routes for the subsequent route cutting off and clustering, leading to better problem decomposition. The proposed RoCaSH was compared with the state-of-the-art MDCARP algorithms on a range of MDCARP instances, including different problem sizes. The experimental results showed that RoCaSH significantly outperformed the state-of-the-art algorithms, especially for the large scale instances. It managed to achieve much better solutions within a much shorter computational time.**

***Index Terms*—Capacitated arc routing problem, multi-depot, scalability, route clustering, search heuristic**

I. Introduction

As a complex combinatorial optimization problem, the Capacitated Arc Routing Problem (CARP) [1] is well-known for its wide applications in the real world, such as winter gritting [2], street salting [3], mail delivery [4], urban waste collection [5], and snow removal [6], [7]. The aim of the problem is to find an optimal set of routes for a fleet of vehicles based at a depot to serve the edges in a given network subject to specific constraints so that the total cost is minimized. There have been extensive research on CARP, and a large amount of competitive approaches have been proposed, e.g. [8]–[17].

Multi-depot CARP (MDCARP) [18] is an important CARP variant, which consists of multiple depots located at different regions of the graph. It reflects a variety of real-world considerations, such as garbage collection, road sprinkling and street sweeping in the large districts, in which more than one depot are involved. Compared with traditional single-depot CARP, MDCARP has a much larger and more complex solution space, as one has to allocate the tasks (i.e. edges) to the depots.

In real world, the problem size of CARP or MDCARP can usually be very large (e.g., thousands of streets in a city need to be served for waste collection). The existing approaches to MDCARP were only examined on the small and medium instances. For example, in [19] two algorithms (i.e., HACA and MDMA) were tested on the *mdgdb* set in which the number of required edges varies from 11 to 55. Though certain works (e.g., [20]–[27]) on large scale CARP (LSCARP) have been proposed, these works only considered the problems with a single depot.

As an efficient strategy for large scale problems, the Divide and Conquer (DC) strategy aims to decompose the whole problem into several sub-problems, and then tackle each sub-problem independently or cooperatively. Finally, all the obtained sub-solutions are concatenated together to form a complete solution to the original problem. The DC strategy has achieved great success in a range of problems, such as multi-objective optimization [28], [29], continuous optimization [30]–[32], vehicle routing [33], [34], job shop scheduling [35], [36] and other large-scale optimization problems [37], [38]. It has also been successfully applied to large scale CARP with a single depot (e.g., [22][33]). Intuitively, we can use the DC strategy in large scale MDCARP (LSMDCARP), and divide the problem into several single-depot CARP sub-problems (each required edge, i.e., *task*, is associated with a depot). However, there is no work to solve MDCARP with the DC strategy so far, not to mention LSMDCARP. It is challenging to find the appropriate assignment.

To address the above issues, in this paper, we propose a route clustering and search heuristic (RoCaSH) for solving LSMDCARP. The proposed RoCaSH employs a novel route clustering scheme and a route cut-off operator to achieve an effective allocation of the tasks to the depots. Then, the smaller single-depot CARP sub-problem associated to each depot is solved by the Ulusoy's split operator and local search, which have been effective for solving LSCARP. To verify the effectiveness of RoCaSH, it was examined on the MDCARP datasets with a wide range of problem sizes. The results showed that the proposed RoCaSH significantly outperformed the state-of-the-art algorithms for MDCARP, especially for the instances with large problem sizes.

The rest of this paper is organized as follows. First, MDCARP is described in Section II. Then, the proposed RoCaSH is introduced in Section III. Afterwards, Section IV

Yuzhou Zhang, Shihua Huang, Xin Zheng and Cuijuan Zhang are with School of Computer and Information, Anqing Normal University, Anqing 246133, China. (e-mails: zhangyuzhou@aqnu.edu.cn, 342268451@qq.com, zxaoyou@gmail.com, zcjmath@163.com).

Yi Mei is with the School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. (e-mail: yi.mei@ecs.vuw.ac.nz).

Corresponding author: Yi Mei.

provides the experimental studies and discussions. Finally, the conclusions and future work are given in Section V.

II. Background and Related Work

A. Multi-Depot Capacitated Arc Routing Problem

In MDCARP, an undirected connected graph $G(V, E)$ is given, where V is the vertex set, and E is the edge set. The vertex set V contains two types of vertices, the *depot* vertices $V^{(D)} \subseteq V$ and *non-depot* vertices $V^{(N)} \subseteq V$, i.e. $V = V^{(N)} \cup V^{(D)}$. Each edge $e \in E$ has three non-negative properties: a demand $dem(e)$, a service cost $sc(e)$ and a deadheading cost $dc(e)$ (which is the cost caused by traversing without service). An edge is called a *task* if its demand is positive (i.e. it is required to be served). Let the set of all the tasks be denoted as $T \subseteq E$. The tasks are to be served by a fleet of identical vehicles with limited capacity of Q. The goal of MDCARP is to find a set of least-cost routes for the vehicles to serve all the tasks subject to the following constraints:

- Each vehicle starts from a depot $v_d^{(D)} \in V^{(D)}$, and returns to the same depot;
- Each task is served exactly once by a vehicle;
- The total demand served by each vehicle cannot exceed the vehicle's capacity Q.

Note that the conventional CARP is a special case of MDCARP, where there is a single depot, i.e., $|V^{(D)}| = 1$.

We give the problem formulation under the task representation [39]. Specifically, each task $t = (u, v) \in T$ is assigned two IDs x_1 and x_2, each representing one of its directions. Each task ID has a *head vertex* and a *tail vertex*. For example, the head and tail vertices of x_1 are $hv(x_1) = u$ and $tv(x_1) = v$, while the head and tail vertices of x_2 are $hv(x_2) = v$ and $tv(x_2) = u$. The two task IDs are inverse to each other, i.e. $x_1 = inv(x_2)$ and $x_2 = inv(x_1)$. Each task ID is also associated with a demand, a service cost, and a deadheading cost, which are set based on their corresponding task. For example, $dem(x_1) = dem(x_2) = dem(t)$, $sc(x_1) = sc(x_2) = sc(t)$, and $dc(x_1) = dc(x_2) = dc(t)$. Given a graph with $|T|$ undirected tasks, we will obtain $2|T|$ task IDs. In addition, we denote the depot loop for each depot using a separate ID $x_d^{(D)}$ ($d = 1, \ldots, |V^{(D)}|$), where $hv(x_d^{(D)}) = tv(x_d^{(D)}) = v_d^{(D)}$, $dem(x_d^{(D)}) = sc(x_d^{(D)}) = dc(x_d^{(D)}) = 0$, $inv(x_d^{(D)}) = x_d^{(D)}$. Overall, we will have $2|T| + |V^{(D)}|$ task IDs.

Under the above representation, the kth route associated with depot $v_d^{(D)}$ can be represented as:

$$R_{d,k} = (x_{d,k,1}, x_{d,k,2}, \ldots, x_{d,k,|R_{d,k}|}), \tag{1}$$

where $|R_{d,k}|$ indicates the number of the tasks in $R_{d,k}$. A solution $\mathbf{S}$ consists of all the routes and can be represented as:

$$\mathbf{S} = (\mathbf{R}_1, \ldots, \mathbf{R}_{|V^{(D)}|}), \tag{2}$$

where $\mathbf{R}_d = (R_{d,1}, \ldots, R_{d,|\mathbf{R}_d|})$ consists of the routes associated with $v_d^{(D)}$. $|\mathbf{R}_d|$ is the number of the routes associated with the depot $v_d^{(D)}$. The total cost of S is calculated as follows:

$$tc(\mathbf{S}) = \sum_{d=1}^{|V^{(D)}|} tc(\mathbf{R}_d), , \tag{3}$$

where

$$tc(\mathbf{R}_d) = \sum_{k=1}^{|\mathbf{R}_d|} \sum_{i=1}^{|R_{d,k}|-1} \Big[sc(x_{d,k,i}) + \varphi(tv(x_{d,k,i}), hv(x_{d,k,i+1})) \Big], \tag{4}$$

where $\varphi(u, v)$ indicates the total deadheading cost of the edges on the shortest path from vertex u to v, which can be pre-calculated by Dijkstra's algorithm [40].

Then, we give the problem formulation of MDCARP as follows.

$$\min \quad tc(\mathbf{S}), \tag{5}$$

$$s.t. \quad x_{d,k,1} = x_d^{(D)}, \ \forall\, d = 1, \ldots, |V^{(D)}|, \ k = 1, \ldots, |\mathbf{R}_d|, \tag{6}$$

$$x_{d,k,|R_{d,k}|} = x_d^{(D)}, \ \forall\, d = 1, \ldots, |V^{(D)}|, \ k = 1, \ldots, |\mathbf{R}_d|, \tag{7}$$

$$\sum_{d=1}^{|V^{(D)}|} \sum_{k=1}^{|\mathbf{R}_d|} (|R_{d,k}| - 2) = |T|, \tag{8}$$

$$x_{d,k,i} \neq x_{d',k',i'}, \ \forall\, d \neq d' \text{ or } k \neq k' \text{ or } i \neq i', \tag{9}$$

$$x_{d,k,i} \neq inv(x_{d',k',i'}), \ \forall\, d \neq d' \text{ or } k \neq k' \text{ or } i \neq i', \tag{10}$$

$$\sum_{i=1}^{|R_{d,k}|} dem(x_{d,k,i}) \leq Q, \ \forall\, d = 1, \ldots, |V^{(D)}|, \quad 1 \leq k \leq |\mathbf{R}_d|. \tag{11}$$

Eq. (5) is the objective function, which is to minimize the total cost calculated by Eq. (3). Eqs. (6) and (7) indicate that each route starts and ends at the same depot. Eqs. (8), (9) and (10) ensure that all the tasks in T must be served exactly once. Eq. (11) guarantees that the capacity constraint is satisfied.

B. Related Work

CARP has attracted a lot of attention since 1981 when it was presented by Golden [1]. The early approaches to CARP can be divided into two types, i.e., mathematical programming (e.g., [17], [20], [41]–[43]) and heuristic algorithms (e.g., [10], [13], [44], [45]). However, most of these studies aimed at the small and medium scale problems (e.g. the tested instances have no more than 190 tasks).

In 2008, Brandão and Eglese [46] generated a large *EGL-G* dataset. In the new dataset, the number of tasks is up to 375, which is much larger than that of the previously used datasets (e.g., *gdb*, *val* and *egl*). This raises the issue of scalability of the algorithms, which leads to the emergence of studies for Large Scale CARP (LSCARP) [15], [20], [22], [24], [26], [46]. In 2017, Tang *et al*. [27] addressed the scalability issue on two much larger datasets, i.e., *Hefei* and *Beijing*, in which the size of the instances is over 3000.

The Divide-and-Conquer (DC) strategy is an effective technique to decompose large problems into smaller ones and solve them independently or cooperatively. For LSCARP, there have been a number of effective algorithms proposed based on the DC strategy. For example, Mei *et al*. [22] proposed

a cooperative co-evolution algorithm called RDG-MAENS, in which the entire evolutionary process is divided into a number of cycles. At the beginning of each cycle, the routes of the best-so-far solution are grouped together based on a Route Distance Grouping (RDG) decomposition scheme. The decomposition scheme can guarantee that the optimal solution under the decomposition can be no worse than the current solution. Tang *et al*. [27] proposed an individual search algorithm based on a hierarchical decomposition, which is called SAHiD. In SAHiD, the tasks are grouped together in an hierarchical way to form the so-called "*virtual tasks*". The search is repeated by heuristically breaking and forming virtual tasks.

