Divide-and-Conquer Large Scale Capacitated Arc Routing Problems with Route Cutting Off Decomposition

Yuzhou Zhang[a], Yi Mei[b,*], Buzhong Zhang[a], Keqin Jiang[a]

[a]*School of Computer and Information, Anqing Normal University, Anqing 246133, China*
[b]*School of Engineering and Computer Science, Victoria University of Wellington, Kelburn 6012, New Zealand*

Abstract

The capacitated arc routing problem is a very important problem with many practical applications. This paper focuses on the large scale capacitated arc routing problem. Traditional solution optimization approaches usually fail because of their poor scalability. The divide-and-conquer strategy has achieved great success in solving large scale optimization problems by decomposing the original large problem into smaller sub-problems and solving them separately. For arc routing, a commonly used divide-and-conquer strategy is to divide the tasks into subsets, and then solve the sub-problems induced by the task subsets separately. However, the success of a divide-and-conquer strategy relies on a proper task division, which is non-trivial due to the complex interactions between the tasks. This paper proposes a novel problem decomposition operator, named the route cutting off operator, which considers the interactions between the tasks in a sophisticated way. To examine the effectiveness of the route cutting off operator, we integrate it with two state-of-the-art divide-and-conquer algorithms, and compared with the original counterparts on a wide range of benchmark instances. The results show that the route cutting off operator can improve the effectiveness of the decomposition, and lead to significantly better results especially when the problem size is very large and the time budget is very tight.

Keywords: Capacitated arc routing problem, route cutting off, large scale optimization, divide-and-conquer

1. Introduction

The Capacitated Arc Routing Problem (CARP) is a very important combinatorial optimization problem with a wide range of real-world applications such as winter gritting [15], mail delivery [11], urban waste collection [24, 49, 7], and snow removal [30]. First presented by Golden and Wong [14], it seeks an optimal set of routes (e.g. cycles starting and ending at a depot) for a fleet of vehicles to serve the edges in a graph subject to certain constraints. There have been extensive studies for solving CARP, and a large number of competitive approaches have been proposed, e.g. [40, 5, 17, 18, 37, 12, 25].

In real world, the problem size of CARP can usually be very large (e.g. thousands of streets in a city need to be served for waste collection). It has been demonstrated that most existing approaches exploring the entire search space of the original problem have poor scalability, i.e. their effectiveness deteriorate rapidly as the problem size grows [20, 22, 21]. As a result, most approaches that are competitive for the small and medium instances cannot solve Large Scale CARP (LSCARP) effectively.

*Corresponding author
Email address: yi.mei@ecs.vuw.ac.nz (Yi Mei)

The divide-and-conquer strategy is a promising approach to address the scalability issue of algorithms when solving large scale optimization problems. The main idea is to decompose the original large problem into smaller sub-problems that can be solved individually. There have been several divide-and-conquer approaches proposed for solving LSCARP, such as the Route Distance Grouping (RDG) decomposition [21] and Hierarchical Decomposition (HD) [38], which have achieved great success in finding competitive solutions efficiently. The main idea of these approaches is to divide the tasks (i.e. required edges) into subsets, solve the sub-problems induced by the subsets separately, and finally combine the solutions to the sub-problems (subsets of routes) together to form the solution to the original problem.

For designing divide-and-conquer approaches for LSCARP, the problem decomposition is the key step. In an *ideal* decomposition, the subsets of tasks are completely independent of each other, and the combination of the optimal solutions to the corresponding sub-problems (union of the routes) can result in the optimal solution to the original problem. However, it is very challenging to identify the ideal decomposition due to the complex interactions between the tasks, if not impossible. Therefore, all the existing divide-and-conquer approaches (e.g. [21, 32, 34, 33, 38]) adopt the *adaptive* decomposition. They start with randomly or heuristically generated task subsets, and gradually improve the decomposition during the search process based on the updated information.

Intuitively, the quality of a decomposition depends on two major factors. First, the tasks belonging to the same route in the optimal solution should be in the same subset to prevent from breaking an optimal route. Second, the tasks that are close to each other should be more likely to be in the same subset, so that solving the corresponding sub-problem can lead to better solutions (i.e. shorter routes). It is challenging to consider these two factors simultaneously. Although there have been some effort from existing approaches [21, 38], there is still great potential for improvement.

The goal of this paper is to propose a new decomposition scheme to consider the above two factors in a sophisticated way during the search process. More specifically,

- we design a *task rank matrix* according to the distances between different tasks and define *good links* and *poor links* between tasks based on the task rank matrix;
- we develop a new Route Cutting Off (RCO) decomposition operator, which is more likely to decompose a route by breaking poor links rather than good links;
- we develop two new divide-and-conquer algorithms by embedding the RCO operator into two state-of-the-art algorithms (i.e. RDG-MAENS [21] and SAHiD [38]);
- we verify the effectiveness of the proposed RCO operator by comparing the newly developed algorithms with their original counterpart and other state-of-the-art approaches on a range of LSCARP instances.

The remainder of this paper is organized as follows. First, Section 2 gives the background, including the problem description and related work. Then, the proposed RCO operator is introduced in Section 3. Section 4 presents the experimental studies and discussions on the results. Finally, the conclusions and future work are given in Section 5.

2. Background

In this section, we briefly describe the problem statement and related work.

2.1. Capacitated Arc Routing Problem

Given an undirected graph $G(V; E)$, where V and E represent the vertex and edge sets. Each edge $e \in E$ has a demand $d(e)$ 0, a non-negative service cost $sc(e)$ 0 and a non-negative deadheading cost $dc(e)$ 0. An edge with

a positive demand is called a *required edge* or a *task*. The set of all the tasks is denoted as $T = \{e \in E : d(e) > 0\} \subseteq E$. A fleet of vehicles with a limited capacity Q is located at the *depot* $v_0 \in V$ to serve the tasks. CARP is to design a set of least-cost routes for the vehicles to serve all the tasks subject to the following constraints:

each route starts and ends at the depot (i.e. is a cycle);

each task is served exactly once by a vehicle;

(*capacity constraint*) the total demand served by each vehicle cannot exceed its capacity.

Under the task representation [39], each task (u, v) can be represented by two IDs, each representing one of its directions. In addition to the demand, service cost and deadheading cost of the corresponding task, each ID t is associated with a *head vertex* $hv(t)$, a *tail vertex* $tv(t)$ and a inverse ID $inv(t)$. Specifically, for task (u, v) and its two IDs t_1 and t_2, we have $hv(t_1) = tv(t_2) = u$, $tv(t_1) = hv(t_2) = v$, $t_2 = inv(t_1)$ and $t_1 = inv(t_2)$.

A CARP solution S can be represented as a set of routes, i.e. $S = \{S_1, \ldots, S_{|S|}\}$, where S_k $(k = 1, \ldots, |S|)$ is the kth route. Each route S_k is represented as a sequence of task IDs $S_k = (t_{k1}, \ldots, t_{k|S_k|})$.

