

A Scalable Approach to Capacitated Arc Routing Problems Based on Hierarchical Decomposition

Ke Tang, *Senior Member, IEEE*, Juan Wang, Xiaodong Li, *Senior Member, IEEE*,
and Xin Yao, *Fellow, IEEE*

Abstract—The capacitated arc routing problem (CARP) is a challenging optimization problem with lots of applications in the real world. Numerous approaches have been proposed to tackle this problem. Most of these methods, albeit showing good performance on CARP instances of small and median sizes, do not scale well to large-scale CARPs, e.g., taking at least a few hours to achieve a satisfactory solution on a CARP instance with thousands of tasks. In this paper, an efficient and scalable approach is proposed for CARPs. The key idea of the proposed approach is to hierarchically decompose the tasks involved in a CARP instance into subgroups and solve the induced subproblems recursively. The output of the subproblems at the lower layer in the hierarchy is treated as virtual tasks and new subproblems are formulated based on these virtual tasks using clustering techniques. By this means, the number of tasks (or virtual tasks) decreases rapidly from the bottom to the top layers of the hierarchy, and the sizes of all subproblems at each layer can be kept tractable even for very large-scale CARPs. Empirical studies are conducted on CARP instances with up to 3584 tasks, which are an order of magnitude larger than the number of tasks involved in all CARP instances investigated in the literature. The results show that the proposed approach significantly outperforms existing methods in terms of scalability. Since the proposed hierarchical decomposition scheme is designed to obtain a good permutation of tasks in a CARP instance, it may also be generalized to other hard optimization problems that can be formulated as permutation-based optimization problems.

Index Terms—Capacitated arc routing problem (CARP), clustering, combinatorial optimization, hierarchical decomposition (HD), scalability.

Manuscript received November 29, 2015; revised March 26, 2016 and June 5, 2016; accepted July 1, 2016. Date of publication August 4, 2016; date of current version October 13, 2017. This work was supported in part by the National Natural Science Foundation of China under Grant 61329302, in part by the Program for New Century Excellent Talents in University under Grant NCET-12-0512, in part by the EPSRC under Grant EP/K001523/1, in part by the Royal Society Newton Advanced Fellowship under Grant NA150123, and in part by the ARC Discovery under Grant DP120102205. The work of X. Yao was supported by the Royal Society Wolfson Research Merit Award. This paper was recommended by Associate Editor H. Ishibuchi.

K. Tang and J. Wang are with the USTC-Birmingham Joint Research Institute in Intelligent Computation and Its Applications, School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China (e-mail: ketang@ustc.edu.cn; jingze@mail.ustc.edu.cn).

X. Li is with the Evolutionary Computation and Machine Learning Research Group, School of Computer Science and Information Technology, RMIT University, Melbourne, VIC 3000, Australia (e-mail: xiaodong.li@rmit.edu.au).

X. Yao is with the Center of Excellence for Research in Computational Intelligence and Applications, School of Computer Science, University of Birmingham, Birmingham B15 2TT, U.K. (e-mail: x.yao@cs.bham.ac.uk).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TCYB.2016.2590558

I. INTRODUCTION

THE CAPACITATED arc routing problem (CARP) [1] is a classical combinatorial optimization problem that seeks an optimal set of routes to cover a certain subset of edges and/or arcs in a given network subject to some specific constraints, where each edge typically stands for a road in the real world [2], [3]. For its wide range of practical applications, such as winter gritting [4], urban waste collection [5], [6] and snow removal [7], [8], CARP has drawn considerable attentions in the past few decades and a large number of algorithms have been proposed [2], [7], [9]–[16]. However, previous investigations are mostly limited to relatively small scale CARPs. The largest CARP instance that has been used in the literature, the EGL-G benchmark set [17], consists of up to 375 edges and 375 tasks. In contrast, with the ever growing of big cities, a real-world CARP might involve much more roads and tasks. For example, the central area of Beijing, China, consists of more than 3000 main roads. Hence, it is natural to ask whether existing approaches can still tackle such large-scale CARP instances satisfactorily.

In spite of its importance, the scalability issue of CARP solvers has been rarely addressed in the literature. Prior to 2008, most algorithms for CARP were only tested on small and medium-scale CARP instances, e.g., the gdb [18], val [19], and Beullens' benchmark sets [14], for most of which the optimal solutions can be found by exact methods. The above-mentioned EGL-G instances were proposed in 2008 and widely used as an additional test set since then. Results obtained on these instances show that the performance of existing approaches clearly deteriorates with the increasing size of CARP instance, both in terms of solution quality (no optimal solution can be found for any EGL-G instance) and in terms of computational cost (less than 10 s for a small-scale val instance but about 30 min for an EGL-G instance) [2], [20], [21].

Motivated by the above observation, Mei *et al.* [22]–[24] proposed several approaches to tackle large-scale CARPs. These methods share a similar iterative search framework called cooperative co-evolution (CC) [25]–[28]. That is, a CARP instance is decomposed into a set of subproblems through dividing its tasks into groups. The subproblems are tackled separately. The obtained partial-solutions are combined into a complete solution to the original CARP instance and evaluated. The best-so-far complete solution is used to reset the decomposition in the next iteration. In these approaches, decomposition (i.e., grouping tasks) is conducted either randomly [22] or

based on a predefined route distance matrix [23], [24], and different optimization techniques can be adopted to solve the subproblems. These CC-based approaches, e.g., the route distance grouping scheme with memetic algorithm with extended neighborhood search (RDG-MAENS) [23], perform significantly better than previous approaches on EGL-G instances. Such advantages should mainly be credited to solving the problem in a divide-and-conquer manner. Nevertheless, these CC-based methods all decompose CARP in a linear way. That is, to keep the subproblems at a tractable size, the number of subproblems needs to increase linearly with the number of tasks. In this case, the complexity of appropriate task groupings increases rapidly with the size of CARP. Consequently, it is more likely that inappropriate groupings of tasks will be obtained on large-scale CARPs and mis-guide the search. Although Shang *et al.* [29] proposed to improve the CC-based RDG-MAENS [23], the improved RDG-MAENS was not tested on the existing largest CARP instances (i.e., the egl-large instances with up to 375 tasks). Thus, the CC-based methods may still encounter scalability issues on large-scale CARPs.

This paper aims to develop a novel approach that can scale well to large-scale CARPs. Specifically, we are interested in CARPs that are at least an order of magnitude larger than the existing benchmark instances, and aim to develop methods for achieving good solutions to such CARPs within acceptable time, e.g., less than half an hour. A scalable approach based on hierarchical decomposition (SAHiD) is proposed for this purpose. The key idea of SAHiD is to hierarchically (rather than linearly) decompose the tasks involved in a CARP instance into subgroups. At the bottom layer of the hierarchy, tasks are decomposed into a few groups and a subproblem is solved for each group. At each intermediate layer of the hierarchy, the output of the subproblems at the lower layer is treated as virtual tasks and new subproblems are formulated based on these virtual tasks rather than the original tasks. The final solution is obtained at the top layer of the hierarchy. With such a hierarchical structure, the number of tasks (or virtual tasks) exponentially decreases from the bottom to the top layers of the hierarchy. Thus, the number of task groups required at each layer, except for the bottom layer, is significantly less than that required for linear decomposition schemes. As a result, the complexity of grouping tasks can be better controlled and inappropriate groupings are less likely to be obtained. Furthermore, as will be shown by our empirical studies, a solution to large-scale CARPs can be obtained efficiently in such a hierarchical way, thus allowing repeating the process in an iterative manner to achieve better solution quality than existing methods in shorter runtime.

The rest of this paper is organized as follows. First, the problem definition and notations of CARP are introduced in Section II. After that, the hierarchical decomposition (HD) scheme and detailed steps of SAHiD are described in Sections III and IV, respectively. Empirical studies are presented in Section V to assess the performance of SAHiD and compare it against state-of-the-art CARP solvers. Finally, Section VI concludes this paper and discusses directions for future research.

II. PROBLEM DEFINITION AND NOTATIONS

An undirected/directed CARP is defined on a connected and undirected/directed graph $G(V, E)$, where V and E represent the sets of vertices and edges, respectively. A cost $c(e) > 0$ and a demand $d(e) \geq 0$ are associated with each edge $e \in E$. The edges with positive demands constitute the task set T , i.e., $T = \{\tau \in E | d(\tau) > 0\}$. A vertex $v_0 \in V$ is predefined as the depot, in which a fleet of vehicles are located. The aim of CARP is to determine a set of routes for the vehicles to serve all the tasks with minimal total costs, subject to the following constraints.

- 1) Each route must start and end at the depot.
- 2) Each task is served exactly once (but the corresponding edge can be traversed more than once).
- 3) The total demand of tasks served in each route cannot exceed the vehicle capacity Q .