In contrast with CARP, MDCARP has another level of challenge, which is to assign tasks/routes to the depots. Since introduced in 2000 [18], MDCARP received a number of research interests over past years (e.g., [19], [47]–[52]). In [47], Zhu *et al*. proposed a Hybrid Genetic Algorithm (HGA) to tackle the MDCARP, in which a route is considered as an individual, and the population is essentially a set of routes. In [19], a memetic algorithm was proposed for MDCARP by Kansou and Yassine, named MDMA, in which each depot involves a set of routes. The initial individuals are created by repetitively assigning a randomly selected task to its nearest depot using the best insertion strategy. Then, the individuals undergo the process of evaluation, selection, crossover and local search, and the best solution is returned after a certain number of iterations. MDMA is essentially an approach to the basic CARP without particular consideration of multiple depots except in the initialization process. Moreover, there have been a lot of work on different variants of MDCARP. In [49], a MILP model was presented for the asymmetric MDCARP, and the valid inequalities tightening its LP relaxation were taken into account. In [51], a memetic algorithm was developed for the MDCARP with multi-objectives, and a hybrid ant colony optimization algorithm was proposed for multi-depot periodic open capacitated arc routing problem in [52].

When the problem size grows, the existing approaches to MDCARP become less effective, as they tend to solve the problem as a whole. For large scale optimization problems, the DC strategy has achieved great success in a wide range of problems, such as single-objective [31], [53], [54] and multi-objective continuous optimization [28]–[30], [32], [55], job shop scheduling [35], [36], [56], vehicle routing [33], [34], timetabling [57], [58], and LSCARP [21], [22], [24], [27].

As the counterpart of CARP, vehicle routing problem (VRP) serves the vertices [59] rather than the edges. There have been extensive studies on multi-depot VRP and large scale VRP with the DC strategy. Lim *et al*. proposed a single-stage approach to multi-depot VRP, in which the assignment of the customers and the routing for each depot are combined, so that the routing information can be fed back to the future customer assignments [60]. Oliveira *et al*. proposed a cooperative co-evolutionary algorithm, which divides all the customers among the depots, and then solves the VRP sub-problem associated to each depot (along with its allocated customers) by a parallel evolution strategy [33]. Lalla-Ruiz *et al*. decomposed the multi-depot VRP into sub-problems by a partial optimization meta-heuristic under special intensification conditions (called POPMUSIC), in which the sub-problem is built from the selected seed part [61]. Besides, there have been various approaches to the extended versions of multi-depot VRP [62]–[65] and DC approaches to large scale VRP with a single depot, e.g., [66]–[68].

However, the existing approaches for VRP cannot be directly applied to LSMDCARP due to two reasons: (1) they are designed for solving either multi-depot or large scale VRP, but not both; (2) CARP tends to be much larger and harder to be solved than the VRP counterpart [11]. For example, a CARP with $|T|$ tasks may be converted to a corresponding VRP with $2|T| + 1$ vertices.

In summary, all the existing studies for MDCARP focus on the small scale instances, and solve the problem as a whole. So far, there has been no effective approach to address the scalability issue for LSMDCARP. On other hand, although the DC strategy has achieved great success in tackling LSCARP effectively, these approaches are not directly applicable to LSMDCARP. For example, they consider only a single depot, [22], [24], [27], and cannot effectively allocate the tasks/routes among different depots. In this paper, we investigate the MDCARP, and propose a route clustering and search heuristic (RoCaSH) for it. The proposed RoCaSH employs a novel route clustering scheme and a Cutting Off Operator (RCO) to allocate the tasks to the depots effectively, and then uses the Ulusoy's split operator as well as an iterative local search procedure to boost the optimization performance of the sub-problem associated to each depot. The details of RoCaSH is described in Section III.

III. Route Clustering and Search Heuristic

It is natural for us to use the DC strategy to decompose the LSMDCARP into several smaller single-depot CARPs due to the structure of the problem. However, there are three key problems which must be solved. The first is the development of the decomposition scheme. It is well-known that an appropriate decomposition scheme is very important for the complex problem, as good scheme can gather the elements which are intense and peel off those loose ones. The second is the design of the optimization method for the CARP sub-problems obtained by the decomposition scheme. The third is how to design effective co-operations among the optimization process of the CARP sub-problems. We propose the RoCaSH to LSMDCARP to address the above three issues.

A. The Overall Framework

The overall framework of RoCaSH is described in Algorithm 1. The flowchart is given in Fig. 1 (along with how the solution changes at each step). It is an individual-based search algorithm, which hybridizes a *route clustering* scheme to decompose the problem into smaller sub-problems and a local search to solve each sub-problem separately. The route clustering and local search are interleaved with each other. On one hand, the route clustering can help the local search improve the solution more efficiently. On the other hand, the

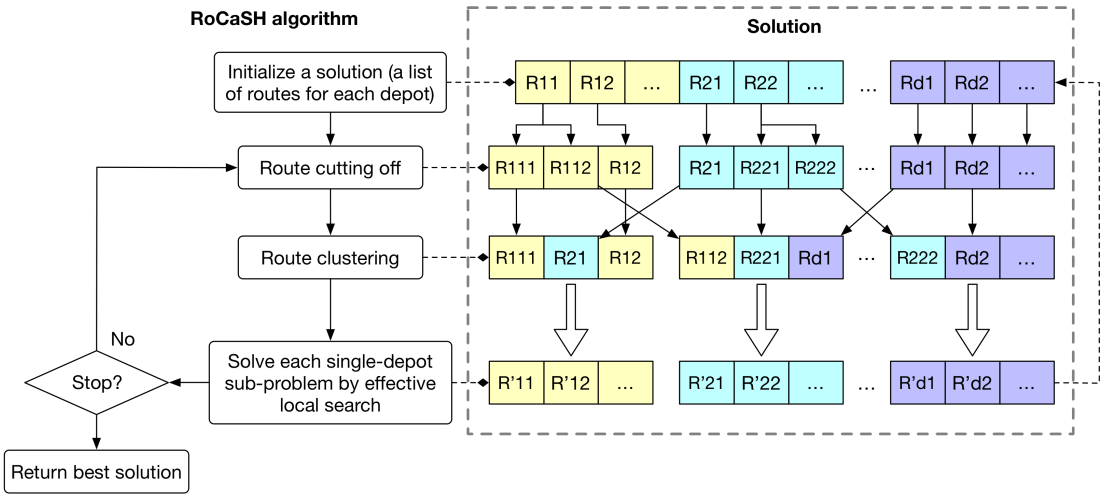

Fig. 1. The flowchart of the proposed RoCaSH.

improved solution by the local search can give feedback to further improve the route clustering.

In Algorithm 1, a solution S is first initialized heuristically (line 2). The solution is composed of a set of $|V^{(D)}|$ sub-solutions $(S_1, \ldots, S_{|V^{(D)}|})$, where S_d $(d = 1, \ldots, |V^{(D)}|)$ is a set of routes associated with a depot $v_d^{(D)}$. During the search process, the average rank $\overline{rank}(S)$ of the links in S is calculated in line 5. Then, for each depot, the routes are first cut into sub-routes by the route cutting off operator [69], which tends to cut the poor (long distance) links rather than the promising (short distance) links (lines 6–11). Then, the sub-routes of all the depots are re-clustered to the depots by the `ClusterRoute`$(\cdot)$ procedure (line 12). Afterwards, the routes for each depot are improved by the Ulusoy's splitting procedure [8] and local search (lines 13–18). The whole algorithm stops when the maximal number of iterations or the maximal number of iterations without improvement is reached. Note that throughout the entire process, the order of the routes in the (sub-)solutions are always retained. In other words, we maintain the routes as *lists* rather than *sets* in the algorithm. This way, we expect to improve the search efficiency without losing much useful information. In the following, we will describe the initialization, route cut off operator, route clustering scheme, and local search in more detail.

Algorithm 1 The Route Clustering and Search Heuristic

1: **procedure** `RoCaSH` (The input instance with task set T, depot set V_D, maximum iterations $MaxIter$, maximum iterations without improvement $MaxNoImp$)
2: Initialize $S = (S_1, \ldots, S_{|V^{(D)}|}) =$ `init`$(T, V^{(D)})$;
3: $iter = 1$, $noimp = 0$;
4: **while** $iter \leq MaxIter$ and $noimp \leq MaxNoImp$ **do**
5: Calculate the average rank $\overline{rank}(S)$ of the links in S;
6: **for** $d = 1 \rightarrow |V^{(D)}|$ **do**
7: $\mathbf{R}_d = ()$;
8: **for** each route R in S_d **do**
9: Cut R into sub-routes using the route cutting off operator based on $\overline{rank}(S)$, and add the obtained sub-routes to the end of $\mathbf{R}_d$ in order;
10: **end for**
11: **end for**
12: `ClusterRoute`$(\mathbf{R}_1, \ldots, \mathbf{R}_{|V^{(D)}|})$;
13: **for** $d = 1 \rightarrow |V^{(D)}|$ **do**
14: Build a giant route by merging all the routes $R \in \mathbf{R}_d$ in order;
15: Apply 2-OPT on the giant route to obtain better one;
16: Split the giant route with Ulusoy's splitting procedure [8] to obtain the sub-solutions S'_d;
17: Improve S'_d with the local search;
18: **end for**
19: Set $S' = (S'_1, \ldots, S'_{|V^{(D)}|})$;
20: **if** S' is better than S **then**
21: $S = S'$, $noimp = 0$;
22: **else**
23: $noimp = noimp + 1$;
24: **end if**
25: **end while**
26: $iter = iter + 1$;
27: **return** the best feasible solution S;
28: **end procedure**

B. Initialization

The pseudo-code of the solution initialization is shown in Algorithm 2. At first, each task $t \in T$ is allocated to a random depot, and a single-task route $(x_d^{(D)}, x(t), x_d^{(D)})$ is formed, where $x(t)$ is a random direction (i.e. task ID) of t. Then, all the single-task routes are re-clustered to the depots by the route clustering scheme. After that, the giant route concatenated by all the single-task routes undergoes the Ulusoy's splitting procedure and local search for each depot.