The total cost of S is calculated as follows:

$$tc(S) = \sum_{k=1}^{|S|} \left[\sum_{i=1}^{|S_k|} sc(S_k[i]) + \sum_{i=1}^{|S_k|-1} \left(tv(S_k[i]), hv(S_k[i+1]) \right) \right], \tag{1}$$

where $S_k[i]$ stands for the ith element (task ID) in S_k. (u, v) indicates the cost of the shortest path from vertices u to v, which can be calculated by Dijkstra's algorithm [9] beforehand.

Then, CARP can be formulated as follows:

$$\min \quad tc(S), \tag{2}$$

$$s.t. \quad S_k[1] = S_k[|S_k|] = t_0, \forall k = 1, \ldots, |S|, \tag{3}$$

$$\sum_{k=1}^{|S|} (|S_k| - 2) = |T|, \tag{4}$$

$$S_k[i] \neq S_{k'}[i'], \quad \forall 1 \le k, k' \le |S|, 2 \le i, i' \le |S_k| - 1, k \neq k' \text{ or } i \neq i', \tag{5}$$

$$S_k[i] \neq inv(S_{k'}[i']), \quad \forall 1 \le k, k' \le |S|, 2 \le i, i' \le |S_k| - 1, k \neq k' \text{ or } i \neq i', \tag{6}$$

$$\sum_{i=2}^{|S_k|-1} d(S_k[i]) \le Q, \quad \forall 1 \le k \le |S|, \tag{7}$$

$$S_k[i] \in T, \quad \forall k = 1, \ldots, |S|, i = 2, \ldots, |S_k| - 1. \tag{8}$$

where in Eq. (3), t_0 is a dummy task ID representing the depot loop, i.e. $hv(t_0) = tv(t_0) = v_0$, $d(t_0) = sc(t_0) = dc(t_0) = 0$ and $inv(t_0) = t_0$. Eq. (2) is the objective function, which is to minimize the total cost calculated by Eq. (1). Eq. (3) specifies that in each route S_k, the first and last element must be the depot loop, indicating that each route starts and ends at the depot. Eq. (4) means that the total number of task IDs served by all the routes (excluding the first and last elements which are the depot loop) equals the total number of tasks. Eqs. (5) and (6) implies that any two task IDs served at different positions of the routes belong to different tasks. Therefore, Eqs. (4)–(6) guarantee that each task is served exactly once. Eq. (7) indicates that the total demand served by each route is no greater than its capacity. Eq. (8) defines the domain of the elements of each route, i.e. except the first and last element, all the other elements must belong to the task set.

2.2. Related Work

Since presented in 1981 [14], CARP has received a lot of research interests over the past decades, and a variety of competitive algorithms ranging from mathematical programming (e.g. [3, 1, 31, 2, 19]) and heuristic solution search approaches (e.g. [17, 5, 26, 18, 6, 23, 37, 12, 46, 35]) have been proposed for solving it. However, most of the

early studies focused on small and medium scaled problems, and only tested on small and medium-sized benchmark instances. For example, the most commonly used gdb [8], val [4], egl [10] and Beullens' benchmark sets [5] have no more than 190 tasks.

In 2008, Brandão and Eglese [6] generated a large scale dataset named EGL-G, in which the number of tasks was increased to 375. Since then, the research interests gradually shifted to the scalability of the approach, and more and more studies were conducted specifically to tackle LSCARP. For example, Brandão and Eglese [6] proposed a tabu search and achieved promising results on the EGL-G instances. Mei et al. [23] proposed a tabu search with a global repair operator, and Martinelli et al. [19] proposed an Iterative Local search based on Random Variable Neighbourhood Descent (ILS-RVND), both of which improved the upper bounds of the EGL-G dataset. Vidal [41] proposed an unified hybrid genetic search (UHGS) which outperformed all the existing algorithms on the EGL-G dataset.

As the problem size grows, solving the problem as a whole becomes much less effective, and divide-and-conquer strategy can be a promising technique in this case. The divide-and-conquer strategy has achieved great success in a range of problems, such as continuous optimization [45, 36, 13], vehicle routing [29, 28] and job shop scheduling [48, 27]. Also for LSCARP, there have been a variety of divide-and-conquer approaches [20, 21, 32, 34, 33] based on decomposing the problem into smaller sub-problems by grouping the routes of the best-so-far solution. Specifically, in these algorithms, the entire search process is divided into cycles. At the beginning of each cycle, the routes of the best-so-far solution are grouped together based on different strategies (e.g. randomly or by clustering methods). They have achieved much better results than the methods without divide-and-conquer.

In 2017, based on two major Chinese cities, i.e. Beijing and Hefei, Tang et al. [38] created two real-world LSCARP datasets, and extended the number of tasks to over 3000. The Beijing and Hefei datasets are much larger than the EGL-G dataset. The existing divide-and-conquer approaches are not effective enough to solve them. Tang et al. [38] proposed a new hierarchical decomposition based on the concept of "virtual task". A virtual task is a sequence of tasks. Starting from elementary tasks, the hierarchical decomposition recursively concatenates (virtual) tasks together to form new higher-level virtual tasks. The resultant algorithm, named SAHiD, managed to obtain much better results than other divide-and-conquer approaches (e.g. RDG-MAENS [21]) within a limited time budget.

In 2017, Kiilerich and Wøhlk [16] generated a very large scale dataset (denoted as the KW set) based on the five countries in Denmark. In the KW dataset, the number of tasks is further enlarged to over 8000. The KW set is the largest dataset by far. Wøhlk et al. [43] proposed a fast heuristic named *Fast-CARP* to solve the LSCARP, in which the whole problem is partitioned into a number of districts and each district is then optimized independently. Fast-CARP can be regarded as a divide-and-conquer approach.

In summary, divide-and-conquer approaches have achieved great success in solving LSCARP effectively. However, the current approaches still have limitations. They focused on the interactions between the tasks in different routes of the best-so-far solution during the decomposition, while the interactions between tasks within each route were neglected. In this paper, we aim to consider the interactions both in different routes and within the same route, and propose the RCO operator for this purpose.

3. Proposed Route Cutting Off Decomposition

In an adaptive decomposition approach, the entire search process consists of a number of cycles. At the beginning of each cycle, a new decomposition is generated based on the latest information, e.g. typically the best-so-far solution. Specifically, the decomposition tends to assign two tasks into the same subset if (1) the two tasks are close to each

other and (2) the two tasks are in the same route of the best-so-far solution. To achieve this, existing works considered to cluster the routes or sub-routes of the best-so-far solution.

When clustering the routes together (e.g. [21, 32, 33]), the advantage is that the optimal solution under the new decomposition is guaranteed to be no worse than the current best-so-far solution. However, it cannot identify and take advantage of the *patterns* within each route. For example, it is inevitable to have both "*good links*" (connecting the tasks that are close to each other) and "*poor links*" (connecting the tasks that are distant from each other) in a route, especially in the early stage of the search. Different links are not distinguished when the routes are clustered as a whole.

On the other hand, one can split a route by breaking the poor links. This way, the links between tasks can potentially be treated more properly, although the monotonically non-increasing decomposition cannot be guaranteed. That is, the optimal solution under the new decomposition may be worse than the current best-so-far solution. However, one may identify better decomposition more efficiently. For example, [38] randomly split a route into two sub-routes during the decomposition, and managed to obtain much better results within a limited time budget than the approaches that cluster the whole routes (e.g. [21]). [44] preserved the promising links by counting the total number of appearances of links and breaking those with small appearances.