A solution of CARP can be represented by a sequence of vertices, which directly indicates the order of vertices for the vehicles to visit. However, given a sequence of the tasks, the minimum cost can be easily achieved by summing up the shortest paths between the vertices of each pair of consecutive tasks in the sequence in polynomial time [15]. Since the task representation is more compact, it is adopted in this paper. That is a solution to CARP is represented by $s = (R_1, 0, R_2, \dots, R_m)$, where m is the number of routes, the k th route $R_k = (\tau_{k,1}, \dots, \tau_{k,l_k})$, and 0 stands for a dummy task that separates two routes. For each R_k , $\tau_{k,t}$, and l_k denote the t th task and the number of tasks served in route R_k , respectively. More formally, let $u_1(\tau)$ and $u_2(\tau)$ represent the endpoints of task τ , and $\text{inv}(\tau)$ the inverse direction of τ , i.e., $u_1(\text{inv}(\tau)) = u_2(\tau)$ and $u_2(\text{inv}(\tau)) = u_1(\tau)$, a CARP can be formulated as follows:

$$\text{minimize } \text{TC}(s) \quad (1)$$

$$\text{s.t. } \sum_{k=1}^m l_k = |T| \quad (2)$$

$$\tau_{k_1, i_1} \neq \tau_{k_2, i_2}, \quad \forall (k_1, i_1) \neq (k_2, i_2) \quad (3)$$

$$\tau_{k_1, i_1} \neq \text{inv}(\tau_{k_2, i_2}), \quad \forall (k_1, i_1) \neq (k_2, i_2) \quad (4)$$

$$\sum_{i=1}^{l_k} d(\tau_{k,i}) \leq Q, \quad \forall k = 1, 2, \dots, m. \quad (5)$$

The objective function, i.e., (1), requires minimizing the total cost $\text{TC}(s)$

$$\text{TC}(s) = \sum_{k=1}^m \text{RC}(R_k). \quad (6)$$

$\text{RC}(R_k)$ is the total cost of route R_k and can be computed using

$$\begin{aligned} \text{RC}(R_k) = & \sum_{i=1}^{l_k} c(\tau_{k,i}) + \text{dc}(v_0, u_1(\tau_{k,1})) \\ & + \sum_{i=2}^{l_k} \text{dc}(u_2(\tau_{k,i-1}), u_1(\tau_{k,i})) + \text{dc}(u_2(\tau_{k,l_k}), v_0) \end{aligned} \quad (7)$$

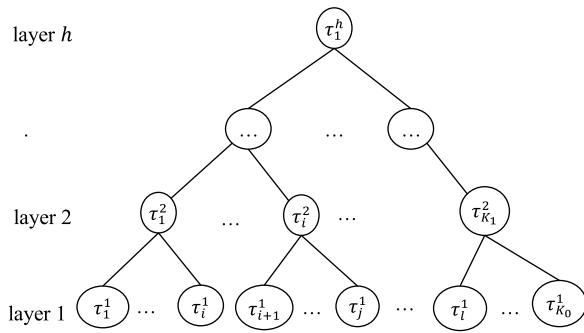


Fig. 1. Hierarchical structure of HD.

where $dc(v_i, v_j) > 0$ stands for the deadheading cost induced by traversing the shortest path from vertices v_i to v_j .

In constraints (3) and (4), the inequality $(k_1, i_1) \neq (k_2, i_2)$ means that the two equalities $k_1 = k_2$ and $i_1 = i_2$ do not hold simultaneously. These two constraints prohibit that a task to be served more than once, either in the same route or different routes. Thus, constraints (2)–(4) ensure that all the tasks are served exactly once. Constraint (5) indicates that the total demand of each route should not exceed the vehicle capacity Q .

The challenge of CARP can be viewed from two perspectives. First, the optimal permutation of tasks corresponds to the optimal solution of CARP, and the latter can be obtained from the former in polynomial time [15]. Second, if the optimal assignments of tasks to vehicles are available, the optimal solution of CARP can also be obtained by solving several independent single-vehicle problems that are considerably easier than CARP. This paper takes the first perspective, i.e., the proposed approach aims to identify a good permutation efficiently.

III. HIERARCHICAL DECOMPOSITION OF CARP

It can be observed that the scale of a CARP instance mainly depends on the number of tasks to be served. Furthermore, a CARP can be addressed in two steps, i.e., finding a permutation of the task set and dividing this permutation into feasible routes. The optimal permutation must correspond to an optimal feasible solution, and vice versa [15], [30]. More importantly, given a permutation of tasks, the corresponding best feasible solution can be acquired in polynomial time [15]. Therefore, the key challenge to CARP can be viewed as finding the optimal permutation of tasks. The proposed HD scheme is essentially a method for finding a good permutation of tasks efficiently. To be concrete, HD introduces a number of virtual tasks to construct a hierarchical structure, as demonstrated in Fig. 1. In the figure, each node (in the l th layer) of the hierarchy corresponds to a virtual task τ_i^l , which is a permutation of several tasks. K_{l-1} represents the number of virtual tasks included in layer l . It will be discussed in detail in Section III-A. Each node at the bottom layer (i.e., layer 1) corresponds to a real task. The node at the top layer (i.e., layer h) represents a permutation of all the tasks. The hierarchy is built in a bottom-up way. At the bottom layer (i.e., layer 1),

Algorithm 1: HD(VT)

Procedure: HD(VT)
Input: virtual task set VT
Output: a permutation of tasks PT

- 1 **repeat**
- 2 randomly choose the cluster number $K \in [1, \beta \cdot |VT|]$;
- 3 divide VT into groups by using k-means;
- 4 order the virtual tasks within each group;
- 5 $VT \leftarrow \{\text{permutation of tasks in each group}\}$;
- 6 **until** $|VT| = 1$;
- 7 **return** the permutation of tasks in VT ;

tasks are grouped and ordered within each group. The permutation of tasks in each group is treated as a virtual task at layer 2. This procedure is executed on the obtained virtual tasks recursively until only 1 virtual task remains, which is a permutation of all the tasks. For example, suppose there are four tasks $\{\tau_1^1, \tau_2^1, \tau_3^1, \tau_4^1\}$ at layer 1, each two of them are connected to a node at layer 2, the virtual task corresponding to this node, denoted by τ_i^2 is a permutation of the two tasks, e.g., $\tau_1^2 = (\tau_2^1, \tau_1^1)$ and $\tau_2^2 = (\tau_3^1, \tau_4^1)$. τ_1^2 and τ_2^2 are then grouped and ordered, forming a virtual task at layer 3, e.g., $\tau_1^3 = (\tau_2^2, \tau_1^2) = (\tau_3^1, \tau_4^1, \tau_2^1, \tau_1^1)$. By this means, a permutation of the four tasks is obtained.

From the above description, for a given CARP instance, the HD scheme starts from the bottom layer and recursively group tasks into virtual tasks of a larger size until only 1 virtual task remains. Each node at nonbottom layers requires solving a subproblem to find the optimal permutation of the tasks (virtual tasks) assigned to that node, which is a partial-solution to the problem of finding optimal permutation of all tasks. Algorithm 1 demonstrates the general framework of the HD scheme. The benefits offered by such a hierarchy are twofold. From the perspective of problem decomposition, a linear decomposition scheme (e.g., as adopted by the CC-based approaches) has to involve a large number of small-size subproblems to cope with large-scale CARPs. Meanwhile, task grouping itself is a nontrivial problem, the complexity of which increases exponentially with the number of task groups (i.e., number of subproblems). Hence, it is highly likely that an inappropriate grouping will be obtained on large-scale CARPs, or a significant computational cost is needed to identify a good grouping of tasks. In other words, the performance of linear decomposition will deteriorate rapidly with the problem-size of CARP. As a result, the solution quality may also deteriorate rapidly since the grouping of tasks significantly affects the search course. In contrast, the HD scheme allows the number of nodes (i.e., virtual tasks) to decrease exponentially from the bottom layer toward the top layer. Thus, the cost for identifying a suitable grouping of tasks increases slowly with the scale of CARP. Hence, the scalability (in terms of the quality of task grouping) of HD is expected to be better than that of a linear grouping, and can lead to a better grouping of tasks (and thereby solution quality) than linear decomposition, especially when the total time budget for solving a CARP is limited and task grouping needs to be done as fast as possible. From the perspective of search effectiveness, since a subproblem (node) of an upper layer only takes the

partial-solutions obtained at the lower layers as its input, but does not change the inner structure of the partial-solutions, the hierarchical structure naturally allows searching at different step-sizes by solving subproblems at different layers and may lead to a more effective search.

The HD strategy involves two design issues. That is, how to group virtual tasks (i.e., lines 2 and 3 in Algorithm 1) and how to find the optimal permutation in a group of virtual tasks (i.e., line 4 in Algorithm 1). These will be detailed below.

A. Grouping Virtual Tasks

Intuitively, tasks close to each other are more likely to be served successively in solutions with high quality. Hence, a natural idea is to assign neighboring tasks to the same group. Thus, the closeness between two virtual tasks needs to be defined in order to group the virtual tasks. An intuitive idea is to take the deadheading cost between two virtual tasks as the closeness. Since the deadheading cost is defined between two vertices rather than two virtual tasks (paths) and four different deadheading costs can be obtained by connecting different pairs of endpoints of two virtual tasks, the closeness between virtual task τ_i^l and τ_j^l is defined as the average deadheading cost between them

$$\Delta(\tau_i^l, \tau_j^l) = \frac{dc(v_i, v_j) + dc(u_i, u_j) + dc(v_i, u_j) + dc(u_i, v_j)}{4} \quad (8)$$

where (v_i, u_i) and (v_j, u_j) denote the start and end vertices of τ_i^l and τ_j^l , respectively. $dc(v_i, v_j)$ is the deadheading cost between vertices v_i and v_j . As a special case, the closeness between virtual task τ_i^l and the depot v_0 is $\Delta(v_0, \tau_i^l) = ((dc(v_i, v_0) + dc(u_i, v_0))/2)$.

Given the closeness measure of two virtual tasks, grouping virtual tasks can be formulated as a clustering problem. In principle, any clustering method can be applied for this purpose. We choose the well-known k -means algorithm [31] for its simplicity. The k -means algorithm requires calculating the centroid of each cluster. This can be easily done in a real-space but cannot be directly computed in case of clustering virtual tasks. Hence, in each iteration of k -means, the centroid of a cluster is defined as the virtual task with minimal average closeness to other virtual tasks in the same cluster.

The number of clusters, K , is a user-defined parameter in k -means and has a great influence on the clustering results. In the HD scheme, since the clustering process is invoked at each layer and the number of virtual tasks varies over layers, a fixed value of K for all clustering processes is inappropriate. Note that each virtual task at layer l corresponds to a cluster obtained at layer $l - 1$. Hence, let K_l denote the number of clusters obtained at layer l , we set it as an integer randomly generated within $[1, \beta \cdot K_{l-1}]$, where $\beta \in (0, 1)$ is a predefined parameter. For the bottom layer (i.e., layer 1), $K_1 = |T|$.