C. Route Cutting Off Operator

It has been shown that grouping the routes can ensure that the optimal solution under the new grouping is no worse than the current solution [22]. However, in practice it may not be sufficiently effective, especially when the routes contain tasks that are connected by poor links (i.e. with long distance), and should have been served in different routes or even allocated to different depots. To address the above issue, we employ the route cutting off operator [69] before reclustering the routes to cut the routes into sub-routes. The route cutting off operator

Algorithm 2 The solution initialization

procedure `Init` (task set T, depot set $V^{(D)}$)
  for $d = 1 \rightarrow |V^{(D)}|$ **do**
    Initialize a route list $\mathbf{R}_d = ()$;
  end for
  for each task $t \in T$ **do**
    Randomly select an ID (direction) $x(t)$;
    Randomly select $d \in \{1, \ldots, |V^{(D)}|\}$;
    Create a route $R(t) = (x_d^{(D)}, x(t), x_d^{(D)})$;
    $\mathbf{R}_d = (\mathbf{R}_d, R(t))$;
  end for
  `ClusterRoute`$(\mathbf{R}_1, \ldots, \mathbf{R}_{|V^{(D)}|})$;
  for $d = 1 \rightarrow |V^{(D)}|$ **do**
    Build a giant route by merging all the routes $R \in \mathbf{R}_d$ in order;
    Split the giant route with Ulusoy's splitting procedure to obtain the sub-solutions S_d;
    Improve S_d with the local search with 2-OPT;
  end for
  return $S = (S_1, \ldots, S_{|V^{(D)}|})$;
end procedure

splits the routes according to the quality of the links. After the cut, the sub-routes tend to maintain the better links (i.e., with shorter deadheading costs than the other links with shared end-nodes). The subsequent route clustering scheme is then conducted on the sub-routes rather than the original entire routes. This way, we expect to achieve more fine-grained allocation of the tasks to the depots.

Algorithm 3 describes the route cutting off operator. It takes a route R, the average rank $\overline{rank}(S)$ (calculated in line 5 of Algorithm 1), and two parameters λ and θ as inputs. First, for each link, we sort the links from the same starting task to all the other tasks from shortest to longest, and get the rank of the link (ties are broken by sharing the rank). Then, the links of R are grouped into *good* and *poor* links based on their ranks relative to $\overline{rank}(S)$. Afterwards, the route R has a (smaller) probability λ of being split by breaking a good link and a (larger) probability θ of being split by breaking a poor link. As a result, a route may be split into at most 3 sub-routes (when breaking both a good link and a poor link). More details of RCO can be found from [69].

In RCO, the most time consuming step seems to be the sorting process in line 4, whose time complexity is $O(|T| \log(|T|))$. However, in practice one can calculate the rank of each task externally and store the ranks globally. Therefore, the complexity of line 4 can be reduced to $O(1)$ in practice. Overall, the complexity of the RCO operator is $O(|T|)$.

D. Route Clustering Scheme

The pseudo code of the route clustering scheme is given in Algorithm 4. Given an *ordered* list of routes, the scheme uses three criteria to re-cluster the routes. First, the routes associated to the depots are reset. That is, each depot $v_d^{(D)}$ is associated with a single empty route $R_{d,0} = (t_d^{(D)}, t_d^{(D)})$, and all the given routes are unassigned. Then, for each unassigned route R, the distance from the route to each depot is defined as the average distance from the route to the routes already assigned to that depot, i.e.,

$$\Delta_d(R) = \frac{1}{|\mathbf{R}_d|} \sum_{R_d \in \mathbf{R}_d} \delta(R, R_d), \tag{12}$$

Algorithm 3 Route cutting off operator

procedure `RCO` (a route R, the average rank $\overline{rank}(S)$, cutting probabilities λ and θ)
  Set good links $\mathcal{L}_g = \emptyset$, poor links $\mathcal{L}_p = \emptyset$;
  for each link $(t_1, t_2) \in R$ **do**
    Sort all the links $\{(t_1, t') | t' \in T\}$ from shortest to longest, and get the rank of (t_1, t_2);
  end for
  for each link $(t_1, t_2) \in R$ **do**
    if the rank of (t_1, t_2) is better than $\overline{rank}(R)$ **then**
      $\mathcal{L}_g = \mathcal{L}_g \cup (t_1, t_2)$;
    else
      $\mathcal{L}_p = \mathcal{L}_p \cup (t_1, t_2)$;
    end if
  end for
  Randomly sample $r_1 \in U[0, 1]$;
  if $r_1 < \lambda$ **then**
    Randomly select a link $\mathcal{L}_g$ and split R from it;
  end if
  Randomly sample $r_2 \in U[0, 1]$;
  if $r_2 < \theta$ **then**
    Randomly select a link from $\mathcal{L}_g$ and split R from it;
  end if
  return the resultant sub-routes;
end procedure

where the distance between two routes R_1 and R_2 is defined as

$$\begin{aligned}\delta(R_1, R_2) = \frac{1}{4}(&\varphi(tv(R_1), tv(R_2)) + \varphi(tv(R_1), hv(R_2)) \\ &+ \varphi(hv(R_1), tv(R_2)) + \varphi(hv(R_1), hv(R_2))),\end{aligned} \tag{13}$$

where $tv(R)$ and $hv(R)$ stand for the tail node of the last task and the head node of the first task of the route, respectively. For an empty route $R_{d,0} = (t_d^{(D)}, t_d^{(D)})$, both $tv(R_{d,0})$ and $hv(R_{d,0})$ are defined as the depot $t_d^{(D)}$.

Based on the above defined distance measure from a route to a depot, the first criterion selects all the unassigned routes whose distance to the closest depot is at least 10% shorter than that to the second closest depot. Among such unassigned routes, it further selects the one with the largest gap, and assign it to its closest depot. Otherwise, if none of the unassigned routes has an obvious closest depot, the second criterion considers the variance of the distances from the route to the routes assigned to the same depot. It selects all the unassigned routes for which there is at least 40% gap between the smallest and second smallest variances. Again, if there exist such routes, it further selects the one with the largest gap, and assigns it to the depot with the smallest variance. If both criteria fail, the third criterion uses the distance to the closest depot. In this case, the unassigned route with the smallest distance to the closest depot is selected and assigned to its closest depot. Finally, the empty routes are removed from the route lists.

The three criteria used for assigning the routes are inspired from the three criteria clustering, which has been shown to be effective for solving multi-depot VRP [70]. In the route clustering scheme, the three criteria are based on the route distance between the unassigned route and assigned ones, and it runs in $O(|R|^2)$ where $|R|$ is the number of routes in the list.

Algorithm 4 The three-criteria route clustering procedure

1: **procedure** `ClusterRoute` (route lists $\mathbf{R}_1, \ldots, \mathbf{R}_{|V^{(D)}|}$)
2: Set the unassigned route list $\bar{\mathbf{R}} = (\mathbf{R}_1, \ldots, \mathbf{R}_{|V^{(D)}|})$;
3: **for** $d = 1 \to |V^{(D)}|$ **do**
4: Construct an empty route $R_{d,0} = (x_d^{(D)}, x_d^{(D)})$;
5: $\mathbf{R}_d = (R_{d,0})$;
6: **end for**
7: **while** $\bar{\mathbf{R}} \neq ()$ **do**
8: Set candidate routes $\Pi_1 = \emptyset$; ▷ first criterion
9: **for** each route $R \in \bar{\mathbf{R}}$ **do**
10: **for** $d = 1 \to |V^{(D)}|$ **do**
11: $\Delta_d(R) = \frac{1}{|\mathbf{R}_d|} \sum_{R_d \in \mathbf{R}_d} \delta(R, R_d)$;
12: **end for**
13: Calculate the gap $\theta_1(R)$ between the smallest and second smallest values in $\{\Delta_d(R) \mid d = 1, \ldots, |V^{(D)}|\}$;
14: **if** $\theta_1(R) \geq 10\%$ **then**
15: $\Pi_1 = \Pi_1 \cup R$;
16: **end if**
17: **end for**
18: **if** $\Pi_1 \neq \emptyset$ **then**
19: Select the route with max gap $R^* = \arg\max_{R \in \Pi_1} \theta_1(R)$;
20: Select the closest route list $d^* = \arg\min_d \Delta_d(R^*)$;
21: $\mathbf{R}_{d^*} = (\mathbf{R}_{d^*}, R^*)$, $\bar{\mathbf{R}} = \bar{\mathbf{R}} \setminus R^*$;
22: **goto** line 7;
23: **end if**
24: $\Pi_2 \leftarrow \emptyset$; ▷ second criterion
25: **for** each route $R \in \bar{\mathbf{R}}$ **do**
26: **for** $d = 1 \to |V_D|$ **do**
27: $\sigma_d(R) = \mathtt{var}(\delta(R, R_d) \mid R_d \in \mathbf{R}_d)$;
28: Calculate the gap $\theta_2(R)$ between the smallest and second smallest values in $\{\sigma_d(R) \mid d = 1, \ldots, |V^{(D)}|\}$;
29: **if** $\theta_2(R) \geq 40\%$ **then**
30: $\Pi_2 = \Pi_2 \cup R$;
31: **end if**
32: **end for**
33: **end for**
34: **if** $\Pi_2 \neq \emptyset$ **then**
35: Select $R^* = \arg\max_{R \in \Pi_2} \theta_2(R)$;
36: Select the closest route list $d^* = \arg\min_d \sigma_d(R^*)$;
37: $\mathbf{R}_{d^*} = (\mathbf{R}_{d^*}, R^*)$, $\bar{\mathbf{R}} = \bar{\mathbf{R}} \setminus R^*$;
38: **goto** line 7;
39: **end if**
40: **for** each route $R \in \bar{\mathbf{R}}$ **do** ▷ third criterion
41: Calculate the closest distance $\Delta_{\min}(R) = \min_d \Delta_d(R)$;
42: **end for**
43: $R^* = \arg\min_{R \in \bar{\mathbf{R}}} \Delta_{\min}(R)$;
44: Select the closest route list $d^* = \arg\min_d \Delta_d(R^*)$;
45: $\mathbf{R}_{d^*} = (\mathbf{R}_{d^*}, R^*)$, $\bar{\mathbf{R}} = \bar{\mathbf{R}} \setminus R^*$;
46: **end while**
47: remove the empty routes $R_{d,0}$ $(d = 1, \ldots, |V^{(D)}|)$;
48: **end procedure**

E. Local Search

After the route clustering, merge and Ulusoy's split, the solution is further improved by the local search. In the local search, we use the single-insertion (SI), double-insertion (DI) and swap operators, which are commonly used move operators for CARP. If the best neighbour is better than the current solution, then it replaces the current solution, and the search continues. Otherwise, the search stops, and the current solution is returned. Since the three employed operators have small search step size, the search can only be done within a limited neighborhood of the incumbent solution. To enlarge the search space, before the Ulusoy's split, we apply another local search process on the giant route (using the 2-OPT operator to distinguish from the three basic move operators, to avoid getting stuck into the same local optimum).

TABLE I
THE PARAMETER SETTINGS OF ROCASH.

Parameter	Description	Value
$MaxIter$	Max iterations	5000
$MaxNoImp$	Max iterations without improvement	1000
λ	Prob. of cutting off good links in Algo. 3	0.1
θ	Prob. of cutting off poor links in Algo. 3	0.5

Each step of the local search has a complexity of $O(|T|^2)$, where $|T|$ is the number of tasks. Thus, the complexity of the local search process is $O(L_{ls}|T|^2)$, where L_{ls} is the expected number of steps in the local search.

F. Complexity Analysis

Overall, the complexity of RoCaSH consists of that of the RCO operator, route clustering and local search. The complexities of the RCO operator and route clustering are $O(|T|)$ and $O(|R|)$, respectively. The local search is the most time consuming step, with a complexity of $O(L_{ls}|T|^2)$. The preprocessing stage contains the calculations of the shortest distance between each pair of the nodes by Dijkstra's algorithm [40] (complexity of $O(|V|^3)$) and the rankings of the tasks (complexity of $|T|^2 \log(|T|)$. However, in practice the complexity of the preprocessing stage can be mostly ignored when comparing with the search process. Therefore, the overall complexity of RoCaSH is $O(MaxIter \times L_{ls} \times |T|^2)$, where $MaxIter$ is the maximal number of iterations, and L_{ls} is the expected steps during the local search.