In this paper, we investigate the *link patterns* within a route more systematically, and propose a *task rank matrix* to represent such patterns. Then, to further improve the effectiveness and efficiency of divide-and-conquer, we propose the RCO decomposition operator based on the task rank matrix.

3.1. Task Rank Matrix

First, we define a *link* from one task to another as the shortest path from the former task to the latter task. The direction of the two tasks are discarded. The cost of the link from task t_1 to task t_2 is defined as:

$$\Delta(t_1; t_2) = \frac{1}{4} \quad (hv(t_1); hv(t_2)) + \quad (hv(t_1); tv(t_2)) + \quad (tv(t_1); hv(t_2)) + \quad (tv(t_1); tv(t_2)) \quad ; \tag{9}$$

where $(v_1; v_2)$ indicates the cost of the shortest path from vertices v_1 to v_2. The quality of a link depends not only on its absolute cost (i.e. how close the two tasks are to each other), but also on the relative cost to other relevant links (i.e. how close they are *in comparison with other alternative tasks*). For example, if a task is isolated and is far away from all the other tasks, then all the links from the task have large costs. However, a link from the isolated task should still be considered as "good" if its cost is much smaller than that of the other links from this isolated task.

Based on the above intuition, we define the *task rank matrix* ($\mathbf{\Gamma}$) to indicate the quality of the links from each task. Given a set of tasks T, the entry $\mathbf{\Gamma}_{t_1;t_2}$ represents the rank of the link from task t_1 to task t_2, which is calculated based on the links from t_1 to all the other tasks.

An example is given in Fig. 1 to show how to calculate the task rank matrix. In the figure, v_0 is the depot. There are 8 tasks (represented by the solid lines) require to be served, and the shortest paths between the tasks are represented as dashed lines. The service cost and deadheading cost of each task are 1, and the cost of the shortest paths between the tasks are given next to the dashed lines.

For the graph shown in Fig. 1, we first calculate the matrix of the link costs in Eq. (10). Then for each row of Eq. (10), we assign the ranks to the links based on their costs, e.g. the link with the lowest cost is given rank 1. If multiple links have the same cost, then they share the same rank. Eq. (11) shows the task rank matrix obtained from Eq.(10).

From the first row of Eq. (11), one can see that the links from $(v_0; v_1)$ to $(v_0; v_8)$ and $(v_0; v_{10})$ are both of rank 1 among all the links from $(v_0; v_1)$, and the link to $(v_2; v_3)$ is of rank 7, since $(v_2; v_3)$ is the farthest task from $(v_0; v_1)$. Note that the task rank matrix is not symmetric. For $(v_0; v_1)$, the task $(v_2; v_3)$ has a very low priority (rank 7), since

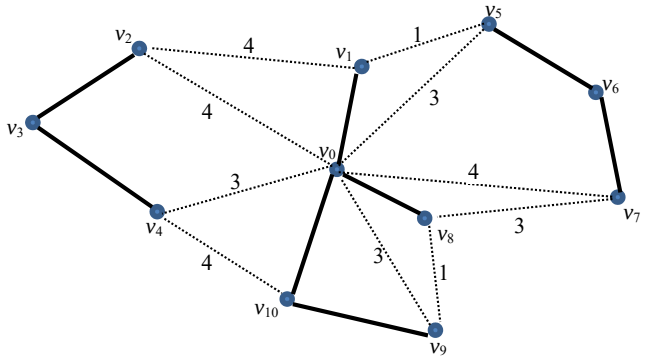

Figure 1: An example to show the calculation of the task rank matrix.

$$
= \begin{array}{c} \\ (v_0;v_1) \\ (v_0;v_8) \\ (v_0;v_{10}) \\ (v_2;v_3) \\ (v_3;v_4) \\ (v_5;v_6) \\ (v_6;v_7) \\ (v_9;v_{10}) \end{array}
\begin{array}{c}
\begin{array}{cccccccc} (v_0;v_1) & (v_0;v_8) & (v_0;v_{10}) & (v_2;v_3) & (v_3;v_4) & (v_5;v_6) & (v_6;v_7) & (v_9;v_{10}) \end{array} \\
\begin{pmatrix}
 & 4{=}4 & 4{=}4 & 18{=}4 & 16{=}4 & 8{=}4 & 12{=}4 & 8{=}4 \\
4{=}4 & & 4{=}4 & 20{=}4 & 16{=}4 & 12{=}4 & 14{=}4 & 6{=}4 \\
4{=}4 & 4{=}4 & & 18{=}4 & 16{=}4 & 12{=}4 & 16{=}4 & 4{=}4 \\
18{=}4 & 20{=}4 & 18{=}4 & & 4{=}4 & 24{=}4 & 28{=}4 & 22{=}4 \\
16{=}4 & 16{=}4 & 16{=}4 & 4{=}4 & & 24{=}4 & 28{=}4 & 20{=}4 \\
8{=}4 & 12{=}4 & 12{=}4 & 24{=}4 & 24{=}4 & & 4{=}4 & 16{=}4 \\
12{=}4 & 14{=}4 & 16{=}4 & 28{=}4 & 28{=}4 & 4{=}4 & & 18{=}4 \\
8{=}4 & 6{=}4 & 4{=}4 & 22{=}4 & 20{=}4 & 16{=}4 & 18{=}4 &
\end{pmatrix}
\end{array} : \tag{10}
$$

$$
= \begin{array}{c} \\ (v_0;v_1) \\ (v_0;v_8) \\ (v_0;v_{10}) \\ (v_2;v_3) \\ (v_3;v_4) \\ (v_5;v_6) \\ (v_6;v_7) \\ (v_9;v_{10}) \end{array}
\begin{array}{c}
\begin{array}{cccccccc} (v_0;v_1) & (v_0;v_8) & (v_0;v_{10}) & (v_2;v_3) & (v_3;v_4) & (v_5;v_6) & (v_6;v_7) & (v_9;v_{10}) \end{array} \\
\begin{pmatrix}
 & 1 & 1 & 7 & 6 & 3 & 5 & 3 \\
1 & & 1 & 7 & 6 & 4 & 5 & 3 \\
1 & 1 & & 7 & 5 & 4 & 5 & 1 \\
2 & 4 & 2 & & 1 & 6 & 7 & 5 \\
2 & 2 & 2 & 1 & & 6 & 7 & 5 \\
2 & 3 & 3 & 6 & 6 & & 1 & 5 \\
2 & 3 & 4 & 6 & 6 & 1 & & 5 \\
3 & 2 & 1 & 7 & 6 & 4 & 5 &
\end{pmatrix}
\end{array} : \tag{11}
$$

there are many other closer tasks for $(v_0;v_1)$ to consider. On the contrary, for $(v_2;v_3)$, $(v_0;v_1)$ is a very promising task to go to (rank 2).