Another important issue related to the effectiveness of k -means is the selection of initial cluster centroids. The initial centroids can be selected randomly or by using specific methods [32]–[34]. Here, we use a simple heuristic that disperse the centroids as widely as possible. Specifically,

Algorithm 2: HDU(VT)

Procedure: HDU(VT)

Input: virtual task set VT

Output: a feasible solution s ;

- 1 apply HD(VT) to generate a permutation of tasks PT ;
 - 2 apply Ulusoy's splitting procedure to partition PT into a solution s ;
 - 3 **return** s ;
-

the heuristic works by adding noncentroid virtual tasks into the centroid set one by one such that the sum of the closeness between the newly added virtual task and existing ones in the set is maximal. It should be noted that the depot is regarded as a dummy virtual task and added to the centroid set at the beginning.

B. Ordering Virtual Tasks

At each node of the hierarchical structure, the permutation of a subset of tasks or virtual tasks needs to be optimized. Recall that the capacity constraints are not considered when solving these subproblems, since the HD scheme aims to achieve the optimal permutation of all tasks regardless of capacity constraints. There exist lots of heuristic methods for finding such a permutation-based problem. Since a number of such subproblems need to be solved for building the hierarchy, it is not worth adopting a time-consuming method. For this consideration, we employ a greedy search heuristic named best insertion heuristic (BIH). BIH first chooses the nearest virtual task to the depot in terms of deadheading cost. Then, at each iteration, the virtual task with the minimal deadheading cost to the current endpoint of the path is added to the end of the path. If multiple virtual tasks satisfy this condition, only one is randomly chosen. The process terminates when all virtual tasks have been added to the path.

C. Generating Solution to CARP Based on Hierarchical Decomposition

As mentioned before, the HD scheme seeks a good permutation of all tasks in CARP regardless of capacity constraint. Given a permutation of tasks, a solution (with respect to this permutation) to CARP can be obtained by splitting the permutation into a number of routes that satisfy capacity constraints. This can be done with well-established methods in polynomial time [15]. Herein, we employ the well-known Ulusoy's splitting procedure [10], an exact method that has been proved to be capable of solving the problem optimally. Thus, the combination of the HD scheme and the Ulusoy's splitting procedure, namely HDU as demonstrated in Algorithm 2, forms our approach to CARPs.

IV. SCALABLE APPROACH BASED ON HIERARCHICAL DECOMPOSITION

The HDU described in the previous section obtains a solution to a CARP in a constructive way. It can also be embedded into an iterative search process, which allows the solution

obtained using HDU to be further improved. The proposed SAHiD is developed following this idea. Briefly speaking, SAHiD is an individual-based iterative search method. At each iteration, it first employs HDU to obtain a solution to the CARP, and then some traditional local search operator is applied to further improve the solution obtained. Specifically, SAHiD involves three phases, i.e., initialization, reconstruction, and local search, as detailed below.

A. Initialization

At the first iteration, an initial solution, say s , is obtained by applying HDU to the CARP instance. Then, the local search operator is applied to s for further improvement.

B. Reconstruction

In the reconstruction phase, HDU is applied to generate new candidate solutions. But different from in the initialization phase, HDU is not applied to achieve a solution from scratch, i.e., based on the un-ordered set of tasks. Instead, the solution obtained in the last iteration, say s , is first randomly split into a number of virtual tasks. Specifically, each route of s is split into two virtual tasks with a predefined probability α , resulting in a set of virtual tasks. Then, HDU is applied to this set to obtain a new solution. Since s is built based on the original set of tasks and polished with local search, it is expected to contain some useful pattern of a good solution. Splitting the routes does not change the permutation of tasks in the same virtual task, and thus are likely to keep the useful patterns. Hence, by applying HDU to the virtual tasks rather than the original tasks, the useful pattern in the previous solution can be exploited, and thus benefit the search for a better solution.

C. Local Search

When a solution is obtained by HDU, either in the initialization or reconstruction phase, a first improvement local search procedure is applied to further improve it. The local search starts with a reverse move operator similar to the 2-opt operator [15] for a single route, i.e., it reverses the direction of a subroute (i.e., part of a route). Suppose the reverse operator is applied to a route consisting of t tasks. At each iteration, all possible subroutes are enumerated with the length of subroute increasing from 1 to $t - 1$. During this course, the current solution is updated once a solution with a lower cost is found. This procedure terminates when all subroutes of each route is checked at least once, and the whole local search procedure terminates if the solution obtained by HDU is updated at least once.

If the reverse operator fails to improve the solution obtained by HDU, the merge-split (MS) operator [16] is applied to conduct a best improvement search. That is, at each step, all solutions that can be reached by the MS operator from the current solution are examined and the best and improved one is chosen to replace the current solution. Compared to the reverse operator, MS is a search operator with a larger step-size, and thus is more likely to escape from the current solution, which is a local optimum. Interested readers are referred to [16] for detailed steps of MS. If the MS operator

Algorithm 3: Local Search Procedure LS(s)

```

Procedure: LS( $s$ )
Input: solution  $s$ 
Output: potentially improved solution  $s$ 
1 repeat
2   for each sub-routes  $SR$  of each route  $R$  in  $s$  do
3     reverse  $SR$  to obtain a new solution  $s'$ ;
4     if  $s'$  is better than  $s$  then
5        $s \leftarrow s'$ ;
6       break;
7     end if
8   end for
9 until  $s$  remains unchanged;
10 if  $s$  is not updated then
11   apply MS operator to improve  $s$ ;
12   if  $s$  is updated then
13     repeat
14       for each sub-routes  $SR$  of each route  $R$  in  $s$  do
15         reverse  $SR$  to obtain a new solution  $s'$ ;
16         if  $s'$  is better than  $s$  then
17            $s \leftarrow s'$ ;
18           break;
19         end if
20       end for
21     until  $s$  remains unchanged;
22   end if
23 end if
24 return  $s$ ;

```

manages to find a better solution, the reverse operator will be applied to the improved solution again to exploit the new local region. Otherwise, the whole local search procedure terminates with the solution obtained by HDU remaining unchanged.

The pseudo-code of the local search procedure is presented in Algorithm 3. Note that HDU always produces feasible solutions, the reverse operator only changes the order of tasks within a feasible route, and the MS operator also always generates feasible routes. Hence, no infeasible solution will be produced during the search process of SAHiD. For this reason, only the total costs are taken into account when comparing two solutions.

Algorithm 4 depicts the steps of SAHiD. It is noteworthy that the best solution found so far is stored in an external archive (lines 10–12) and outputted as the final solution. It might be inappropriate to keep the best solution in the search process of SAHiD if it cannot be improved for a long time. Otherwise, the search will be stuck at this local best solution. Hence, we adopt the threshold accepting idea [35] in SAHiD. Given a solution s , if no better solution is found after σ consecutive iterations, a new solution worse than s will still be accepted (i.e., replace s) as long as its quality is not worse than $\theta\%$ of that of the best-found solution. Finally, the SAHiD can be terminated either when a predefined time budget is used up or no better solution is found for a predefined number of iterations.

V. EXPERIMENTAL STUDIES

To evaluate the effectiveness of SAHiD, two sets of empirical studies have been conducted to compare SAHiD against a number of state-of-the-art approaches to CARPs. In the

Algorithm 4: Pseudo Code of SAHiD(T)

```

Procedure: SAHiD( $T$ )
Input: task set  $T$ 
Output: a feasible solution  $s^*$ 
1 generate an initial solution  $s$  using HDU( $T$ );
2 apply LS( $s$ ) to improve  $s$ ;
3  $s^* \leftarrow s$ 
4 while stopping criteria are not met do
5     generate a virtual task set  $VT$  by splitting the routes of  $s$ ;
6     generate a solution  $s'$  using HDU( $VT$ );
7     apply LS( $s'$ ) to improve  $s'$ ;
8     if  $s'$  is acceptable then
9          $s \leftarrow s'$ 
10        if  $s'$  is better than  $s^*$  then
11             $s^* \leftarrow s'$ 
12        end if
13    end if
14 end while
15 return  $s^*$ ;

```

TABLE I
INFORMATION OF INSTANCES IN HEFEI AND BEIJING TEST SET

Name	$ V $	$ E $	$ T $	Proportion of tasks	Q	M
Hefei-1	850	1212	121	10%	9000	7
Hefei-2	850	1212	242	20%	9000	14
Hefei-3	850	1212	364	30%	9000	19
Hefei-4	850	1212	485	40%	9000	28
Hefei-5	850	1212	606	50%	9000	35
Hefei-6	850	1212	727	60%	9000	42
Hefei-7	850	1212	848	70%	9000	49
Hefei-8	850	1212	970	80%	9000	56
Hefei-9	850	1212	1091	90%	9000	63
Hefei-10	850	1212	1212	100%	9000	69
Beijing-1	2820	3584	358	10%	25000	7
Beijing-2	2820	3584	717	20%	25000	11
Beijing-3	2820	3584	1075	30%	25000	18
Beijing-4	2820	3584	1433	40%	25000	23
Beijing-5	2820	3584	1792	50%	25000	30
Beijing-6	2820	3584	2151	60%	25000	36
Beijing-7	2820	3584	2509	70%	25000	41
Beijing-8	2820	3584	2868	80%	25000	47
Beijing-9	2820	3584	3226	90%	25000	52
Beijing-10	2820	3584	3584	100%	25000	58

first study, the performance of different algorithms is examined in terms of the time required to achieve a predefined solution quality. In the second study, the algorithms are compared from the perspective of solution quality obtained using a predefined time budget. In addition, further empirical analysis has also been conducted to assess the contribution of the HD scheme to SAHiD.