IV. Experimental Studies

To verify the effectiveness of the proposed RoCaSH, we carried out experiments on a range of MDCARP datasets with different problem sizes. In the experiments, RoCaSH was compared with the Hybrid Genetic Algorithm (HGA) [47] and the Multi-Depot Memetic Algorithm (MDMA) [19], which are the two existing state-of-the-art algorithms for MDCARP. Note that another Hybrid Ant Colony Algorithm (HACA) was also proposed in [19]. However, it showed much worse performance than MDMA in the experimental studies. Thus, we omitted the comparison with HACA in our experimental studies.

HGA and MDMA focused mainly on the small-scale MDCARP instances, and thus are not divide-and-conquer methods. However, to the best of our knowledge, there is no existing study particularly for large scale MDCARP. The comparison with HGA and MDMA can also demonstrate the effectiveness of the proposed problem decomposition strategy.

A. Datasets

So far, there is no benchmark dataset specifically for MDCARP. Instead, the existing studies for MDCARP extended the single-depot CARP instances to multi-depot ones. For example, in [19], the *gdb* small scale CARP instances are converted into MDCARP instances. Specifically, each *gdb*

instance is converted into a double-depot *mdgdb* instance by switching the last vertex into the second depot.

In our experimental studies, we consider a wide range of instances with different problem sizes, and particularly focus on the large problem instances. Specifically, we consider the *mdgdb* instances generated in [19], as well as convert the larger *val*, *egl*, *EGL-G*, *Hefei* and *Beijing* instances into multi-depot ones following the idea in [19].

The *gdb* set consists of 23 small size instances with 11–55 edges. The *val* set contains 34 instances with 34 to 97 edges. The *egl* set consists of 24 instances with 98–190 edges, which are generated from the network of roads in Lancashire, U.K. All of the 24 instances are based on two graphs and constructed by varying required edges and capacity constraints. Likewise, The *EGL-G* dataset consists of 10 instances, which are also derived from the road network of Lancashire, UK, but with 375 edges. The dataset contains two groups *G*1 and *G*2, each with 5 instances. The instances belonging to the same group have the same task set, and different vehicle capacities. The *Hefei* and *Beijing* datasets were generated by Tang *et al*. from the maps of the two cities in China [27]. The *Hefei* dataset contains ten instances sharing a graph with 1212 edges, and the *Beijing* dataset also consists of ten instances sharing a graph with 3584 edges. The tasks for the instances of each dataset are randomly selected from the edge set, while the numbers of tasks are set from 10% to 100% of the number of edges.

The MDCARP instances are generated from the single-depot CARP instances based on the following steps.

- First we set the number of depots for the generated datasets based on the minimum number of vehicles required for the problems. Specifically, the *mdval* instances have 3 depots, and the *mdegl* and *mdEGL-G* instances have 4 and 8 depots respectively. The number of depots is set to 5 for the *mdHefei* and *mdBeijing* instances.
- Then, we set the depots uniformly among the vertices. The depot set contains the first vertex, as well as the vertices whose indices are $k \cdot \left\lfloor \frac{|V|}{m-1} \right\rfloor$, where m is the number of depots, and $k = 1, \ldots, m-1$. For example, if there are 50 vertices (index from 1 to 50) and 3 depots, then the depots contain the nodes 1, 25 and 50.

To be consistent with [19], the depot with the maximum index is replaced by the last vertex for the *mdgdb* dataset.

B. Parameter Settings

The parameter settings of the proposed RoCaSH are listed in Table I. The maximal number of iterations and maximal iterations without improvement are set to 5000 and 1000 respectively. For the RCO operator (Algorithm 3), the probabilities of cutting off the good and poor links are set to 0.1 and 0.5.

The stopping criteria of the compared HGA and MDMA are set according to the settings used in the original papers. Specifically, for HGA, the maximal number of iterations is 100000, and the maximal iterations without improvement is 2000. For MDMA, the maximal number of iterations is 5000. All the other parameters of the compared algorithms are set the same as the original papers [19], [47]. Furthermore, we replace the original pure random initialization of HGA, which leads to very poor initial solutions, with the heuristic initialization of RoCaSH to remove the biased effect of the initialization on the final performance in the comparison.

In the experiments, all compared algorithms are coded with C++ and run on Intel Core i5-7500 with 3.4 GHz. For each instance, each of the compared algorithms were run 30 times independently.

C. Results and Discussions

First, we examine the performance of RoCaSH on the *mdgdb*, *mdval*, *mdegl* and *mdEGL-G* datasets. These datasets have their problem sizes are from 11 edges to 375 edges. The results are given in Tables II–V. In each table, the columns headed "$|V|$", "$|E|$", "$|T|$" and "Q" indicate the number of vertices, edges, tasks and the capacity of vehicle. For each algorithm, the columns "Average", "Std" and "Time" are the average and standard deviation of the total costs and average runtime (in seconds) over 30 independent runs. The minimal results are marked in bold. In addition, statistical test is carried out between RoCaSH and each other compared algorithm with Wilcoxon rank sum test under the significance level of 0.05. If an algorithm is significantly worse (better) than RoCaSH, then it is marked with "-" ("+"). At the bottom of the table, the "Mean" row provides the mean values of the results over all the instances in the test set, and the "B-C-W" row indicates the number of instances on which RoCaSH performed significantly better than ("B"), statistically comparable with ("C"), and significantly worse than ("W") the corresponding algorithm.

From Table II, one can see that RoCaSH significantly outperformed HGA (MDMA) on 17 (10) out of the 23 *mdgdb* instances. Moreover, RoCaSH obtained the minimal mean value over all the instances in the *mdgdb* set (e.g 251.15 for RoCaSH, 251.43 for MDMA and 259.99 for HGA). It is notable that the runtime of RoCaSH on most instances is below one second. This indicates that RoCaSH can converge much faster than the compared algorithm.

To give a more intuitive comparison between RoCaSH and other compared algorithms, some representative instances are selected from the *mdgdb* set, and the convergence curves of RoCaSH, MDMA and HGA on them are shown in Fig. 2. In the figure, the x-axis is the runtime, and the y-axis is the mean total cost of the best-so-far solutions over the 30 runs. Note that the 30 runs may have different runtime (e.g., one run reached the local optimum very quickly and stopped early with the stopping criterion of 1000 iterations without improvement, while another run reached the 10000 maximal number of iterations). For each algorithm, the length of its convergence curve is decided by the run with the longest runtime. For the shorter runs, we simply use its final total cost to calculate the mean total cost at any time later than its stopping time.

To better distinguish the difference between the algorithms, each curve starts from the total cost obtained after the first iteration, i.e., iteration zero is ignored in the figures, as the algorithms produce almost the same initial solutions.

TABLE II

THE AVERAGE PERFORMANCE OVER 30 INDEPENDENT RUNS OF THE COMPARED ALGORITHMS ON THE *mdgdb* DATASET. FOR EACH INSTANCE, THE MINIMAL MEAN TOTAL COST IS MARKED IN BOLD. UNDER WILCOXON RANK SUM TEST WITH SIGNIFICANCE LEVEL OF 0.05, AN ALGORITHM IS MARKED WITH "-" ("+") IF IT IS SIGNIFICANTLY WORSE (BETTER) THAN ROCASH.

Name	$\|V\|$	$\|E\|$	$\|T\|$	Q	HGA			MDMA			RoCaSH		
					Average	Std	Time (s)	Average	Std	Time (s)	Average	Std	Time (s)
mdgdb1	12	22	22	5	317.17-	5.80	2.68	**304.73**	5.67	3.41	304.97	6.65	<1
mdgdb2	12	26	26	5	338.00-	6.76	2.86	327.60-	4.71	3.37	**325.40**	5.74	<1
mdgdb3	12	22	22	5	272.97-	8.44	2.60	270.53-	6.65	3.08	**265.60**	6.29	<1
mdgdb4	11	19	19	5	291.83	11.58	2.39	**286.67+**	5.67	3.03	288.90	12.64	<1
mdgdb5	13	26	26	5	390.63-	14.34	2.87	376.67-	8.34	4.14	**372.13**	8.73	<1
mdgdb6	12	22	22	5	305.63	9.50	2.63	**294.07+**	8.63	3.02	300.17	7.81	<1
mdgdb7	12	22	22	5	325.80	2.59	2.62	327.13-	3.21	3.58	**325.20**	0.40	<1
mdgdb8	27	46	46	27	375.73-	8.48	4.12	339.13	5.70	5.44	**338.87**	3.69	1.67
mdgdb9	27	51	51	27	324.57-	9.25	3.95	292.67	4.75	5.21	**290.73**	5.00	1.28
mdgdb10	12	25	25	35	295.00-	0.00	2.57	281.27-	4.96	3.55	**275.83**	2.50	<1
mdgdb11	22	45	45	50	406.60	2.39	3.26	**399.43+**	8.26	6.06	406.27	4.52	<1
mdgdb12	13	23	23	10	454.73-	9.80	2.57	**444.80**	13.03	3.17	448.77	10.71	<1
mdgdb13	10	28	28	41	544.70-	4.14	2.83	539.73-	3.86	4.88	**535.33**	4.63	<1
mdgdb14	7	21	21	21	100.40-	1.23	2.57	**98.47**	1.84	3.11	99.20	2.90	<1
mdgdb15	7	21	21	37	60.73-	1.31	2.12	57.20-	0.98	3.30	**56.40**	0.80	<1
mdgdb16	8	28	28	27	126.77-	1.94	2.68	125.27-	0.68	3.45	**125.00**	0.00	<1
mdgdb17	8	28	28	27	92.53-	0.85	2.30	**91.00**	0.00	4.23	**91.00**	0.00	<1
mdgdb18	9	36	36	27	162.10	1.30	2.93	**159.80+**	1.40	4.06	164.00	3.39	<1
mdgdb19	8	11	11	27	55.20	0.60	2.00	**55.00**	0.00	1.29	**55.00**	0.00	<1
mdgdb20	11	22	22	27	127.87-	2.16	2.33	**121.00+**	0.00	1.86	122.87	0.50	<1
mdgdb21	11	33	33	27	163.40-	2.35	3.01	156.80	1.49	1.99	**156.67**	0.94	<1
mdgdb22	11	44	44	27	205.03-	1.05	3.15	200.27-	1.12	3.37	**198.07**	1.06	1.02
mdgdb23	11	55	55	27	242.27-	1.84	4.08	233.67-	1.90	3.74	**230.03**	1.56	1.95
Mean					259.99		2.83	251.43		3.58	251.15		
B-C-W					17-6-0			10-8-5					

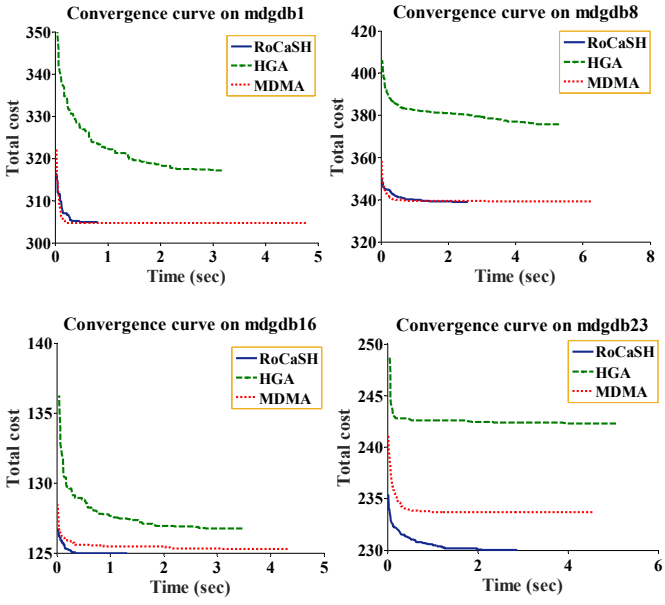

Fig. 2. Convergence curves on some *mdgdb* instances.