3.2. Route Cutting Off Decomposition Operator

Based on the task rank matrix, we propose the new RCO operator for decomposition. Briefly speaking, given a best-so-far solution, the RCO operator splits the routes of the solution based on the task rank matrix, to generate a pool of sub-routes. Clustering these sub-routes is expected to obtain more promising decomposition than the existing decomposition methods.

The pseudo code of the RCO operator is described in Algorithm 1. Given the best-so-far solution S, the average task rank $\bar{}(S)$ of the links in S is first calculated based on the task rank matrix $\mathbf{\Gamma}$ (line 2). For each route S_k, we categorize the links into "good links" and "poor links" (lines 3–22). Here we adopt a simple rule for the categorization. A link is considered to be good if its rank is smaller than $\bar{}(S)$, and poor otherwise. Then, we randomly cut off a good

link with probability , and a poor link with probability (lines 11–20). The corresponding sub-routes are finally inserted into Ω. Fig. 2 shows an example solution to the graph shown in Fig. 1, where the tasks are denoted as x_1 to x_8. Only the links between the tasks in the same route are considered (e.g. there is no link from x_3 to x_4). Based on Eq. (11), the ranks of the 5 links are $\mathbf{\Gamma}_{x_1;x_2} = 7$, $\mathbf{\Gamma}_{x_2;x_3} = 1$, $\mathbf{\Gamma}_{x_4;x_5} = 5$, $\mathbf{\Gamma}_{x_5;x_6} = 1$, and $\mathbf{\Gamma}_{x_7;x_8} = 1$. In this case, the average task rank of the solution is $(7+1+5+1+1)/5 = 3$. $\langle x_2; x_3\rangle$, $\langle x_5; x_6\rangle$ and $\langle x_7; x_8\rangle$ are good links, while $\langle x_1; x_2\rangle$ and $\langle x_4; x_5\rangle$ are poor links.

Algorithm 1 RCO($S; \mathbf{\Gamma};$;)

Input: The best-so-far solution S, the task rank matrix $\mathbf{\Gamma}$, cutting probabilities , ;

Output: A set of sub-routes Ω;

1: Set $\Omega =$;;
2: Calculate the average task rank $\bar{}(S)$ of S based on $\mathbf{\Gamma}$;
3: **for** each route S_k of S **do**
4: Set the good links $GL =$;, and the poor links $PL =$;;
5: **for** each link $\langle S_k[i]; S_k[i+1]\rangle$ in S_k **do**
6: **if** $\mathbf{\Gamma}_{t;t'} <$ $\bar{}(S)$ **then**
7: $GL = GL \cup \langle S_k[i]; S_k[i+1]\rangle$;
8: **else**
9: $PL = PL \cup \langle S_k[i]; S_k[i+1]\rangle$;
10: **end if**
11: Set cut-off good link $gl = null$, cut-off poor link $pl = null$;
12: Randomly generate a number $r_1 \in [0; 1]$;
13: **if** $r_1 <$ **then**
14: Randomly select the cut-off good link gl from GL;
15: **end if**
16: Randomly generate a number $r_2 \in [0; 1]$;
17: **if** $r_2 <$ **then**
18: Randomly select the cut-off poor link pl from PL;
19: **end if**
20: Cut off gl and pl in S_k to obtain sub-routes $\{S_{k;1}; \ldots\}$;
21: $\Omega = \Omega \cup \{S_{k;1}; \ldots\}$;
22: **end for**
23: **end for**
24: **return** Ω;

3.3. Divide-and-Conquer using Route Cutting Off

Given the best-so-far solution, the proposed RCO operator identifies promising cutting points to split the routes of the solution into promising sub-routes for clustering. In other words, the RCO operator is a generic decomposition operator, and can be embedded into any divide-and-conquer algorithm based on clustering the routes/sub-routes.

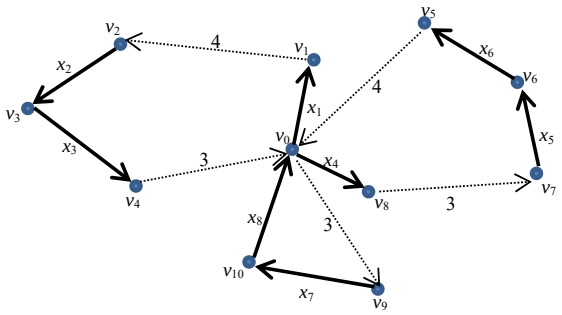

Figure 2: A solution with three routes to the example graph in Fig. 1.

In this paper, we embed the RCO operator into two state-of-the-art algorithms, i.e. RDG-MAENS [21] and SAHiD [38] to verify the effectiveness of the RCO operator. The resultant algorithms are named RCO-RDG-MAENS and RCO-SAHiD, respectively.

The pseudo code of RCO-RDG-MAENS is described in Algorithm 2. The algorithm is the same as the original one [21], except that instead of clustering the routes directly, the RCO operator is used to split the best-so-far solution before clustering (line 6).

Algorithm 2 The pseudo code of RCO-RDG-MAENS

Input: The input instance with task set T, group number g, RCO parameters and ;

Output: The best-found feasible solution S^*;

1: Calculate the task rank matrix $\mathbf{\Gamma}$;
2: Initialize a population $pop(T)$;
3: Evaluate each individual in $pop(T)$;
4: Set $S^* = \arg\min_{S \in pop(T)} ftc(S)g$;
5: **for** $cycle = 1 \ ! \ max\_cycle$ **do**
6: $\Omega = \mathtt{RCO}(S^*; \mathbf{\Gamma}; \ ; \ )$;
7: Cluster the sub-routes in Ω using the fuzzy k-medoid method in RDG [21] to obtain $fT_1; T_2; : : : ; T_g g$;
8: **for** $i = 1 \ ! \ g$ **do**
9: $pop(T_i) = \mathtt{pop2subpop}(pop(T); T_i)$;
10: Evolve $pop(T_i)$ by MAENS [37];
11: **end for**
12: $pop(T) = \mathtt{subpop2pop}(pop(T_1); : : : ; pop(T_g))$;
13: $S' = \arg\min_{S \in pop(T)} ftc(S)g$;
14: **if** $tc(S') < tc(S^*)$ **then**
15: $S^* = S'$;
16: **end if**
17: **end for**
18: **return** S^*;

Likewise, the pseudo code of RCO-SAHiD is described in Algorithm 3, where the key difference from SAHiD [38]

is that the routes of the current solution S are split by RCO (line 6), while they are randomly split in SAHiD.

Algorithm 3 The pseudo code of RCO-SAHiD

Input: The input instance with task set T, RCO parameters and

Output: A best feasible solution S^*;

1: Calculate the task rank matrix $\mathbf{\Gamma}$;
2: Generate an initial solution S with HDU(T) [38] operator;
3: Improve S with local search procedure;
4: Set $S^* = S$;
5: **while** termination conditions are not satisfied **do**
6: $\Omega = \mathtt{RCO}(S; \mathbf{\Gamma};\ ;\ )$;
7: Construct virtual task set VT based on the sub-routes in Ω;
8: Call HDU(VT) to generate a solution S^0;
9: Improve S^0 with local search procedure;
10: **if** S^0 is acceptable **then**
11: $S = S^0$;
12: **if** $tc(S^0) < tc(S^*)$ **then**
13: $S^* = S^0$;
14: **end if**
15: **end if**
16: **end while**
17: **return** S^*;

4. Experimental Studies

To evaluate the effectiveness of the proposed RCO operator for LSCARP, we conduct experiments to compare RCO-RDG-MAENS and RCO-SAHiD with their original counterparts as well as other state-of-the-art algorithms on a range of LSCARP instances. In addition to RDG-MAENS and SAHiD, we also compare with VNS [30], TSA1 [6], ILS-RVND [19], IRDG-MAENS [32], QICA-CARP [34], ESMEANS [33], Fast-CARP [43] and PS [47].