A. Benchmark Set

Since this paper mainly studies the scalability of search methods, we are more interested in the performance on large-scale CARPs rather than small scale ones. Furthermore, as mentioned before, most of the existing small or median-scale CARP benchmark instances can be solved near optimally in a rather short time period (e.g., 10 s). Therefore, instead of using the more traditional benchmark sets, two new sets of

TABLE II
PARAMETER SETTINGS OF SAHiD

Name	Description	Value
β	Scale parameter in HD (Sub-Section III.A)	0.1
α	Probability of partitioning a route (Section IV)	0.1
p	Parameter of the MS operator (Section IV)	2
θ	Parameter of Threshold accepting (Section IV)	110%
σ	Maximum number of idle iterations for accepting an ascending move (Section IV)	10000

CARP instances, namely Hefei and Beijing test sets,¹ are generated. The Hefei set is generated from the map of the Hefei city in China, which consists of 1212 main roads (i.e., edges). The Beijing set is generated from the central area (the area inside the fifth ring road) of Beijing, China, which consists of 3584 main roads. For each set, ten instances are generated by randomly setting part of the edges as tasks. The number of tasks for each set increases from 10% to 100% of the number of edges with a step-size 10%. Table I shows the detailed information of these two benchmark sets. It can be observed that these two sets involves instances that are one order of magnitude larger than the largest CARP benchmark instances used in the literature, i.e., EGL-G, which consists of at most 375 edges and 375 tasks. Furthermore, since the major challenge of CARP depends on the number of tasks rather than the number of edges, the Hefei and Beijing sets allow assessing the scalability of an algorithm in this regard. It should also be noted that the performance of an algorithm on a CARP is also affected by the capacity constraints. Thus, the same capacity constraints are set to all instances in the same set, so as to focus our investigation on scalability. M is the minimal number of vehicles required to serve all the tasks, which is obtained as follows:

$$M = \left\lceil \frac{\sum_{\tau \in \mathcal{T}} d(\tau)}{Q} \right\rceil. \quad (9)$$

B. Compared Algorithms

Three algorithms, including variable neighborhood search (VNS) [7], tabu search algorithm 1 (TSA1) [17] and RDG-MAENS [23], are chosen for our comparative studies. VNS and TSA1 are both individual-based search approaches for CARPs. They have shown appealing performance not only in terms of solution quality, but also (and more importantly in the context of this paper) in terms of efficiency. RDG-MAENS is an approach dedicated to large-scale CARPs and has been shown to outperform other population-based search methods, e.g., MAENS [16], on large-scale CARP instances. Thus, it was chosen as the state-of-the-art representative population-based search methods for CARPs.

C. Experimental Protocol

To make a fair comparison, all algorithms involved in the empirical studies are implemented in C++ and run on the same computing platform, i.e., Intel Core i7-4790 processor

¹Instances of the two test sets are available at <http://staff.ustc.edu.cn/~ketang/codes/LSCARPset.zip>.

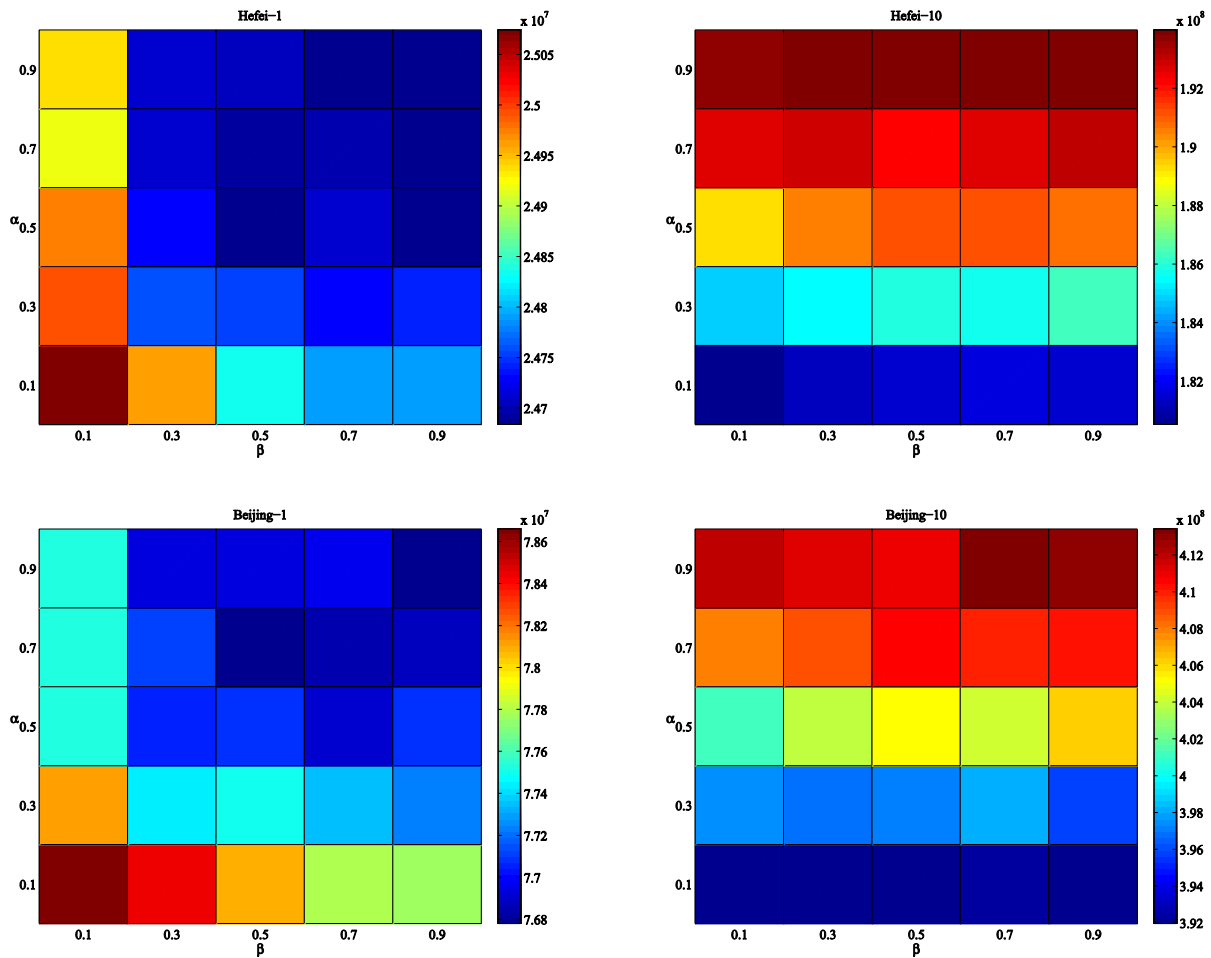


Fig. 2. Heat map of average total cost obtained by SAHiD with different values of α and β on instances Hefei-1, Hefei-10, Beijing-1, and Beijing-10. Blue indicates better results and red stands for worse results.

with 3.6 GHz. For all experiments presented hereafter, the results are obtained by executing the algorithms for 25 independent runs. Each of the tested algorithms consists of a few parameters to be predefined.

SAHiD has five user-defined parameters, i.e., the scale parameter β in HD; the probability α of partitioning a route in the reconstruction phase; parameter p introduced by the MS operator [16] and θ and σ introduced by threshold accepting strategy [35], [36]. Since the last three parameters are proposed in the previous works for local search, but are not introduced by the HD decomposition scheme, the values suggested in the original publications are directly adopted. For the parameters α and β , a sensitivity analysis is carried out to test the performance of SAHiD as well as choosing parameter values for more comprehensive empirical studies. Specifically, five values (0.1, 0.3, 0.5, 0.7, and 0.9) were tested for both parameters, which led to 25 combinations of parameters values. The sensitivity analysis is conducted on four typical instances, including Hefei 1, Hefei 10, Beijing 1, and Beijing 10. For each instance, SAHiD is executed for 10 times with each of the 25 parameter combinations (i.e., 250 runs in total). Fig. 2 depicts the heat maps of the average results (over 10 runs) obtained by SAHiD with different values of α and β . It can be observed that α is more critical to performance on

larger instances (Hefei 10 and Beijing 10), while β mainly affects the performance of SAHiD on smaller instances. The reason is that, HD is the core of SAHiD, the tree size obtained by HD has a great influence on the algorithm performance. Both α and β affect the size of tree, α determines the number of leaf nodes and β controls the upper bound of number of intermediate nodes. As described in Section III-A, the number of nodes in layer l is within $[1, \beta \cdot K_{l-1}]$, where K_{l-1} is the number of nodes in layer $l-1$. Hence, the number of intermediate nodes is not only affected by β but also determined by the number of bottom nodes (leaf nodes), which means α plays a more important role than β when the problem size is large. On the smaller instances (Hefei 1 and Beijing 1), the number of routes is relatively small. Thus, a larger α is needed to ensure a sufficient number of leaf nodes (virtual tasks), if there are very few leaf nodes, the role of the hierarchy will be greatly reduced. For the larger instances (i.e., Hefei 10 and Beijing 10), on the other hand, a small value of α already led to splitting a large number of routes. In fact, Table I indicates that the routes (i.e., the number of vehicles, M) in a solution to Hefei 10 is about 10 times larger than that in a solution to Hefei 1. Thus, setting α to 0.9 and 0.1 for Hefei 1 and Hefei 10 actually resulted in comparable number of routes being split. Hence, as a rule of thumb,

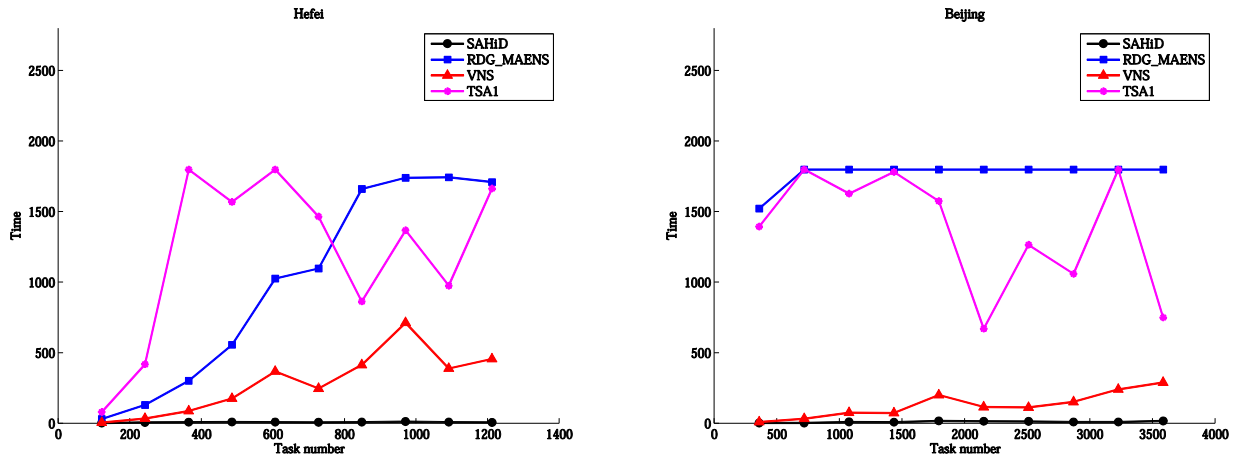


Fig. 3. Average computational time (to achieve a predefined solution quality) versus the number of tasks over all the instances of Hefei and Beijing sets for each compared algorithm.