From Fig. 2, one can see that the convergence curves of RoCaSH are below those of other two compared algorithms on most instances excluding the mdgdb1. Especially, on mdgdb16 and mdgdb23 instances, there are obvious gaps the convergence curves of RoCaSH and others.

From Table III, it is clear to see that RoCaSH performed much better than HGA on the *mdval* set. It performed slightly poorer than MDMA in terms of the mean values over all 34 instances (i.e. 331.18 for RoCaSH, while 329.53 for MDMA). However, there is no statistical significant difference (13 wins and 13 losses). In addition, the mean runtime for RoCaSH to converge is much less than those of MDMA and HGA (1.06 sec for RoCaSH and 12.37 sec for MDMA, 4.24 for HGA).

Fig. 3 shows the convergence curves of the compared algorithms on some representative *mdval* instances. From the figure, we can see that RoCaSH converged much faster than the other two compared algorithms on three *mdval* instances (mdval1A, mdval8A and mdval10D). It also stopped much earlier than the compared algorithms. These are consistent with the Table III.

From Table IV, we can see that RoCaSH significantly outperformed HGA and MDMA on the most (17 out of 24) *mdegl* instances. For the mean total cost over all 24 instances, RoCaSH reached the minimal value among all the three compared algorithms. RoCaSH was also much less time consuming than that of MDMA. Though HGA had slightly shorter runtime, it showed very poor performance in terms of total cost.

The convergence curves of the compared algorithms on some representative *mdegl* instances are shown in Fig. 4. From the figure, it can be seen that the convergence curves of RoCaSH are below those of the other compared algorithms and the runtime of RoCaSH is also shorter than that of MDMA.

From Table V, we can see RoCaSH outperformed HGA on

TABLE III
THE AVERAGE PERFORMANCE OVER 30 INDEPENDENT RUNS OF THE COMPARED ALGORITHMS ON THE *mdval* DATASET. FOR EACH INSTANCE, THE MINIMAL MEAN TOTAL COST IS MARKED IN BOLD. UNDER WILCOXON RANK SUM TEST WITH SIGNIFICANCE LEVEL OF 0.05, AN ALGORITHM IS MARKED WITH "-" ("+") IF IT IS SIGNIFICANTLY WORSE (BETTER) THAN ROCASH.

Name	$\|V\|$	$\|E\|$	$\|T\|$	Q	HGA			MDMA			RoCaSH		
					Average	Std	Time (s)	Average	Std	Time (s)	Average	Std	Time (s)
mdval1A	24	39	39	200	186.73-	5.05	2.88	183.43-	3.07	9.27	**181.60**	1.43	0.36
mdval1B	24	39	39	120	187.37-	4.76	2.95	183.33-	3.72	10.61	**181.17**	1.86	0.40
mdval1C	24	39	39	45	224.07-	3.81	3.59	203.87-	4.22	10.21	**201.23**	3.49	0.61
mdval2A	24	34	34	180	**220.30**+	3.29	2.90	223.37+	3.82	9.88	230.20	6.19	0.40
mdval2B	24	34	34	120	**222.47**+	4.67	3.10	223.37+	3.57	10.31	232.30	4.91	0.43
mdval2C	24	34	34	40	303.30-	2.98	3.38	304.80-	4.38	10.08	**298.87**	0.72	0.33
mdval3A	24	35	35	80	79.27-	2.06	2.78	81.50-	1.65	9.50	**78.43**	2.60	0.43
mdval3B	24	35	35	50	80.20-	2.91	2.83	81.77-	1.43	9.70	**78.30**	2.08	0.37
mdval3C	24	35	35	20	91.60-	2.22	3.50	89.70-	1.39	10.27	**88.27**	1.24	0.44
mdval4A	41	69	69	225	438.60-	10.98	4.33	**412.27**+	9.96	12.04	423.03	10.46	1.11
mdval4B	41	69	69	170	427.00-	9.53	4.38	**410.23**+	8.55	12.12	419.97	10.73	1.01
mdval4C	41	69	69	130	430.87-	12.70	4.86	**414.73**	11.52	12.29	420.37	11.92	0.97
mdval4D	41	69	69	75	470.00-	4.49	5.20	449.67-	9.14	12.44	**437.43**	11.35	1.02
mdval5A	34	65	65	220	451.03-	10.23	4.54	441.03-	9.30	10.59	**435.63**	8.15	0.63
mdval5B	34	65	65	165	450.23-	9.50	4.52	**439.50**	10.40	11.86	439.67	8.06	0.77
mdval5C	34	65	65	130	458.53-	9.55	4.57	**435.07**	8.54	11.89	437.83	8.25	0.88
mdval5D	34	65	65	75	511.00-	7.29	4.80	480.87-	13.04	12.42	**458.13**	7.28	0.71
mdval6A	31	50	50	170	250.90-	6.95	3.59	**226.23**+	3.68	14.59	239.33	6.62	0.84
mdval6B	31	50	50	120	262.50-	7.56	3.58	**237.43**+	4.65	14.77	245.70	6.22	1.07
mdval6C	31	50	50	50	343.37-	6.24	4.34	**314.30**	4.63	14.19	317.00	5.62	1.39
mdval7A	40	66	66	200	308.60-	8.97	3.76	**287.90**+	6.19	15.05	300.27	7.37	0.96
mdval7B	40	66	66	150	313.60-	9.49	4.17	**290.43**+	7.83	14.82	301.47	10.49	1.22
mdval7C	40	66	66	65	355.00-	1.37	4.96	324.17-	7.22	14.67	**316.47**	6.41	1.68
mdval8A	30	63	63	200	420.77-	10.62	3.85	410.17	9.56	11.39	**406.40**	6.06	0.83
mdval8B	30	63	63	150	425.93-	10.94	4.21	**412.20**	6.95	11.76	413.80	9.26	0.79
mdval8C	30	63	63	65	475.83-	13.58	5.26	442.20-	9.00	12.00	**433.53**	9.23	1.04
mdval9A	50	92	92	235	352.77-	5.99	4.34	**342.23**	5.13	12.94	343.77	7.80	1.67
mdval9B	50	92	92	175	352.20-	5.59	4.98	**338.03**+	5.46	13.49	342.20	6.07	1.60
mdval9C	50	92	92	140	363.70-	4.80	5.32	**338.73**+	5.14	13.91	343.33	6.86	1.32
mdval9D	50	92	92	70	385.37-	6.54	6.28	365.30-	6.89	14.32	**360.50**	6.69	1.63
mdval10A	50	97	97	250	462.70-	7.10	5.52	**443.90**+	5.23	13.59	456.60	10.11	2.38
mdval10B	50	97	97	190	455.83	8.34	4.97	**441.97**+	5.44	13.91	456.17	7.87	1.99
mdval10C	50	97	97	150	455.53	6.58	4.77	**446.67**+	7.11	14.57	460.67	11.26	2.12
mdval10D	50	97	97	75	494.40-	4.10	5.28	483.53	9.10	15.11	**480.33**	10.04	2.75
Maen					344.46		4.24	329.53		12.37	331.18		1.06
B-C-W					30-2-2			13-8-13					

10 *mdEGL-G* instances and MDMA on 8 *mdEGL-G* instances. There is no instance on which RoCaSH obtained worse results. Moreover, RoCaSH obtained much better mean total cost over the 10 instances than the other algorithms. The mean runtime of RoCaSH is slightly longer than that of HGA, but still much shorter than that of MDMA.

Fig. 5 shows the convergence curves of the three compared algorithms on selected representative *mdEGL-G* instances. From the figure, it can be seen that the convergence curves of RoCaSH are below those of both compared algorithms on all the four selected instances. For three instances (i.e. mdEGL-G1-E, mdEGL-G2-C and mdEGL-G2-E), the advantage was shown from the very beginning of the search process.

D. The performance on large scale problems

As the problem size increases, it becomes harder to set the number of iterations for the compared algorithms to lead to similar runtime. Therefore, in the experiments on the large *mdHefei* and *mdBeijing* datasets, we set the same time budget for the compared algorithms to make fair comparison. Specifically, the time budget for an instance is set as $0.2 \times |T|$ seconds, where $|T|$ is the number of the instance. For example, if an instance contains 1210 tasks, the time budget is set to $0.2 \times 1210 = 242$ seconds.

The results are shown in Tables VI and VII. From the tables, we can clearly see that RoCaSH achieved significantly better results than HGA and MDMA on all the *mdHefei* and *mdBeijing* instances.

Figs 6 and 7 show the convergence curves of the three compared algorithms on selected representative *mdHefei* and *mdBeijing* instances. Since the average results obtained by HGA are far worse than RoCaSH, it is difficult to put the curves of three compared algorithms together. Therefore, the figures only consist of two curves of MDMA and RoCaSH for each representative instance. From the figures, one can see that the convergence curves of RoCaSH are obviously below those of MDMA on all the eight selected instances. On some instances (e.g.,. mdBeijing-1, mdBeijing-10), RoCaSH showed the poor ability at beginning of the search process, but it converged very fast and caught up very soon.

TABLE IV
THE AVERAGE PERFORMANCE OVER 30 INDEPENDENT RUNS OF THE COMPARED ALGORITHMS ON THE *mdegl* DATASET. FOR EACH INSTANCE, THE MINIMAL MEAN TOTAL COST IS MARKED IN BOLD. UNDER WILCOXON RANK SUM TEST WITH SIGNIFICANCE LEVEL OF 0.05, AN ALGORITHM IS MARKED WITH "-" ("+") IF IT IS SIGNIFICANTLY WORSE (BETTER) THAN ROCASH.