4.1. Datasets

Since our work focuses on LSCARP, we select the four existing LSCARP datasets, i.e. the EGL-G [6], Hefei [38], Beijing [38] and KW [16] datasets. The EGL-G dataset consists of 10 instances, which are derived from a real-world road network of Lancashire, UK, with 255 nodes and 375 edges. The dataset contains two groups G1 and G2, each with 5 instances. The instances belonging to the same group have the same task set, and different vehicle capacities. The Hefei dataset contains 10 instances based on a road network in Hefei, China, with 850 nodes and 1212 edges. The instances have the same vehicle capacity, but vary in their task sets. Similarly, the Beijing dataset consists of 10 instances sharing the same road network in Beijing, China (2820 nodes and 3584 edges) and vehicle capacity. The KW dataset consists of 264 CARP benchmark instances generated from 88 graphs by varying the vehicle capacity. In the KW dataset, the largest instance contains 11640 nodes, 12675 edges, and 8581 required edges. Overall, EGL-G is the smallest LSCARP dataset. Hefei and Beijing are much larger than EGL-G, and KW is the largest dataset.

4.2. Experiment Design

Since different datasets have different available results, we designed three experimental comparisons as follows.

Experiment 1: compare *RCO-RDG-MAENS* with RDG-MAENS [21], ILS-RVND [19], IRDG-MAENS [32], QICA-CARP [34] and ESMAENS [33] on the EGL-G dataset.

Experiment 2: compare *RCO-SAHiD* with SAHiD [38], UHGS [41], RDG-MAENS [21], VNS [30] and TSA1 [6] on the Hefei and Beijing datasets.

Experiment 3: compare *RCO-SAHiD* with SAHiD [38], UHGS [41], Fast-CARP [43] and PS [47] on some large KW instances (task number ranging 7831 to 8581).

In each experiment, we tried our best to compare with all the algorithms whose results are available in literature for the corresponding datasets. For each instance, each algorithm was run multiple times independently, and the Wilcoxon rank sum test [42] was conducted to test the results statistically.

To make fair comparisons, we set the algorithm parameters consistent with the settings in the literature. Specifically, in Experiment 1, following the parameter settings in [21], in RCO-RDG-MAENS, we set the population size to 30, offspring population size to 180, maximum generation number to 500, number of cycles to 50, and probability of local search to 0.2. The RDG operator has two parameters: the number of groups g and degree of fuziness (used by the fuzzy k-medoid clustering). We set $g = 2$ and $= 5$ for both RCO-RDG-MAENS and RDG-MAENS, as they showed the best performance [21].

In Experiment 2, following the parameter settings in [38], in RCO-SAHiD, the scale parameter in HD is 0.1, the threshold for accepting a worse solution is 110%, and the maximum number of idle iterations for accepting an ascending move is 10000. Note that in Experiment 2, the stopping criterion of RCO-SAHiD and UHGS is the runtime i.e. after 30 minutes. The runtime depends on a variety of factors such as CPU frequency, RAM, operating system, coding language and compiler. In our experiments, we implemented RCO-SAHiD based on the original SAHiD source code to make sure they share the same programming language and compiler. To improve fairness, a common approach used by previous studies (e.g.[24, 49, 37, 23, 19, 21]) is to scale the runtime based on the CPU frequency. In this paper, we adopt the same scaling approach. RCO-SAHiD and UHGS were run on Intel(R) Xeon(R) E5-2650 v2 with 2.6 GHz, 64GBs RAM, and the other compared algorithms were run on Intel Core i7-4790 with 3.6 GHz. Therefore, the maximum runtime for RCO-SAHiD and UHGS was set to 3.6=2.6 30 60 = 2492 seconds.

In Experiment 3, the parameters of RCO-SAHiD are the same as in Experiment 2. The runtime follows the configuration of the original literature of Fast-CARP [43], which is one minute per 1000 nodes. Fast-CARP was run on an Intel Xeon CPU with 3.5 GHz, while RCO-SAHiD, SAHiD and UHGS were run on Intel(R) Xeon(R) E5-2650 v2 with 2.6 GHz, 64GBs RAM. The runtime of these three algorithms were scaled to 3.5=2.6 60 = 81 seconds per 1000 nodes.

4.3. Parameter Sensitivity Analysis

The RCO operator has two important parameters, namely the probability of cutting good links and the probability of cutting poor links . Intuitively, it is more promising to cut poor links than good links. On the other hand, cutting good links can increase the exploration capability of the algorithm, and help the search jump out of the current local optimum. Based on the above consideration, we should set to a small value, and to a relatively large value. To analyze the sensitivity of and , we conducted some pilot experiments by running RCO-RDG-MAENS with = 0.05 and 0.1, and = 0.1, 0.2 and 0.3 (six combinations in total) on the EGL-G dataset. Each algorithm was run 30 times independently on each instance.

Fig. 3 and Fig. 4 show the average total cost and computational time of RCO-RDG-MEANS with different values of and over the 10 EGL-G instances. From the figures, one can see that for all the instances, there is no significant difference among the different and values in terms of the average total cost. However, the and values have some impact on the computational time. The algorithm with = 0:1 and = 0:3 usually had the longest computational time. All the other five versions had similar computation time. Overall, the algorithm with (;) = (0:05; 0:2) seems to have a short computational time over all the instances.

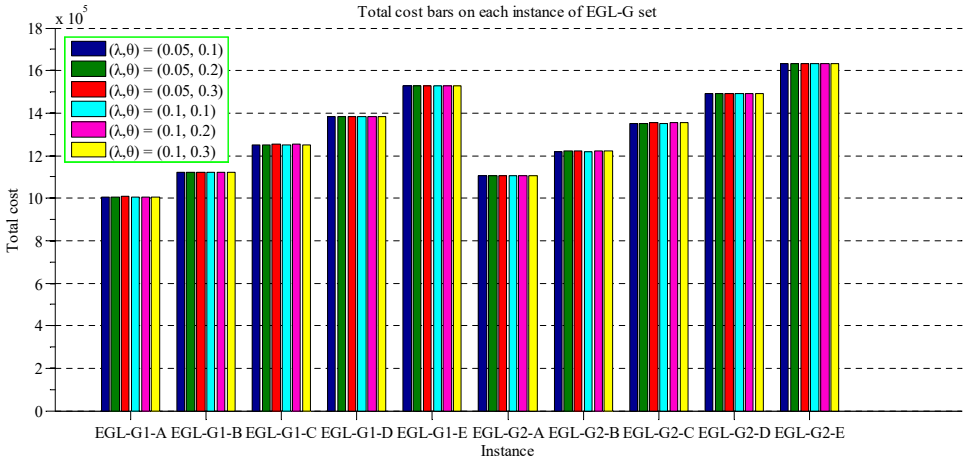

Figure 3: The average total cost over the 30 independent runs of RCO-RDG-MEANS on each EGL-G instance.