TABLE III

RESULTS ON HEFEI BENCHMARK SET IN TERMS OF THE TOTAL SOLUTION COSTS. BEST AND AVERAGE STAND FOR THE BEST AND AVERAGE RESULTS OBTAINED FROM 25 INDEPENDENT RUNS. STD STANDS FOR THE STANDARD DEVIATION. THE MINIMAL AVERAGE RESULTS ARE MARKED WITH *. FOR EACH INSTANCE, BOLD (UNDERLINED) RESULTS INDICATE THAT THE CORRESPONDING ALGORITHM IS BETTER (WORSE) THAN SAHiD BASED ON WILCOXON RANK-SUM TEST WITH THE LEVEL OF SIGNIFICANCE 0.05. “# OF ‘W-D-L’” SUMMARIZES THE NUMBER OF “WIN-DRAW-LOSE” OF SAHiD VERSUS THE OTHER ALGORITHMS

Name	V	T	E	Q	SAHiD			RDG-MAENS			VNS			TSA1		
					Best	Average	Std	Best	Average	Std	Best	Average	Std	Best	Average	Std
Hefei-1	850	1212	121	9000	248048	251024	1820	246221	247341*	2293	245596	247819	2745	250155	<u>252615</u>	1591
Hefei-2	850	1212	242	9000	441574	445376	2476	436020	441539*	4142	436637	<u>449979</u>	5375	447853	<u>456228</u>	5539
Hefei-3	850	1212	364	9000	586880	590969	2305	583050	589152*	2697	588682	<u>595263</u>	3108	623795	<u>637201</u>	8003
Hefei-4	850	1212	485	9000	754015	759402*	2495	754855	761351	4362	763256	<u>774323</u>	6394	774182	<u>791790</u>	5481
Hefei-5	850	1212	606	9000	964772	976276*	4742	980153	<u>991813</u>	5755	984121	<u>994794</u>	6109	1019224	<u>1042701</u>	11496
Hefei-6	850	1212	727	9000	1095530	1106735*	5318	1119584	<u>1132063</u>	8966	1110030	<u>1128667</u>	9404	1134041	<u>1162641</u>	13806
Hefei-7	850	1212	848	9000	1299430	1309474*	4792	1329745	<u>1361125</u>	14356	1322290	<u>1337353</u>	6745	1339160	<u>1353502</u>	6235
Hefei-8	850	1212	970	9000	1474390	1483694*	4857	1526453	<u>1550509</u>	13695	1492790	<u>1517151</u>	12477	1521857	<u>1537169</u>	6709
Hefei-9	850	1212	1091	9000	1648840	1659700*	6103	1705381	<u>1749079</u>	18872	1675790	<u>1694957</u>	10164	1696706	<u>1716256</u>	9236
Hefei-10	850	1212	1212	9000	1793890	1808860*	7836	1837767	<u>1923264</u>	31697	1834860	<u>1852622</u>	10183	1873504	<u>1901167</u>	12679
# of “w-d-l”									6-1-3			9-0-1			10-0-0	

α is suggested to take the value that will lead to the split of around seven routes in the reconstruction phase. Since β does not appear to affect SAHiD as much as α , 0.1 can be used as the default value. The results reported in this paper were all obtained with $\alpha = 0.1$ and $\beta = 0.1$.

Table II summarizes the parameter settings for SAHiD. For the compared VNS [7], TSA1 [17], and RDG-MAENS [23], the best parameter settings reported in the original publications are employed. By this means, we hope to keep the comparison as fair as possible.

D. Comparison in Terms of Runtime

Runtime is one of the most important issues when investigating the scalability of an algorithm. Ideally, the runtime for an algorithm to achieve the optimal solution or a solution within a given approximation ratio should be tested. However, such an analysis cannot be done for large-scale CARP instances used in this paper, because the optimal solution and the lower bound on the total cost are unknown for

the instances. Hence, we resort to a threshold of total costs as the target for the compared algorithms. Specifically, for the Hefei and Beijing sets, SAHiD is first run for 25 times. A time budget of 30 s is given for each run on each instance of Hefei and Beijing sets. For each instance, the total costs of the 25 final solutions are recorded. The largest one among these 25 values indicates the worst performance of SAHiD in the 25 runs. Thus, it is used as the target for other compared algorithms. The runtime for each of the other algorithms to achieve the same solution quality for the first time is recorded. Since no matter how the target solution quality is chosen, it is possible that an algorithm may not reach this target or takes extremely long time to reach it, the compared algorithms are terminated if a larger time budget of 30 min is used up.

Fig. 3 depicts the average runtime for the compared algorithms on the Hefei and Beijing sets. Each point in the figure corresponds to the runtime of an algorithm on an instance. Note that, since the worst total costs obtained by SAHiD are used as the target, the average runtime required by SAHiD on each instance is always not greater than 30 s

TABLE IV

RESULTS ON BEIJING BENCHMARK SET IN TERMS OF THE TOTAL SOLUTION COSTS. BEST AND AVERAGE STAND FOR THE BEST AND AVERAGE RESULTS OBTAINED FROM 25 INDEPENDENT RUNS. STD STANDS FOR THE STANDARD DEVIATION. THE MINIMAL AVERAGE RESULTS ARE MARKED WITH *. FOR EACH INSTANCE, BOLD (UNDERLINED) RESULTS INDICATE THAT THE CORRESPONDING ALGORITHM IS BETTER (WORSE) THAN SAHiD BASED ON WILCOXON RANK-SUM TEST WITH THE LEVEL OF SIGNIFICANCE 0.05. # OF W-D-L SUMMARIZES THE NUMBER OF WIN-DRAW-LOSE OF SAHiD VERSUS THE OTHER ALGORITHMS

Name	V	T	E	Q	SAHiD			RDG-MAENS			VNS			TSA1		
					Best	Average	Std	Best	Average	Std	Best	Average	Std	Best	Average	Std
Beijing-1	2820	3584	358	25000	775523	784727	5591	812647	<u>829406</u>	12688	774502	782415*	4452	813907	<u>829132</u>	6340
Beijing-2	2820	3584	717	25000	1167480	1183955*	8431	1303570	<u>1337954</u>	18939	1168190	<u>1192292</u>	10196	1353567	<u>1401363</u>	25378
Beijing-3	2820	3584	1075	25000	1586180	1605846*	9231	1777852	<u>1847922</u>	33258	1591540	<u>1618484</u>	11888	1678224	<u>1709279</u>	14801
Beijing-4	2820	3584	1434	25000	1910880	1936994*	11694	2126151	<u>2193399</u>	34159	1920330	<u>1953892</u>	16746	2053938	<u>2070885</u>	14532
Beijing-5	2820	3584	1792	25000	2273080	2298630*	16879	2581910	<u>2639458</u>	32481	2293120	<u>2335915</u>	23040	2396483	<u>2440319</u>	26726
Beijing-6	2820	3584	2151	25000	2664510	2707500*	18433	2968102	<u>3047295</u>	41112	2705060	<u>2743677</u>	18024	2774161	<u>2814735</u>	22018
Beijing-7	2820	3584	2509	25000	3013590	3038157*	15658	3331900	<u>3388263</u>	26081	3015790	<u>3063813</u>	25226	3147294	<u>3186240</u>	22426
Beijing-8	2820	3584	2868	25000	3283530	3313590*	21925	3584696	<u>3697025</u>	44951	3323850	<u>3366215</u>	24686	3415275	<u>3456037</u>	22381
Beijing-9	2820	3584	3226	25000	3621490	3684250*	32404	3934270	<u>4061793</u>	49504	3653630	<u>3723830</u>	45148	3890129	<u>3943883</u>	37089
Beijing-10	2820	3584	3584	25000	3935540	4004310*	29488	4206005	<u>4353966</u>	51063	4002040	<u>4040694</u>	27384	4066188	<u>4103532</u>	15501
# of "w-d-l"									10-0-0			9-1-0			10-0-0	

TABLE V

RESULTS ON EGL-G BENCHMARK SET IN TERMS OF THE TOTAL SOLUTION COSTS. BEST AND AVERAGE STAND FOR THE BEST AND AVERAGE RESULTS OBTAINED FROM 25 INDEPENDENT RUNS. STD STANDS FOR THE STANDARD DEVIATION. THE MINIMAL AVERAGE RESULTS ARE MARKED WITH *. FOR EACH INSTANCE, BOLD (UNDERLINED) RESULTS INDICATE THAT THE CORRESPONDING ALGORITHM IS BETTER (WORSE) THAN SAHiD BASED ON WILCOXON RANK-SUM TEST WITH THE LEVEL OF SIGNIFICANCE 0.05. # OF W-D-L SUMMARIZES THE NUMBER OF "WIN-DRAW-LOSE" OF SAHiD VERSUS THE OTHER ALGORITHMS