Name	$\|V\|$	$\|E\|$	$\|T\|$	Q	HGA			MDMA			RoCaSH		
					Average	Std	Time (s)	Average	Std	Time (s)	Average	Std	Time (s)
mdegl-e1-A	77	98	51	305	2839.27-	108.02	4.44	2832.40	17.54	15.18	**2811.23**	68.08	6.31
mdegl-e1-B	77	98	51	220	3306.33-	83.69	4.19	**3120.43+**	50.90	15.21	3172.63	51.66	6.16
mdegl-e1-C	77	98	51	160	3959.50-	58.13	4.55	3724.33-	37.36	15.08	**3693.20**	64.39	9.55
mdegl-e2-A	77	98	72	280	3685.37-	80.51	5.81	3655.40-	46.10	16.42	**3524.40**	57.68	6.65
mdegl-e2-B	77	98	72	200	4363.63-	91.13	6.17	4095.40	70.99	16.34	**4074.60**	75.36	10.61
mdegl-e2-C	77	98	72	140	5137.97-	45.36	5.09	5071.83-	39.67	16.29	**4890.43**	21.65	9.64
mdegl-e3-A	77	98	87	280	4482.17-	78.30	6.61	4249.10-	62.87	18.71	**4033.97**	69.64	12.11
mdegl-e3-B	77	98	87	190	5312.47-	68.68	6.95	5025.67-	84.15	19.33	**4889.77**	34.88	11.49
mdegl-e3-C	77	98	87	135	6428.97-	80.96	7.22	6153.30-	63.03	19.35	**5986.97**	15.99	13.05
mdegl-e4-A	77	98	98	280	4794.33-	50.75	7.28	4662.40-	51.60	21.83	**4521.47**	69.30	12.27
mdegl-e4-B	77	98	98	180	6188.53-	75.31	8.03	5816.70-	52.97	22.79	**5711.17**	60.97	14.56
mdegl-e4-C	77	98	98	130	7245.07-	30.84	7.11	6963.20-	78.54	22.47	**6836.60**	51.25	14.84
mdegl-s1-A	140	190	75	210	3906.60-	91.92	5.81	3723.57-	24.46	16.56	**3669.67**	41.98	8.75
mdegl-s1-B	140	190	75	150	4437.77-	113.43	6.34	4233.33-	25.56	16.52	**4141.67**	46.28	11.00
mdegl-s1-C	140	190	75	103	5492.03-	76.82	5.50	5175.50-	29.99	16.38	**5118.80**	28.12	13.42
mdegl-s2-A	140	190	147	235	7153.57-	126.13	11.53	6431.10	67.30	27.97	**6391.73**	87.68	19.44
mdegl-s2-B	140	190	147	160	8659.73-	140.44	12.09	7768.63-	88.30	27.95	**7725.53**	92.83	20.90
mdegl-s2-C	140	190	147	120	10050.43-	164.94	10.56	9284.53-	101.16	28.42	**9143.10**	107.45	23.62
mdegl-s3-A	140	190	159	240	7538.67-	137.30	13.28	**6590.60**	70.91	29.67	6621.40	118.93	20.82
mdegl-s3-B	140	190	159	160	9025.93-	111.47	10.70	8102.80-	123.14	30.21	**8008.43**	61.96	24.50
mdegl-s3-C	140	190	159	120	10458.97-	130.11	10.86	9640.13-	120.67	30.12	**9516.90**	94.17	24.18
mdegl-s4-A	140	190	190	230	9453.63-	280.58	14.91	8000.77-	138.28	31.61	**7921.53**	87.22	22.48
mdegl-s4-B	140	190	190	160	11164.23-	149.22	13.18	9776.87	138.21	32.51	**9756.23**	76.00	24.95
mdegl-s4-C	140	190	190	120	12980.37-	164.88	12.72	11766.07	155.29	33.06	**11737.13**	75.87	28.76
Mean					6586.06		8.37	6077.67		22.50	5995.77		15.42
B-C-W					24-0-0			17-6-1					

TABLE V
THE AVERAGE PERFORMANCE OVER 30 INDEPENDENT RUNS OF THE COMPARED ALGORITHMS ON THE *mdEGL-G* DATASET. FOR EACH INSTANCE, THE MINIMAL MEAN TOTAL COST IS MARKED IN BOLD. UNDER WILCOXON RANK SUM TEST WITH SIGNIFICANCE LEVEL OF 0.05, AN ALGORITHM IS MARKED WITH "-" ("+") IF IT IS SIGNIFICANTLY WORSE (BETTER) THAN ROCASH.

Name	$\|V\|$	$\|E\|$	$\|T\|$	Q	HGA			MDMA			RoCaSH		
					Average	Std	Time (s)	Average	Std	Time (s)	Average	Std	Time (s)
mdEGL-G1-A	255	375	347	28600	1013300.60-	10598.24	19.22	862162.17	12922.65	36.60	**856936.10**	13843.36	26.21
mdEGL-G1-B	255	375	347	22800	1031053.63-	8599.66	16.82	911432.97-	13429.38	38.22	**899692.83**	14904.57	32.76
mdEGL-G1-C	255	375	347	19000	1110681.67-	10082.94	22.31	973987.10-	15130.05	41.21	**940262.13**	10970.78	40.60
mdEGL-G1-D	255	375	347	16200	1196695.33-	7702.92	20.52	1021681.67-	14620.42	43.44	**994569.43**	12872.59	43.83
mdEGL-G1-E	255	375	347	14100	1251558.87-	6947.65	20.17	1089379.57-	22695.62	43.25	**1069231.10**	11383.86	44.50
mdEGL-G2-A	255	375	375	28000	1078531.93-	8772.57	16.74	928218.00	13292.86	41.94	**921700.10**	14566.88	28.65
mdEGL-G2-B	255	375	375	23100	1130622.33-	6482.25	19.43	974024.00-	13643.74	42.57	**956795.17**	11448.57	34.23
mdEGL-G2-C	255	375	375	19400	1172994.30-	10274.47	21.15	1029920.87-	15967.06	45.67	**1004728.10**	11516.97	42.67
mdEGL-G2-D	255	375	375	16700	1250813.77-	9018.68	20.97	1083767.30-	16528.80	45.69	**1065950.20**	12999.64	42.39
mdEGL-G2-E	255	375	375	14700	1346460.30-	6017.33	22.70	1153119.63-	15421.05	46.81	**1127900.27**	13006.33	48.98
Mean					1158248.91		20.00	1002769.33		42.54	983776.54		38.48
B-C-W					10-0-0			8-2-0					

In summary, from the analysis of the results on different datasets with a wide range of problem sizes (i.e. *mdgdb*, *mdval*, *mdegl*, *mdEGL-G*, *mdHefei* and *mdBeijing*), it can be seen that RoCaSH can tackle MDCARP well, especially for the LSMDCARP. From Figs 2–5, it seems that as the problem size grows, the computational time of RoCaSH increases rapidly when compared with HGA and MDMA. However, for the large instances, the curve of RoCaSH is almost always below that of HGA and MDMA after a few iterations. This suggests that for the large and complex instances, RoCaSH is less likely to be stuck into poor local optima, and can use the allowed computational budget more effectively than HGA and MDMA. From Figs 6 and 7, we can see that RoCaSH has a clear superiority on the large MDCARP test instances.

TABLE VI
THE AVERAGE PERFORMANCE OVER 30 INDEPENDENT RUNS OF THE COMPARED ALGORITHMS ON THE *mdHefei* DATASET. FOR EACH INSTANCE, THE MINIMAL MEAN TOTAL COST IS MARKED IN BOLD. UNDER WILCOXON RANK SUM TEST WITH SIGNIFICANCE LEVEL OF 0.05, AN ALGORITHM IS MARKED WITH "-" ("+") IF IT IS SIGNIFICANTLY WORSE (BETTER) THAN ROCASH.

Name	$\|V\|$	$\|E\|$	$\|T\|$	Q	HGA		MDMA		RoCaSH	
					Average	Std	Average	Std	Average	Std
mdHefei-1	850	1212	121	9000	203948.07-	3131.22	213924.57-	1430.23	**197212.77**	930.41
mdHefei-2	850	1212	242	9000	367128.23-	6932.00	361173.17-	3122.22	**343753.17**	3055.02
mdHefei-3	850	1212	364	9000	494900.50-	10172.96	476446.03-	3945.95	**453233.90**	2191.37
mdHefei-4	850	1212	485	9000	655931.27-	15893.29	600753.8-	6830.45	**581350.40**	3110.64
mdHefei-5	850	1212	606	9000	828839.77-	17028.25	725769.93-	7536.15	**700613.17**	3992.64
mdHefei-6	850	1212	727	9000	992164.40-	25607.82	856229.00-	10351.70	**816784.20**	5483.50
mdHefei-7	850	1212	848	9000	1161062.80-	24107.72	991237.90-	8885.14	**957752.03**	5101.69
mdHefei-8	850	1212	970	9000	1334204.30-	28438.79	1111462.90-	10623.02	**1069352.30**	6412.04
mdHefei-9	850	1212	1091	9000	1499142.77-	29525.45	1233504.93-	11079.67	**1191800.50**	7315.59
mdHefei-10	850	1212	1212	9000	1629957.80-	30778.06	1344466.73-	11601.35	**1295919.43**	8238.76
Mean					916727.99		791496.90		760777.19	
B-C-W					10-0-0		10-0-0			

TABLE VII
THE AVERAGE PERFORMANCE OVER 30 INDEPENDENT RUNS OF THE COMPARED ALGORITHMS ON THE *mdBeijing* DATASET. FOR EACH INSTANCE, THE MINIMAL MEAN TOTAL COST IS MARKED IN BOLD. UNDER WILCOXON RANK SUM TEST WITH SIGNIFICANCE LEVEL OF 0.05, AN ALGORITHM IS MARKED WITH "-" ("+") IF IT IS SIGNIFICANTLY WORSE (BETTER) THAN ROCASH.

Name	$\|V\|$	$\|E\|$	$\|T\|$	Q	HGA		MDMA		RoCaSH	
					Average	Std	Average	Std	Average	Std
mdBeijing-1	2820	3584	358	25000	719234.97-	11886.66	686975.27-	2276.83	**670299.77**	2498.79
mdBeijing-2	2820	3584	717	25000	1150250.27-	24800.00	991639.77-	5983.89	**969526.90**	3987.90
mdBeijing-3	2820	3584	1075	25000	1617555.9-	52678.27	1291706.77-	7998.94	**1262995.90**	5536.61
mdBeijing-4	2820	3584	1434	25000	2094669.57-	57565.46	1532919.13-	8790.30	**1501499.47**	6308.62
mdBeijing-5	2820	3584	1792	25000	2509434.83-	84950.27	1765292.37-	12919.29	**1730308.47**	8791.86
mdBeijing-6	2820	3584	2151	25000	3015946.30-	77773.91	2037539.30-	12574.15	**2006983.97**	10079.70
mdBeijing-7	2820	3584	2509	25000	3478082.53-	84965.25	2241324.77-	16682.94	**2198449.20**	12032.55
mdBeijing-8	2820	3584	2868	25000	3870752.67-	114643.07	2411349.87-	15910.58	**2376049.93**	12630.53
mdBeijing-9	2820	3584	3226	25000	4359410.80-	137537.60	2660257.63-	16518.70	**2614046.63**	11341.64
mdBeijing-10	2820	3584	3584	25000	4769283.93-	144797.72	2856373.80-	16984.42	**2800199.50**	12662.83
Mean					2758462.18		1847537.87		1813035.97	
B-C-W					10-0-0		10-0-0			

E. Scalability

To investigate the scalability of RoCaSH, we give the scatter plot of the ratio between the performance of RoCaSH and other compared algorithms (i.e., HGA, MDMA) versus the number of tasks on the test instances. The plot is shown in Fig. 8, where the x-axis is the number of tasks in the instance, and the y-axis is the ratio between the mean total costs of RoCaSH and other two compared algorithms. For example, on mdegl-s4-C, the mean total costs of HGA, MDMA and RoCaSH are 12980.37, 11766.07 and 11737.13, thus the ratios of RoCaSH versus HGA and MDMA are $11737.13/12980.37 = 0.904$ and $11737.13/11766.07 = 0.998$.