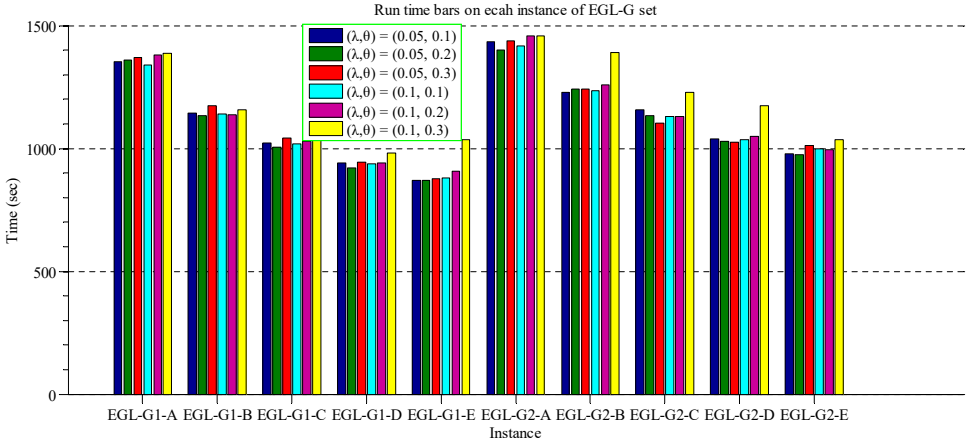

Figure 4: The average computational time over the 30 independent runs of RCO-RDG-MEANS on each EGL-G instance.

In summary, the tested and values have little impact on the performance, and affect the computational time only slightly. This is a good sign, as it means that the effectiveness and efficiency of the algorithm are not sensitive to the and values within a considerable range.

Fig. 4 shows that (;) = (0:05; 0:2) achieved the overall lowest computation time. Therefore, in the subsequent experiments, we select (;) = (0:05; 0:2) for the RCO operator in both RCO-RDG-MAENS and RCO-SAHiD.

4.4. Results on Experiment 1

In Experiment 1, RCO-RDG-MAENS is compared with RDG-MAENS [21], ILS-RVND [19], IRDG-MAENS [32], QICA-CARP [34], and ESMEANS [33]. For RDG-MAENS, we downloaded the code from online [1] and reran it for 30 times independently on each instance. However, for all the other compared algorithms, no code is available for rerunning the experiments. Therefore, we directly copied the results of these algorithms from their original literature. Note that we configured RCO-RDG-MAENS in exactly the same way as RDG-MAENS. Therefore, we can guarantee a fair comparison with RDG-MAENS and other algorithms (as they compared with RDG-MAENS under the same configuration).

Table 1 shows the average performance of the compared algorithms on the 10 EGL-G instances. In the table, the columns "$|V|$", "$|E|$" and "$|T|$" stand for the number of vertices, edges and tasks, respectively. $\lceil \cdot \rceil$ indicates the minimum number of vehicles required to serve all the routes, which can be computed as $= \lceil \frac{\sum_{t \in T} d(t)}{Q} \rceil$, where $d(t)$ is the demand of task t and Q refers to the capacity of the vehicles. In general, with the same problem size, a larger value implies a more complex problem instance. For each algorithm, the columns "Mean" and "Std" are the mean and standard deviation of the total costs obtained by 30 independent runs. Note that there is no "Std" column for ILS-RVND and IRDG-MAENS, since only the mean value was reported in their original literature.

For each instance, the minimal mean total cost among all the algorithms is marked with "$*$". In addition, we conduct statistical test between RCO-RDG-MAENS and each compared algorithm using Wilcoxon rank sum test under the significance level of 0.05. If an algorithm is significantly worse (better) than RCO-RDG-MAENS, then it is marked with underline (in bold). In addition, the last row "W-D-L" stands for the number of instances on which RCO-RDG-MAENS performed significantly better than ("W"), statistically comparable with ("D"), and significantly worse than ("L") the corresponding algorithm. For example, "1-9-0" under RDG-MAENS indicates that RCO-RDG-MAENS performed significantly better than RDG-MAENS on 1 instance, and statistically comparable with it on the remaining 9 instances.

From Table 1, RCO-RDG-MAENS performed statistically comparable with RDG-MAENS on 9 out of 10 EGL-G instances, and significantly outperformed it on G2-B. In comparison with the other algorithms, RCO-RDG-MAENS performed much better. It significantly outperformed ILS-RVND and QICA-CARP on all the instances, and IRDG-MAENS and ESMAENS on 7 and 8 instances, respectively. RCO-RDG-MAENS never performed significantly worse than any compared algorithm on any instance.

Table 2 shows the best total cost obtained from the 30 independent runs of the compared algorithms on each instance.[2] The minimal total cost is marked with "$*$" and result is marked with undeline (in bold) if it is larger (smaller) than that of RCO-RDG-MAENS. In the last two rows, the "Mean" row stands for the mean values of the best total costs obtained by each compared algorithm over all the instances, and the last row "G-E-S" stands for the number of instances on which results obtained by RCO-RDG-MAENS was greater than ("G"), equal to ("E'), and smaller than ("S") the corresponding algorithm. Note that the table does not include IRDG-MAENS [32], since the best performance of IRDG-MAENS was not reported in the original literature.

From Table 2, one can see that RCO-RDG-MAENS reached the minimal total cost on 6 out of 10 instances, which is much more than that of the other algorithms (i.e., 3 instances for RDG-MAENS, 0 for ILS-RVND, 1 for QICA-CARP and 2 for ESMAENS). In terms of the mean of the best total cost over the 10 instances, RCO-RDG-MAENS performed much better than the other compared algorithms (e.g. 1300592.9 versus 1301236.6 in comparison with RDG-MAENS).

[1] The C code of RDG-MAENS is available from `http://homepages.ecs.vuw.ac.nz/~yimei/codes/RDG-MAENS.zip`

[2] The full detail of the best solutions can be found from `https://meiyi1986.github.io/files/data/carp/results.zip`.

Table 1: The average performance over 30 independent runs of the compared algorithms on the EGL-G dataset. For each instance, the minimal mean total cost is marked with "y". Under Wilcoxon rank sum test with significance level of 0:05, an algorithm is marked with underline (in bold) if it is significantly worse (better) than RCO-RDG-MAENS.