Name	V	T	E	Q	SAHiD			RDG-MAENS			VNS			TSA1		
					Best	Average	Std	Best	Average	Std	Best	Average	Std	Best	Average	Std
EGL-G1-A	255	375	347	28600	1017030	1033874	6588	1002347	1014177*	5898	1048220	<u>1061032</u>	8238	1017089	1034531	9933
EGL-G1-B	255	375	347	22800	1139860	1155899	7812	1124610	1133625*	5960	1160620	<u>1191821</u>	13372	1130096	1145438	10493
EGL-G1-C	255	375	347	19000	1270110	1281702	7131	1252061	1261541*	6933	1303100	<u>1324484</u>	12496	1263835	1287478	12070
EGL-G1-D	255	375	347	16200	1397490	1422693	7387	1386044	1397790*	6851	1430320	<u>1465704</u>	15134	1411361	<u>1433744</u>	14015
EGL-G1-E	255	375	347	14100	1549540	1565029	6856	1527473	1542491*	9543	1575240	<u>1614683</u>	12059	1563234	<u>1597127</u>	16095
EGL-G2-A	255	375	375	28000	1128040	1141925	8592	1108242	1122013*	7280	1142490	<u>1166879</u>	14840	1119328	1130786	9423
EGL-G2-B	255	375	375	23100	1238740	1253578	7236	1221077	1233611*	6088	1279280	<u>1293706</u>	8402	1242060	<u>1258284</u>	7418
EGL-G2-C	255	375	375	19400	1374000	1386629	7823	1355956	1369175*	7305	1411810	<u>1439277</u>	13400	1378920	<u>1398971</u>	14892
EGL-G2-D	255	375	375	16700	1505570	1529137	9698	1485341	1506033*	8679	1563960	<u>1587437</u>	14493	1522390	<u>1555457</u>	15661
EGL-G2-E	255	375	375	14700	1656860	1673222	7754	1637063	1649882*	7608	1694730	<u>1724093</u>	13871	1672677	<u>1704553</u>	21171
# of 'w-d-l'									0-0-10			10-0-0			6-2-2	

for Hefei and Beijing sets. From the figure, it can be observed that VNS, TSA1, and RDG-MAENS all consume much more runtime than SAHiD. In fact, some even fail to reach the target solution quality in 30 min for some large-scale instances. For example, RDG-MAENS fails in all 25 runs on all the Beijing instances except for Beijing-1, on which the average runtime consumed is still close to the given time budget. TSA1 also reaches the given time budget on 2 out of the 20 instances, and always takes more than 500 s to reach the target solution quality. VNS appears to be more efficient than RDG-MAENS and TSA1, but the runtime consumed by it show a growth trend with respect to the problem size and thus its scalability is not as good as SAHiD. These observations clearly demonstrate the superiority of SAHiD to the compared algorithms.

E. Comparison in Terms of Solution Quality

In addition to runtime, another important characteristic of an algorithm is the solution quality that can be achieved with

a given time budget. This is evaluated by our second experiment. Specifically, each algorithm is given 30 min to search for the solution to a CARP instance in Hefei and Beijing benchmark sets. Further, since the EGL-G benchmark set [17] has been used to evaluate TSA1 and RDG-MAENS in the original publications, this set of instances is also employed in the experiment. The time budget is set to 15 min as the scale of EGL-G is much smaller than Hefei and Beijing sets.

Tables III–V present the costs of the final solutions obtained by the compared algorithms on the three test sets. The first 5 columns present the basic information of instances. For each compared algorithm, the columns headed "best" and "average" provide the best and average costs among the 25 runs, respectively. The last column headed "std" present the standard deviations calculated over the 25 runs. The minimal average results are marked with "*" On each instance, SAHiD is compared to RDG-MAENS, VNS, and TSA1 by using Wilcoxon rank-sum test with the level of significance 0.05 over 25 runs, results highlighted in bold/underline

TABLE VI

RESULTS OF SAHiD AND SARANDOM ON ALL TEST SETS IN TERMS OF THE TOTAL SOLUTION COSTS. BEST AND AVERAGE STAND FOR THE BEST AND AVERAGE RESULTS OBTAINED FROM 25 INDEPENDENT RUNS. STD STANDS FOR THE STANDARD DEVIATION. FOR EACH INSTANCE, BOLD (UNDERLINED) RESULTS INDICATE THAT SARANDOM IS BETTER (WORSE) THAN SAHiD BASED ON WILCOXON RANK-SUM TEST WITH THE LEVEL OF SIGNIFICANCE 0.05

Name	SAHiD			SArandom-			RPD(%)
	Best	Average	Std	Best	Average	Std	
EGL-G1-A	1017030	1033874	6588	1038660	<u>1051855</u>	6843	1.74
EGL-G1-B	1139860	1155899	7812	1159420	<u>1182700</u>	10447	2.32
EGL-G1-C	1270110	1281702	7131	1298250	<u>1312731</u>	7861	2.42
EGL-G1-D	1397490	1422693	7387	1444100	<u>1461256</u>	7968	2.71
EGL-G1-E	1549540	1565029	6856	1594180	<u>1613778</u>	7966	3.11
EGL-G2-A	1128040	1141925	8592	1145130	<u>1163482</u>	10036	1.89
EGL-G2-B	1238740	1253578	7236	1263190	<u>1282771</u>	7966	2.33
EGL-G2-C	1374000	1386629	7823	1414250	<u>1433035</u>	9519	3.35
EGL-G2-D	1505570	1529137	9698	1562830	<u>1577365</u>	9240	3.15
EGL-G2-E	1656860	1673222	7754	1715300	<u>1730814</u>	8614	3.44
							2.65(mean)
<i>Hefei-1</i>	248048	251024	1820	246265	247401	776	-1.44
<i>Hefei-2</i>	441574	445376	2476	439437	445329	3580	-0.01
<i>Hefei-3</i>	586880	590969	2305	589521	<u>598517</u>	3321	1.28
<i>Hefei-4</i>	754015	759402	2495	778721	<u>786735</u>	4942	3.6
<i>Hefei-5</i>	964772	976276	4742	1005620	<u>1024557</u>	8398	4.95
<i>Hefei-6</i>	1095530	1106735	5318	1161690	<u>1179338</u>	8389	6.56
<i>Hefei-7</i>	1299430	1309474	4792	1376010	<u>1399276</u>	14850	6.86
<i>Hefei-8</i>	1474390	1483694	4857	1578120	<u>1603242</u>	11507	8.06
<i>Hefei-9</i>	1648840	1659700	6103	1777520	<u>1808153</u>	12278	8.94
<i>Hefei-10</i>	1793890	1808860	7836	1953270	<u>1986672</u>	17833	9.83
							4.86(mean)
<i>Beijing-1</i>	776379	784891	6223	767512	777961	5902	-0.86
<i>Beijing-2</i>	1180010	1188836	6589	1170610	1182402	7381	-0.13
<i>Beijing-3</i>	1593720	1610383	8198	1632690	<u>1658375</u>	16942	3.27
<i>Beijing-4</i>	1927850	1943053	9055	1985840	<u>2036163</u>	28357	5.12
<i>Beijing-5</i>	2273080	2298630	16879	2439860	<u>2504124</u>	38455	8.94
<i>Beijing-6</i>	2664510	2707500	18433	2885540	<u>3004417</u>	86641	10.97
<i>Beijing-7</i>	3013590	3038157	15658	3281750	<u>3382513</u>	77413	11.33
<i>Beijing-8</i>	3283530	3313590	21925	3564160	<u>3728470</u>	103502	12.52
<i>Beijing-9</i>	3621490	3684250	32404	3972310	<u>4162260</u>	128888	12.97
<i>Beijing-10</i>	3935540	4004310	29488	4292190	<u>4515296</u>	114710	12.76
							7.69(mean)

indicate that the corresponding algorithm is significantly better/worse than SAHiD on the corresponding instance. Results without any symbol indicate that the difference between SAHiD and the corresponding algorithm is statistically insignificant.

From Tables III and IV, it can be observed that SAHiD significantly outperforms the other algorithms in terms of solution quality. It achieves the smallest total costs on 7 out of 10 instances in Hefei and 9 out of 10 instances in Beijing. Statistical tests also confirmed that the significant difference between SAHiD and the other algorithms. The results in Tabel V are more mixed. To be specific, SAHiD is not as competitive as RDG-MAENS on the 10 EGL-G instances. In comparison to TSA1, SAHiD performs better on six instances, i.e., EGL-G1-D, EGL-G1-E, EGL-G2-B, EGL-G2-C, EGL-G2-D, and EGL-G2-E, while is inferior on the other four instances. SAHiD still outperforms VNS on all ten instances. The inferior performance of SAHiD to RDG-MAENS on the EGL-G set is understandable, because the former is designed with the aim to tackle large-scale CARPs. As the size of EGL-G is relatively small, a more costly but more precise search method like RDG-MAENS should be able to find a better solution while the computational time is still

acceptable. In addition, unlike the Hefei and Beijing sets, different EGL-G instances are subject to different capacity constraints. Thus, another interesting observation from Table V is that SAHiD outperforms TSA1 mainly in the cases where the capacity constraints are tighter. A possible reason might be that SAHiD always visits feasible solutions during the search, while TSA1 may generate infeasible solutions and thus only part of the time budget is used for searching in the feasible region.

Furthermore, the performance of the algorithm is also tested using different time budgets less than the above given time budgets. The average costs obtained over 25 runs are plotted in Fig. 4. For the sake of brevity, only the results on two instances are provided for each benchmark set. The results on other instances follow a similar pattern and are made available in the online appendix.² The figure further confirms the superiority of SAHiD on large scale instances. For instance, the curve for SAHiD is always beneath the curves of the other algorithms on Hefei-10 and Beijing-10, indicating that SAHiD can always perform the best among the 4 algorithms if a solution is needed with a tighter time budget on a large scale instance. As for the

²Available at <http://staff.ustc.edu.cn/~ketang/codes/SAHiDresults.pdf>.