From Fig. 8, it can be seen that for large instances (e.g., more than 1000 tasks), the ratio values are smaller than 1, indicating that RoCaSH performed better than HGA and MDMA. HGA or MDMA performed slightly better than RoCaSH (the ratios are between 1 and 1.05) only on some small instances (problem size smaller than 200). This verifies the scalability of RoCaSH on LSMDCARP instances.

V. CONCLUSIONS AND FUTURE WORK

The large scale multi-depot CARP (LSMDCARP) is a challenging problem due to its large solution space. This paper solves LSMDCARP based on a divide-and-conquer strategy, and proposed a Route Clustering and Search Heuristic (RoCaSH). Specifically, to address the challenge of allocating the tasks among the depots. We developed novel route cutting off and clustering schemes in RoCaSH. Then, for the tasks associated to each depot, we employ the Ulusoy's splitting procedure combined with the local search for solving single-depot LSCARP. The route clustering and search components interact with each other to solve the problem effectively. The experimental studies on a wide range of MDCARP instances demonstrate that the proposed RoCaSH significantly outperformed the state-of-the-art approaches in terms of both effectiveness and efficiency.

In the future, we will further investigate the interactions between tasks and routes under the problem constraints (e.g. the capacity constraints), and propose more effective problem

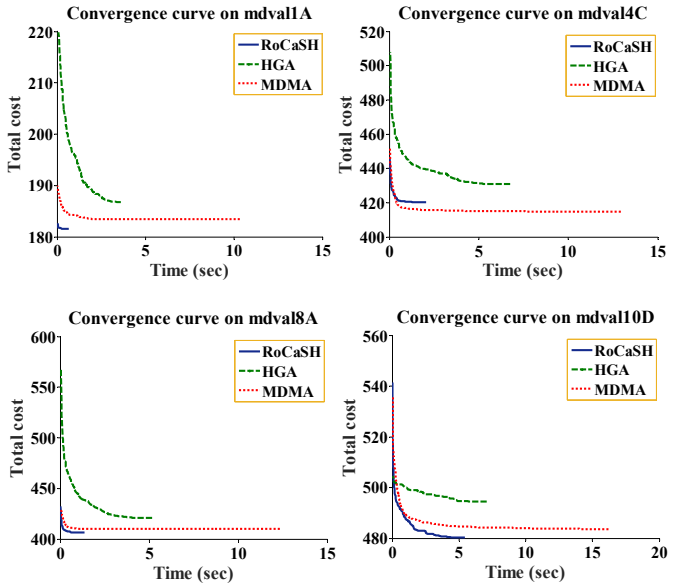

Fig. 3. Convergence curves on some *mdval* instances.

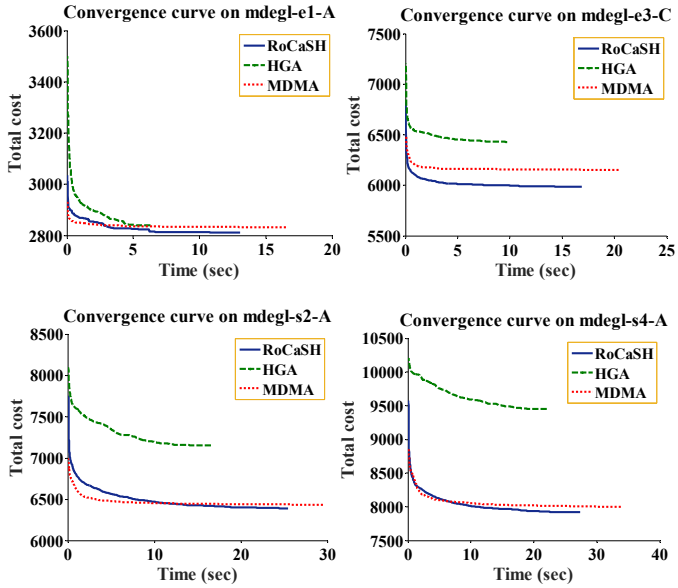

Fig. 4. Convergence curves on the *mdegl* instances.

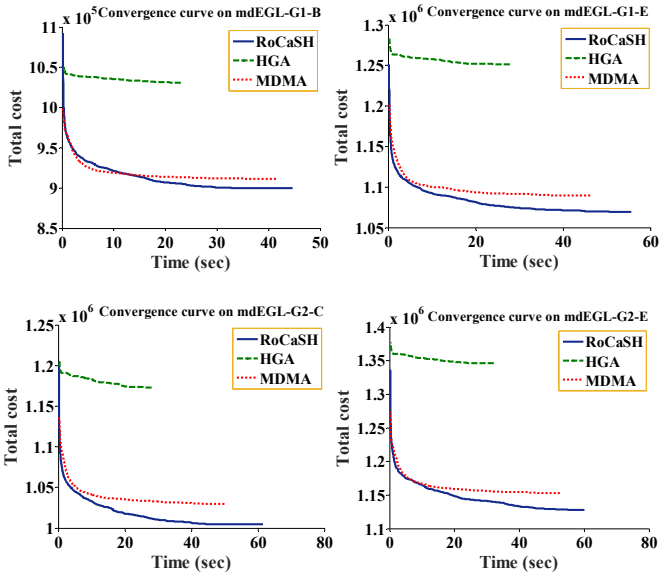

Fig. 5. Convergence curves on the *mdEGL–G* instances.

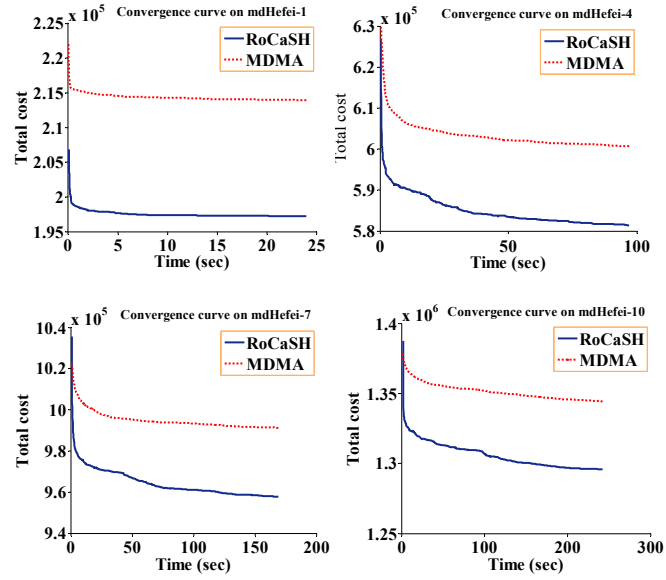

Fig. 6. Convergence curves on the *mdHefei* instances.

decomposition methods. For example, we can focus more on the tasks and routes that are on the boundaries of multiple depots, i.e. which have similar distances to multiple depots, and re-cluster the routes based on the membership of the routes to different depots. We will also consider bi-level decomposition schemes that employ the divide-and-conquer strategy not only for allocating tasks/routes between different depots, but also for decomposing the sub-problems for each single depot (as known as large scale single-depot CARP [22]).

ACKNOWLEDGMENT

This work was supported by Anhui Provincial Natural Science Foundation (No. 1808085MF173) and Natural Science Key Research Project for Higher Education Institutions of Anhui Province(Nos. KJ2019A0554, KJ2016A438).

REFERENCES

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

[2] H. Handa, L. Chapman, and X. Yao, "Robust route optimization for gritting/salting trucks: a CERCIA experience," *IEEE Computational Intelligence Magazine*, vol. 1, no. 1, pp. 6–9, 2006.

[3] H. Handa, D. Lin, L. Chapman, and X. Yao, "Robust solution of salting route optimisation using evolutionary algorithms," in *IEEE Congr. Evol. Comput. 2006*. Vancouver, BC, Canada, 2006, pp. 3098–3105.

[4] W.-L. Pearn, A. Assad, and B. L. Golden, "Transforming arc routing into node routing problems," *Computers & operations research*, vol. 14, no. 4, pp. 285–288, 1987.

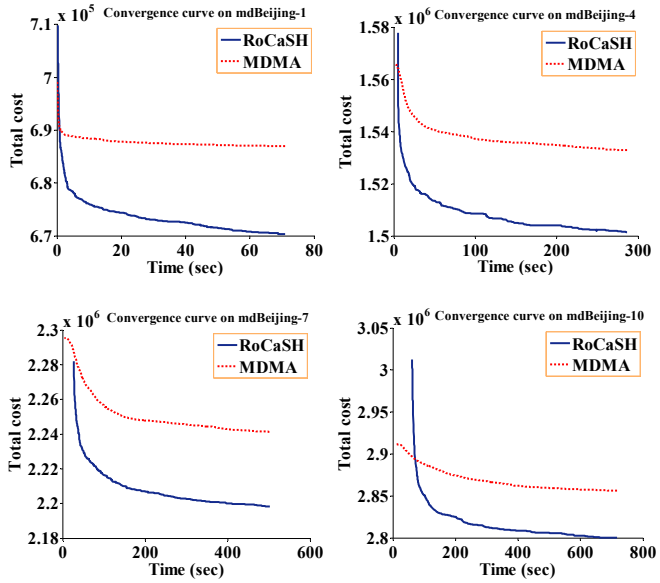

Fig. 7. Convergence curves on the *mdBeijing* instances.

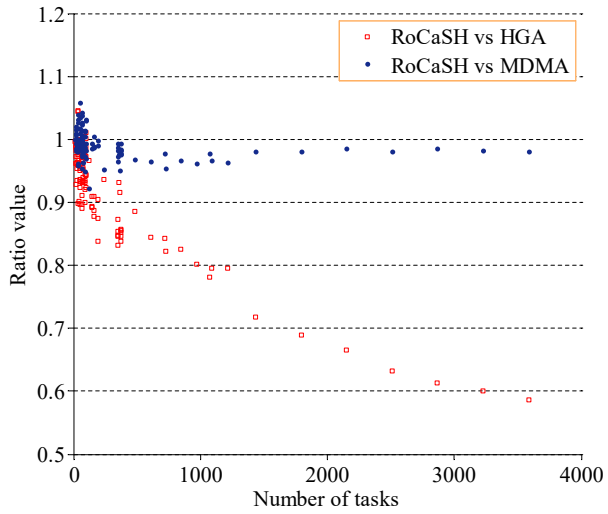

Fig. 8. The ratio between the mean total costs of RoCaSH and other compared algorithms versus the number of tasks on four datasets.

[5] F. Chu, N. Labadi, and P. C., "The Periodic Capacitated Arc Routing Problem linear programming model, metaheuristic and lower bounds," *Journal of Systems Science and Systems Engineering*, vol. 13, no. 4, pp. 423–435, 2004.

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

[7] J. Campbell and A. Langevin, "Roadway snow and ice control," *Arc routing: theory, solutions and applications. Boston, MA: Kluwer*, pp. 389–418, 2000.