Name	$\|V\|$	$\|E\|$	$\|T\|$		ILS-RVND	IRDG-MAENS	QICA-CARP		ESMAENS		RDG-MAENS		RCO-RDG-MAENS	
					Mean	Mean	Mean	Std	Mean	Std	Mean	Std	Mean	Std
G1-A	255	375	347	20	1010937.4	1007977.1	1008151.8	4441.1	1007807.0	4462.0	1007368.0	4311.2	1005870.4†	3827.0
G1-B	255	375	347	25	1137141.5	1125763.6	1125874.0	5802.8	1125649.7	5214.6	1123369.1	5528.0	1121529.2†	4437.7
G1-C	255	375	347	30	1266576.8	1255674.1	1252912.8	5242.1	1254856.3	6233.1	1251028.7	4268.1	1250070.5†	4048.2
G1-D	255	375	347	35	1406929.0	1388277.5	1387461.7	6012.5	1385882.0	4112.6	1384901.5	6131.4	1383354.8†	4390.6
G1-E	255	375	347	40	1554220.2	1528397.0	1529252.2	6101.5	1530893.7	7361.4	1527631.0	5641.2	1526502.9†	5895.1
G2-A	255	375	375	22	1118363.0	1108959.5	1109462.4	5923.1	1107939.3	3282.9	1106081.9†	5144.1	1106843.3	4586.6
G2-B	255	375	375	27	1233720.5	1223541.5	1222531.7	4843.3	1223247.4	5608.4	1223705.7	5802.2	1220453.5†	5086.5
G2-C	255	375	375	32	1374479.7	1353653.7	1355637.0	5344.8	1355667.3	5589.5	1353819.1	5169.6	1352801.7†	4289.5
G2-D	255	375	375	37	1515119.3	1495822.2	1492428.0	3696.0	1492155.9	6385.7	1492745.4	7146.8	1490704.2†	5973.5
G2-E	255	375	375	42	1658378.1	1636473.4	1636746.5	5764.3	1635161.3	5737.0	1633191.9	5704.8	1631377.8†	6041.6
W-D-L					10-0-0	7-3-0	10-0-0		8-2-0		1-9-0			

Table 2: The best total cost of the compared algorithms on EGL-G dataset. For each instance, the minimal total cost is marked with "y". An algorithm is marked with underline (in bold) if its total cost is greater (smaller) than RCO-RDG-MAENS.

Name	ILS-RVND	QICA-CARP	ESMAENS	RDG- MAENS	RCO-RDG-MAENS
G1-A	1002264	999151	998682	**998405**†	998763
G1-B	1126509	1118030†	1118092	1118030†	1118030†
G1-C	1260193	1245398	1246350	**1242897**†	1243096
G1-D	1397656	1376795	1377291	1375583	1375319†
G1-E	1541853	1518055	1516089	1518694	1513589†
G2-A	1111127	1100447	1100134	1097581	1097291†
G2-B	1223737	1213004	1212564	1211805	1211789†
G2-C	1366629	1344221	**1343044**†	1344228	1344353
G2-D	1506024	1482861	**1478162**†	1482216	1482345
G2-E	1650657	1625984	1622275	1622927	1621354†
Mean	1318665.0	1302394.6	1301268.3	1301236.6	1300592.9
G-E-S	10-0-0	9-1-0	8-0-2	7-1-2	

To make more intuitive comparisons between RCO-RDG-MAENS and RDG-MAENS, two representative instances (i.e., G1-A and G2-E) are selected from EGL-G, and the convergence curves of RCO-RDG-MAENS and RDG-MAENS on them are shown in Fig. 5. In the figure, the x-axis is the computational time, and the y-axis is the mean total cost of the best-so-far solutions obtained by the two algorithms. From the figure, one can see that the convergence curves of the two algorithms are very close to each other. RCO-RDG-MAENS tend to converge slightly slower than RDG-MAENS in the early stage of the search, and then catch up and achieve better results than RDG-MAENS in the later stage (e.g. the two curves crossed each other after around 500 seconds for both instances).

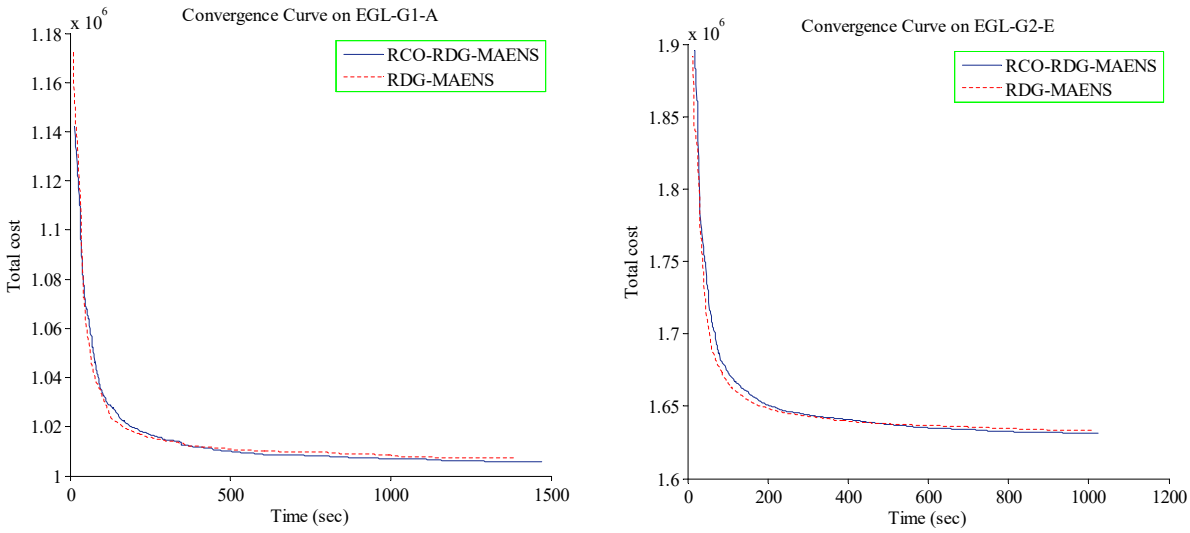

Figure 5: Convergence curves of RCO-RDG-MAENS and RDG-MAENS on instances EGL-G1-A and EGL-G2-E.

In summary, the comparison between RCO-RDG-MAENS and other algorithms including RDG-MAENS shows that the RCO operator can improve the performance of RDG-MAENS on the EGL-G instances. In terms of average performance, RCO-RDG-MAENS significantly outperformend RDG-MAENS on 1 instance, and was never beaten by RDG-MAENS. In terms of best performance, RCO-RDG-MAENS managed to outperform RDG-MAENS (and other compared algorithms) on most EGL-G instances.

The advantage of the RCO operator looks marginal in Experiment 1. This is partially because RDG-MAENS already has a high decomposition accuracy for medium-sized instances such as the EGL-G instances [38], and there is not much space for improvement by the RCO operator. However, the Hefei, Beijing and KW datasets are much larger and more complex. Thus, we expect the RCO operator to show more advantage in Experiments 2 and 3.

4.5. Results on Experiment 2

In Experiment 2, RCO-SAHiD is compared with SAHiD [38], UHGS [41], RDG-MAENS [21], VNS [30] and TSA1 [6] on the Hefei and Beijing datasets, which are much larger than the EGL-G dataset. Following the same practice in [38], both RCO-SAHiD and SAHiD were ran 25 times independently. The maximum runtime of the compared algorithms is set to 2492 seconds after the scaling.