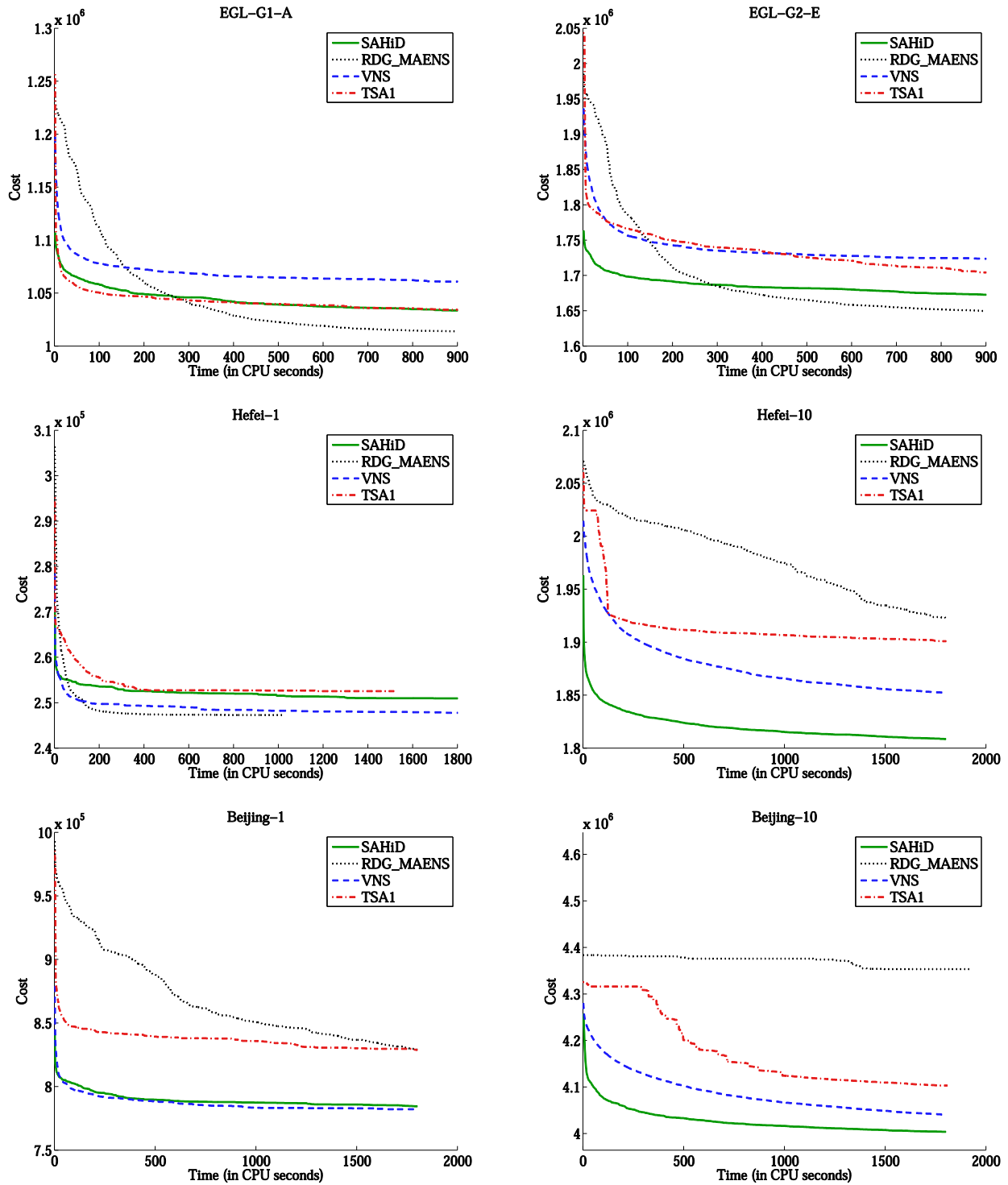


Fig. 4. Convergence curves of SAHiD, RDG-MAENS, VNS, and TSA1 on instances EGL-G1-A, EGL-G2-E, Hefei-1, Hefei-10, Beijing-1, and Beijing-10.

EGL-G instances, it can be found that curves of SAHiD drop rapidly with time, but level off later. RDG-MAENS improves the solution slower than SAHiD at the beginning, but continues to improve it. This observation is consistent with the expectation that RDG-MAENS is a more costly but more precise method, which might achieve better solutions than SAHiD if the problem size is moderate in comparison to the time budget.

F. Analysis of the Contribution of HD to SAHiD

Since the core component of SAHiD is the HD scheme. It is also interesting to investigate whether the HD scheme is indispensable for SAHiD. For this purpose, another algorithm namely SARandom is developed. The only difference between SAHiD and SARandom is that the latter does not use the HD scheme in the reconstruction phase (line 6 in Algorithm 4).

Instead, SARandom randomly merges virtual tasks into permutations. As in Section V-E, SARandom is applied to the three benchmark sets and the results are summarized in Table VI. The relative percentage of deviation (RPD) is given in the last column for each instance. RPD is the ratio of the difference between the average costs obtained by SAHiD and SARandom to the average costs obtained by SARandom. It measures the improvement that can be achieved by replacing the random permutation scheme with the HD scheme, i.e., a larger RPD indicates a more significant influence of HD on SAHiD.

Table VI shows that SAHiD outperforms SARandom on 28 out of 30 test instances and the gap between them shows a growing trend with respect to the problem size (e.g., the average RPD on EGL-G, Hefei and Beijing are 2.65%, 4.86%, and 7.69%, respectively). Moreover, comparing Table VI to Tables III and IV reveals that the results of SARandom are significantly worse than those of RDG-MAENS, VNS, and TSA1. Specifically, SARandom achieves worse performance than TSA1 on 11 out of 20 instances in Hefei and Beijing sets (the largest 5 of Hefei instances and 6 of Beijing instances). When compared to RDG-MAENS, SARandom performs worse on all the Hefei instances and the largest 3 Beijing instances (i.e., Beijing-8, Beijing-9, and Beijing-10). The gap between SARandom and VNS is more obvious, VNS beats SARandom on 16 out of 20 instances in Hefei and Beijing sets (8 of Hefei and 8 of Beijing). This fact confirms that the advantages of SAHiD over the other compared algorithms should be credited to the HD scheme.

VI. CONCLUSION

This paper presents a novel approach, namely SAHiD, to CARPs. SAHiD distinguishes from previous methods in the sense that it employs a HD scheme, which is capable of generating a good permutation of tasks, i.e., an intermediate solution to the CARP, in a very efficient way. By employing the proposed HD scheme in an iterative search process, SAHiD can tackle CARP instances of large scales. Empirical studies on two new CARP benchmark sets that are one order of magnitude larger than the existing ones show that SAHiD significantly outperforms state-of-the-art methods in terms of both computational time and solution quality (given a time budget less than 30 min). Hence, SAHiD can better scale up to large-scale CARPs than the compared methods, particularly in cases when a solution needs to be obtained in a few minutes or even seconds.

The promising performance of SAHiD has pointed to several future research directions. First, in addition to the HD scheme, the other components of SAHiD, e.g., the methods of grouping and ordering virtual tasks in HD and the local search procedure, can be improved using alternative techniques in the literature. Since instances generated from real-world maps may not represent all instances that could be synthesized using the mathematical formulation of CARP. The specific clustering method used in HD might fail in some of the latter cases. Hence, the interactions between them and the decomposition scheme can be further investigated, so as to develop novel components that suit SAHiD better. Second, the experimental

results reveal that the capacity constraints may also affect the scalability of an algorithm on CARPs. Although this observation seems to be obvious, it has never been systematically studied and it is unclear how the scalability of CARP solvers can be enhanced in this aspect. Finally, the HD scheme proposed in this paper is in essence a method for efficiently obtaining good permutations of tasks. It can be generalized to other permutation-based optimization problems, e.g., vehicle routing [37], [38], scheduling [39]–[41], and path planning [42], [43], as long as a suitable closeness measure could be designed. Note that, in practice, some of these problems may need to be solved either in real-time [44], [45] or in dynamic environments [46], which means efficiency is even more important for these problems than for CARP. The idea of HD is expected to benefit the development of scalable approaches in those domains.

REFERENCES

- [1] B. L. Golden and R. T. Wong, "Capacitated arc routing problems," *Networks*, vol. 11, no. 3, pp. 305–315, 1981.
- [2] R. W. Eglese, "Routeing winter gritting vehicles," *Discr. Appl. Math.*, vol. 48, no. 3, pp. 231–244, 1994.
- [3] V. Maniezzo, "Algorithms for large directed CARP instances: Urban solid waste collection operational support," Dept. Comput. Sci., Univ. Bologna, Bologna, Italy, Tech. Rep. UBLCS-2004-16, 2004.
- [4] 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.
- [5] P. Lacomme, C. Prins, and W. Ramdane-Cherif, "Evolutionary algorithms for periodic arc routing problems," *Eur. J. Oper. Res.*, vol. 165, no. 2, pp. 535–553, 2005.
- [6] F. Chu, N. Labadi, and C. Prins, "A scatter search for the periodic capacitated arc routing problem," *Eur. J. Oper. Res.*, vol. 169, no. 2, pp. 586–605, 2006.
- [7] M. Polacek, K. F. Doerner, R. F. Hartl, and V. Maniezzo, "A variable neighborhood search for the capacitated arc routing problem with intermediate facilities," *J. Heuristics*, vol. 14, no. 5, pp. 405–423, 2008.
- [8] J. F. Campbell and A. Langevin, "Roadway snow and ice control," in *Arc Routing*. New York, NY, USA: Springer, 2000, pp. 389–418.
- [9] W. L. Pearn, "Approximate solutions for the capacitated arc routing problem," *Comput. Oper. Res.*, vol. 16, no. 6, pp. 589–600, 1989.
- [10] G. Ulusoy, "The fleet size and mix problem for capacitated arc routing," *Eur. J. Oper. Res.*, vol. 22, no. 3, pp. 329–337, 1985.
- [11] R. Hirabayashi, Y. Saruwatari, and N. Nishida, "Tour construction algorithm for the capacitated arc routing problem," *Asia Pac. J. Oper. Res.*, vol. 9, no. 2, pp. 155–175, 1992.
- [12] 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.
- [13] H. Longo, M. P. de Aragão, and E. Uchoa, "Solving capacitated arc routing problems using a transformation to the CVRP," *Comput. Oper. Res.*, vol. 33, no. 6, pp. 1823–1837, 2006.
- [14] P. Beullens, L. Muyldermans, D. Cattrysse, and D. Van Oudheusden, "A guided local search heuristic for the capacitated arc routing problem," *Eur. J. Oper. Res.*, vol. 147, no. 3, pp. 629–643, 2003.
- [15] P. Lacomme, C. Prins, and W. Ramdane-Cherif, "Competitive memetic algorithms for arc routing problems," *Ann. Oper. Res.*, vol. 131, nos. 1–4, pp. 159–185, 2004.
- [16] K. Tang, Y. Mei, and X. Yao, "Memetic algorithm with extended neighborhood search for capacitated arc routing problems," *IEEE Trans. Evol. Comput.*, vol. 13, no. 5, pp. 1151–1166, Oct. 2009.
- [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, 2008.
- [18] J. S. DeArmon, "A comparison of heuristics for the capacitated Chinese postman problem," M.S. thesis, Univ. Maryland, College Park, MD, USA, 1981.
- [19] E. Benavent, V. Campos, A. Corberan, and E. Mota, "The capacitated arc routing problem: Lower bounds," *Networks*, vol. 22, no. 7, pp. 669–690, 1992.
- [20] R. W. Eglese and L. Y. O. Li, "A tabu search based heuristic for arc routing with a capacity constraint and time deadline," in *Meta-Heuristics*. New York, NY, USA: Springer, 1996, pp. 633–649.