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

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

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

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

[12] K. Tang, Y. Mei, and X. Yao, "Memetic algorithm with extended neighborhood search for capacitated arc routing problems," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 5, pp. 1151–1166, 2009.

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

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

[15] ——, "A global repair operator for capacitated arc routing problem," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 39, no. 3, pp. 723–734, 2009.

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

[17] C. Schlebusch and S. Irnich, "Cut-first branch-and-price-second for the capacitated arc-routing problem," *Operations research*, vol. 60, no. 5, pp. 1167–1182 ., 2012.

[18] A. Amberg, W. Domschke, and S. Voß, "Multiple center capacitated arc routing problems: A tabu search algorithm using capacitated trees," *European Journal of Operational Research*, vol. 124, no. 2, pp. 360–376, 2000.

[19] A. Kansou and Y. A., "New upper bounds for the multi-depot capacitated arc routing problem," *International Journal of Metaheuristics*, vol. 1, no. 1, pp. 81–95, 2010.

[20] R. Martinelli, M. Poggi, and A. Subramanian, "Improved bounds for large scale capacitated arc routing problem," *Computers & Operations Research*, vol. 40, no. 8, pp. 2145–2160, 2013.

[21] Y. Mei, X. Li, and X. Yao, "Decomposing large-scale capacitated arc routing problems using a random route grouping method," in *Proceedings of the 2013 IEEE Congress on Evolutionary Computation*, 2013, pp. 1013–1020.

[22] ——, "Cooperative coevolution with route distance grouping for large-scale capacitated arc routing problems," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 3, pp. 435–449, 2014.

[23] ——, "Variable neighborhood decomposition for large scale capacitated arc routing problem," in *Proceedings of the 2014 IEEE Congress on Evolutionary Computation*, 2014, pp. 1313–1320.

[24] R. Shang, K. Dai, L. Jiao, and et al., "Improved memetic algorithm based on route distance grouping for multiobjective large scale capacitated arc routing problems," *IEEE Transactions on Cybernetics*, vol. 46, no. 4, pp. 1000–1013, 2016.

[25] R. Shang, B. Du, K. Dai, L. Jiao, and et al., "Quantum–inspired immune clonal algorithm for solving large–scale capacitated arc routing problems," *Memetic Computing*, pp. 1–22, 2017.

[26] R. Shang, B. Du, K. Dai, and et al., "Memetic algorithm based on extension step and statistical filtering for large–scale capacitated arc routing problems," *Natural Computing*, pp. 1–17, 2017.

[27] K. Tang, J. Wang, X. Li, and X. Yao, "A scalable approach to capacitated arc routing problems based on hierarchical decomposition," *IEEE Transactions on Cybernetics*, vol. 47, no. 11, pp. 3928–3940, 2017.

[28] L. Ke, Q. Zhang, and R. Battiti, "MOEA/D-ACO: a multiobjective evolutionary algorithm using decomposition and AntColony," *IEEE Transactions on Cybernetics*, vol. 43, no. 6, pp. 1845–1859, 2013.

[29] M. Wu, K. Li, S. Kwong, and et al, "Evolutionary many-objective optimization based on adversarial decomposition," *IEEE Transactions on Cybernetics*, pp. 1–12, 2018.

[30] C. Goh and K. Tan, "A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 1, pp. 103–127, 2009.

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

[32] K. Tan, Y. Yang, and C. Goh, "A distributed cooperative coevolutionary algorithm for multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 5, pp. 527–549, 2006.

[33] F. Oliveira, R. Enayatifar, H. Sadaei, and J. Potvin, "A cooperative coevolutionary algorithm for the multi-depot vehicle routing problem," *Expert Systems with Applications*, vol. 43, no. C, pp. 117–130, 2016.

[34] A. Ostertag, K. Doerner, R. Hartl, E. Taillard, and P. Waelti, "Popmusic for a real-world large-scale vehicle routing problem with time windows," *Journal of the Operational Research Society*, vol. 60, no. 7, pp. 934–943, 2009.

[35] J. Gu, M. Gu, C. Cao, and X. Gu, "A novel competitive co-evolutionary quantum genetic algorithm for stochastic job shop scheduling problem," *Computers & Operations Research*, vol. 37, no. 5, pp. 927–937, 2010.

[36] S. Nguyen, M. Zhang, M. Johnston, and K. Tan, "Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 2, pp. 193–208, 2014.

[37] X. Peng, Y. Jin, and H. Wang, "Multimodal optimization enhanced cooperative coevolution for large-scale optimization," *IEEE Transactions on Cybernetics*, pp. 1–14, 2018.

[38] Z. Ren, Y. Liang, A. Zhang, and et al, "Boosting cooperative coevolution for large scale optimization with a fine-grained computation resource allocation strategy," *IEEE Transactions on Cybernetics*, pp. 1–14, 2018.

[39] M. Tang and X. Yao, "A memetic algorithm for VLSI floorplanning," *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, vol. 37, no. 1, pp. 62–69, 2007.

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

[41] J. Belenguer and E. Benavent, "A cutting plane algorithm for the capacitated arc routing problem," *Computers and Operations Research*, vol. 30, no. 5, pp. 705–728, 2003.

[42] 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, 2010.

[43] E. Bartolini, J.-F. Cordeau, and G. Laporte, "Improved lower bounds and exact algorithm for the capacitated arc routing problem," *Mathematical Programming Series B*, vol. 137, no. 1–2, pp. 409–452, 2013.

[44] M. Mourão and L. Amado, "Heuristic method for a mixed capacitated arc routing problem: A refuse collection application," *European Journal of Operational Research*, vol. 160, no. 1, pp. 139–153, 2005.

[45] P. Lacomme, C. Prins, and M. Sevaux, "A genetic algorithm for a bi-objective capacitated arc routing problem," *Computers and Operations Research*, vol. 33, no. 12, pp. 3473–3493, 2006.

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

[47] Z. Zhu, X. Li, Y. Yong, and et al, "A hybrid genetic algorithm for the multiple depot capacitated arc routing problem," in *IEEE International Conference on Automation & Logistics*, 2007, pp. 2253–2258.

[48] T. Liu, Z. Jiang, and G. Na, "A genetic local search algorithm for the multi-depot heterogeneous fleet capacitated arc routing problem," *Flexible Services & Manufacturing Journal*, vol. 26, no. 4, pp. 540–564, 2014.

[49] D. Krushinsky and T. V. Woensel, "An approach to the asymmetric multi-depot capacitated arc routing problem," *European Journal of Operational Research*, vol. 244, no. 1, pp. 100–109, 2015.

[50] L. Xing, P. Rohlfshagen, Y. Chen, and X. Yao, "An evolutionary approach to the multidepot capacitated arc routing problem," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 3, pp. 356–374, 2010.

[51] B. Cao, R. Li, and C. S., "A modified memetic algorithm for multi-depot green capacitated arc routing problem," *BIC-TA (1)*, pp. 739–750.

[52] B. Kanso, "Hybrid ant colony algorithm for the multi-depot periodic open capacitated arc routing problem," *International Journal of Artificial Intelligence & Applications*, vol. 11, no. 1, pp. 53–64, 2020.

[53] Y. Mei, M. Omidvar, X. Li, and X. Yao, "A competitive divide-and-conquer algorithm for unconstrained large-scale black-box optimization," *Acm Transactions on Mathematical Software*, vol. 42, no. 2, pp. 1–24, 2016.

[54] M. Omidvar, M. Yang, Y. Mei, X. Li, and X. Yao, "DG2: A faster and more accurate differential grouping for large-scale black-box optimization," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 6, pp. 929–942, 2017.

[55] A. Iorio and X. Li, "A cooperative coevolutionary multiobjective algorithm using non-dominated sorting," in *Genetic and Evolutionary Computation - GECCO, Seattle, Wa, Usa, June 26-30, 2004, Proceedings*, 2004, pp. 537–548.

[56] R. Zhang and C. Wu, "A divide-and-conquer strategy with particle swarm optimization for the job shop scheduling problem," *Engineering Optimization*, vol. 42, no. 7, pp. 641–670, 2010.

[57] C. Chan, H. Gooi, and M. Lim, "Co-evolutionary algorithm approach to a university timetable system," in *Proceedings of the 2002 Congress on Evolutionary Computation*, 2002, pp. 1946–1951.

[58] D. Arenas, R. Chevrier, J. Rodriguez, and C. Dhaenens, "Application of a co-evolutionary genetic algorithm to solve the periodic railway timetabling problem," in *International Conference on Industrial Engineering and Systems Management*, 2013, pp. 1–7.

[59] G. Dantzig and J. Ramser, "The truck dispatching problem," *Management Science*, vol. 6, no. 1, pp. 80–91, 1959.

[60] A. Lim and F. Wang, "Multi-depot vehicle routing problem: A one-stage approach," *IEEE Transactions on Automation Science and Engineering*, vol. 2, no. 4, pp. 397–402, 2005.

[61] E. Lalla-Ruiz and S. Voß, "A popmusic approach for the multi-depot cumulative capacitated vehicle routing problem," *Optimization Letters*, pp. 1–21, 2019.

[62] P. Stodola, "Hybrid ant colony optimization algorithm applied to the multi-depot vehicle routing problem," *Natural Computing*, pp. 1–13, 2020.

[63] J. Wang, T. Weng, and Q. Zhang, "A two-stage multiobjective evolutionary algorithm for multiobjective multidepot vehicle routing problem with time windows," *IEEE Transactions on Cybernetics*, vol. 49, no. 7, pp. 2467–2478, 2019.

[64] U. Michiel, H. Albert, B. J., and et al, "Asymmetric multi-depot vehicle routing problems: valid inequalities and a branch-and-cut algorithm," *Operations Research*, pp. 1–42, 2020.

[65] J. Brãndo, "A memory-based iterated local search algorithm for the multi-depot open vehicle routing problem," *European Journal of Operational Research*, vol. 284, pp. 559–571, 2020.

[66] M. Qi, W.-H. Lin, N. Li, and L. Miao, "A spatiotemporal partitioning approach for large-scale vehicle routing problems with time windows," *Transportation Research Part E: Logistics and Transportation Review*, vol. 48, no. 1, pp. 248–257, 2012.

[67] W. Tu, Q. Li, Q. Li, J. Zhu, B. Zhou, and B. Chen, "A spatial parallel heuristic approach for solving very large-scale vehicle routing problems," *Transactions in Gis*, vol. 21, no. 6, pp. 1130–1147, 2017.

[68] J. Xiao, T. Zhang, J. Du, and X. Zhang, "An evolutionary multiobjective route grouping-based heuristic algorithm for large-scale capacitated vehicle routing problems," *IEEE Transactions on Cybernetics*, pp. 1–14, 2019.

[69] Y. Zhang and Y. Mei, "Divide-and-conquer large scale capacitated arc routing problems with route cutting off decomposition," *arXiv e-prints*, p. arXiv:1912.12667, Dec 2019.

[70] I. D. Giosa, I. L. Tansini, and I. O. Viera, "New assignment algorithms for the multi-depot vehicle routing problem," *The Journal of the Operational Research Society*, vol. 53, no. 9, pp. 977–984, 2002.