- [21] 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.
- [22] Y. Mei, X. Li, and X. Yao, "Decomposing large-scale capacitated arc routing problems using a random route grouping method," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Cancún, Mexico, 2013, pp. 1013–1020.
- [23] Y. Mei, X. Li, and X. Yao, "Cooperative coevolution with route distance grouping for large-scale capacitated arc routing problems," *IEEE Trans. Evol. Comput.*, vol. 18, no. 3, pp. 435–449, Jun. 2014.
- [24] Y. Mei, X. Li, and X. Yao, "Variable neighborhood decomposition for large scale capacitated arc routing problem," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Beijing, China, 2014, pp. 1313–1320.
- [25] Z. Yang, K. Tang, and X. Yao, "Large scale evolutionary optimization using cooperative coevolution," *Inf. Sci.*, vol. 178, no. 5, pp. 2985–2999, 2008.
- [26] M. N. Omidvar, X. Li, Z. Yang, and X. Yao, "Cooperative co-evolution for large scale optimization through more frequent random grouping," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Barcelona, Spain, 2010, pp. 1–8.
- [27] X. Li and X. Yao, "Cooperatively coevolving particle swarms for large scale optimization," *IEEE Trans. Evol. Comput.*, vol. 16, no. 2, pp. 210–224, Apr. 2012.
- [28] W.-F. Gao, G. G. Yen, and S.-Y. Liu, "A dual-population differential evolution with coevolution for constrained optimization," *IEEE Trans. Cybern.*, vol. 45, no. 5, pp. 1108–1121, May 2015.
- [29] R. Shang, K. Dai, L. Jiao, and R. Stolkin, "Improved memetic algorithm based on route distance grouping for multiobjective large scale capacitated arc routing problems," *IEEE Trans. Cybern.*, vol. 46, no. 4, pp. 1000–1013, Apr. 2016.
- [30] P. Lacomme, C. Prins, and W. Ramdane-Chérif, "A genetic algorithm for the capacitated arc routing problem and its extensions," in *Applications of Evolutionary Computing*. Heidelberg, Germany: Springer, 2001, pp. 473–483.
- [31] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. Math. Stat. Probab.*, Berkeley, CA, USA, 1967, pp. 281–297.
- [32] M. R. Anderberg, *Cluster Analysis for Applications*. New York, NY, USA: Academic Press, 1973.
- [33] L. Bobrowski and J. C. Bezdek, "C-means clustering with the L1 and L_∞ norms," *IEEE Trans. Syst., Man Cybern.*, vol. 21, no. 3, pp. 545–554, May/Jun. 1991.
- [34] S. S. Khan and A. Ahmad, "Cluster center initialization algorithm for K-means clustering," *Pattern Recognit. Lett.*, vol. 25, no. 11, pp. 1293–1302, 2004.
- [35] G. Dueck and T. Scheuer, "Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing," *J. Comput. Phys.*, vol. 90, no. 1, pp. 161–175, 1990.
- [36] M. Polacek, R. F. Hartl, K. Doerner, and M. Reimann, "A variable neighborhood search for the multi depot vehicle routing problem with time windows," *J. Heuristics*, vol. 10, no. 6, pp. 613–627, 2004.
- [37] J. Luo, X. Li, M.-R. Chen, and H. Liu, "A novel hybrid shuffled frog leaping algorithm for vehicle routing problem with time windows," *Inf. Sci.*, vol. 316, pp. 266–292, Sep. 2015.
- [38] L. Tan, F. Lin, and H. Wang, "Adaptive comprehensive learning bacterial foraging optimization and its application on vehicle routing problem with time windows," *Neurocomputing*, vol. 151, pp. 1208–1215, Mar. 2015.
- [39] Z. Xiao and Z. Ming, "A method of workflow scheduling based on colored Petri nets," *Data Knowl. Eng.*, vol. 70, no. 2, pp. 230–247, 2011.
- [40] Z. Chen, M. Qiu, Z. Ming, L. T. Yang, and Y. Zhu, "Clustering scheduling for hardware tasks in reconfigurable computing systems," *J. Syst. Architect.*, vol. 59, no. 10, pp. 1424–1432, 2013.
- [41] J. Li *et al.*, "Thermal-aware task scheduling in 3D chip multiprocessor with real-time constrained workloads," *ACM Trans. Embedded Comput. Syst.*, vol. 12, no. 2, 2013, Art. no. 24.
- [42] Z. Zhu, J. Xiao, J.-Q. Li, F. Wang, and Q. Zhang, "Global path planning of wheeled robots using multi-objective memetic algorithms," *Integr. Comput.-Aided Eng.*, vol. 22, no. 4, pp. 387–404, 2015.
- [43] A. Macwan, J. Vilela, G. Nejat, and B. Benhabib, "A multirobot path-planning strategy for autonomous wilderness search and rescue," *IEEE Trans. Cybern.*, vol. 45, no. 9, pp. 1784–1797, Sep. 2015.
- [44] H. Luo, J. Fang, and G. Q. Huang, "Real-time scheduling for hybrid flowshop in ubiquitous manufacturing environment," *Comput. Ind. Eng.*, vol. 84, pp. 12–23, Jun. 2015.
- [45] J. Li *et al.*, "Online optimization for scheduling preemptable tasks on IaaS cloud systems," *J. Parallel Distrib. Comput.*, vol. 72, no. 5, pp. 666–677, 2012.
- [46] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Automatic programming via iterated local search for dynamic job shop scheduling," *IEEE Trans. Cybern.*, vol. 45, no. 1, pp. 1–14, Jan. 2015.



Ke Tang (S'05–M'07–SM'13) received the B.Eng. degree from the Huazhong University of Science and Technology, Wuhan, China, in 2002, and the Ph.D. degree from Nanyang Technological University, Singapore, in 2007.

Since 2007, he has been with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, China, where he is currently a Professor. He has authored/co-authored over 100 refereed publications. His current research interests include evolutionary computation and machine learning and their real-world applications.

Dr. Tang was a recipient of the Royal Society Newton Advanced Fellowship. He is an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, the *IEEE Computational Intelligence Magazine*, and *Computational Optimization and Applications* (Springer), and on the editorial board of a few other journals. He is a member of the IEEE Computational Intelligence Society (CIS) Evolutionary Computation Technical Committee and the IEEE CIS Emergent Technologies Technical Committee.



Juan Wang received the Ph.D. degree from the University of Science and Technology of China, Hefei, China, in 2016.

She is currently an Algorithm Engineer with a company in Shenzhen, China. Her current research interests include memetic algorithm and metaheuristics for capacitated arc routing problems.



Xiaodong Li (M'03–SM'07) received the B.Sc. degree from Xidian University, Xi'an, China, and the Ph.D. degree in information science from the University of Otago, Dunedin, New Zealand.

He is currently an Associate Professor with the School of Computer Science and Information Technology, RMIT University, Melbourne, VIC, Australia. His current research interests include evolutionary computation, neural networks, complex systems, multiobjective optimization, and swarm intelligence.

Dr. Li was a recipient of the 2013 ACM SIGEVO Impact Award. He serves as an Associate Editor for the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, *Swarm Intelligence* (Springer), and the *International Journal of Swarm Intelligence Research*. He is a Founding Member and currently the Vice-Chair of the IEEE Computational Intelligence Society (CIS) Task Force on Swarm Intelligence, and the Vice-Chair of the IEEE Task Force on Multimodal Optimization, and the Former Chair of the IEEE CIS Task Force on Large Scale Global Optimization. He was the General Chair of the SEAL'08, the Program Co-Chair of AI'09 and the IEEE CEC'2012.



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.

He is a Chair Professor of Computer Science and the Director of the Centre of Excellence for Research in Computational Intelligence and Applications with the University of Birmingham, Birmingham, U.K. He published over 200 refereed international journal papers. His current research interests include evolutionary computation and ensemble learning. He has been publishing frequently on the topic of capacitated arc routing problems since his first paper in 2006.

Prof. Yao was a recipient of the 2001 IEEE Donald G. Fink Prize Paper Award, the 2015 and 2010 IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION Outstanding Paper Awards, the 2010 BT Gordon Radley Award for Best Author of Innovation (Finalist), the 2011 IEEE TRANSACTIONS ON NEURAL NETWORKS Outstanding Paper Award, the prestigious Royal Society Wolfson Research Merit Award in 2012, the IEEE Computational Intelligence Society (CIS) Evolutionary Computation Pioneer Award in 2013, and many other best paper awards. He was an Editor-in-Chief of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION. He is the President of the IEEE CIS from 2014 to 2015.