Note that the configuration of Experiment 2 is very different from Experiment 1. The instances in Experiment 2 is much larger than the instances in Experiment 1, and the time budget is much tighter. Therefore, the search efficiency of the algorithm within a very limited time budget becomes much more important.

Table 3 shows the average performance of the compared algorithm on the 10 Hefei instances. For each instance, the minimal mean total cost is marked with "*y*". Under Wilcoxon rank sum test with significance level of 0:05, if an algorithm is significantly worse (better) than RCO-SAHiD, then its result is marked with underline (in bold).

From Table 3, one can see that RCO-SAHiD significantly outperforms all the other compared algorithms except UHGS with respect to the average performance. In particular, RCO-SAHiD significantly outperformed SAHiD on 9 out of the 10 instances, with much smaller mean and standard deviation. This indicates that embedding RCO into SAHiD can greatly improve its effectiveness and stability.

Table 4 shows the best total cost obtained by the compared algorithms over the 25 independent runs on the Hefei dataset. The table shows consistent patterns in terms of the best performance with the average performance. UHGS

Table 3: The average performance over 25 independent runs of the compared algorithms on the Hefei dataset. For each instance, the minimal mean total cost is marked with "y". Under Wilcoxon rank sum test with significance level of 0:05, an algorithm is marked with underline (in bold) if it is significantly worse (better) than RCO-SAHiD.

Name	$\|V\|$	$\|E\|$	$\|T\|$		RDG-MAENS		VNS		TSA1		UHGS		SAHiD		RCO-SAHiD	
					Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
Hefei-1	850	1212	121	7	247341	2293	247819	2745	252615	1591	**245596**†	0	251024	1820	247351	536
Hefei-2	850	1212	242	14	441539	4142	449979	5375	456228	5539	**433807**†	99	445376	2476	437631	1208
Hefei-3	850	1212	364	19	589152	2697	595263	3108	637201	8003	**573737**†	955	590969	2305	586795	1241
Hefei-4	850	1212	485	28	761351	4362	774323	6394	791790	5481	**740404**†	1577	759402	2495	753859	1898
Hefei-5	850	1212	606	35	991813	5755	994794	6109	1042701	11496	**946574**†	1741	976276	4742	967045	2766
Hefei-6	850	1212	727	42	1132063	8966	1128667	9404	1162641	13806	**1072864**†	3024	1106735	5318	1098915	3964
Hefei-7	850	1212	848	49	1361125	14356	1337353	6745	1353502	6235	**1272880**†	3920	1309474	4792	1305057	3798
Hefei-8	850	1212	970	56	1550509	13695	1517151	12477	1537169	6709	**1436048**†	4838	1483694	4857	1478098	4466
Hefei-9	850	1212	1091	63	1749079	18872	1694957	10164	1716256	9236	**1605554**†	5151	1659700	6103	1656147	4493
Hefei-10	850	1212	1212	69	1923264	31697	1852622	10183	1901167	12679	**1754889**†	4306	1808860	7836	1810301	6003
W-D-L					9-1-0		10-0-0		10-0-0		0-0-10		9-1-0			

Table 4: The best total cost of the compared algorithms on the Hefei dataset. For each instance, the minimal total cost is marked with "y". An algorithm is marked with underline (in bold) if its total cost is greater (smaller) than RCO-SAHiD.

Name	RDG-MAENS	VNS	TSA1	UHGS	SAHiD	RCO-SAHiD
Hefei-1	246221	**245596**†	250155	**245596**†	248048	246571
Hefei-2	**436020**	436637	447853	**433648**†	441574	436031
Hefei-3	583050	588682	623795	**572545**†	586880	582839
Hefei-4	754855	763256	774182	**737730**†	754015	750687
Hefei-5	980153	984121	1019224	**941278**†	964772	961376
Hefei-6	1119584	1110030	1134041	**1068035**†	1095530	1092667
Hefei-7	1329745	1322290	1339160	**1266931**†	1299430	1299360
Hefei-8	1526453	1492790	1521857	**1427531**†	1474390	1469819
Hefei-9	1705381	1675790	1696706	**1598203**†	1648840	1645841
Hefei-10	1837767	1834860	1873504	**1748829**†	**1793890**	1799158
Mean	1051922.9	1045405.2	1068047.7	1004032.6	1030736.9	1028434.9
G-E-S	9-0-1	9-0-1	10-0-0	0-0-10	9-0-1	

performed the best on all the 10 Hefei instances. It is followed by RCO-SAHiD, which is much better than all the other algorithms.

Tables 5 and 6 show the average and best performance of the compared algorithms on the Beijing dataset. From the tables, we can observe consistent patterns with those on the Hefei dataset. UHGS performed the best on all the Beijing instances in terms of both average and best performance. RCO-SAHiD was the second best algorithm, showing significantly better performance than all the other compared algorithms (including SAHiD) on all the Beijing instances.

Table 5: The average performance over 25 independent runs of the compared algorithms on the Beijing dataset. For each instance, the minimal mean total cost is marked with "y". Under Wilcoxon rank sum test with significance level of 0:05, an algorithm is marked with underline (in bold) if it is significantly worse (better) than RCO-SAHiD.

Name	$\|V\|$	$\|E\|$	$\|T\|$		RDG-MAENS		VNS		TSA1		UHGS		SAHiD		RCO-SAHiD	
					Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
Beijing-1	2820	3584	358	7	829406	12688	782415	4452	829132	6340	**760578**y	0	784727	5591	770199	3178
Beijing-2	2820	3584	717	11	1337954	18939	1192292	10196	1401363	25378	**1132987**y	1638	1183955	8431	1163978	6258
Beijing-3	2820	3584	1075	18	1847922	33258	1618484	11888	1709279	14801	**1542405**y	3801	1605846	9231	1577027	6798
Beijing-4	2820	3584	1434	23	2193399	34159	1953892	16746	2070885	14532	**1847355**y	5571	1936994	11694	1896581	8411
Beijing-5	2820	3584	1792	30	2639458	32481	2335915	23040	2440319	26726	**2210443**y	5638	2298630	16879	2255386	8316
Beijing-6	2820	3584	2151	36	3047295	41112	2743677	18024	2814735	22018	**2571748**y	6003	2707500	18433	2650420	9621
Beijing-7	2820	3584	2509	41	3388263	26081	3063813	25226	3186240	22426	**2871881**y	10590	3038157	15658	2952809	14474
Beijing-8	2820	3584	2868	47	3697025	44951	3366215	24686	3456037	22381	**3150688**y	7879	3313590	21925	3233296	15953
Beijing-9	2820	3584	3226	52	4061793	49504	3723830	45148	3943883	37089	**3485819**y	10731	3684250	32404	3575671	15372
Beijing-10	2820	3584	3584	58	4353966	51063	4040694	27384	4103532	15501	**3785520**y	11830	4004310	29488	3884308	16206
W-D-L					10-0-0		10-0-0		10-0-0		0-0-10		10-0-0			

Table 6: The best total cost of the compared algorithms on the Beijing dataset. For each instance, the minimal total cost is marked with "y". An algorithm is marked with underline (in bold) if its total cost is greater (smaller) than RCO-SAHiD.

Name	RDG-MAENS	VNS	TSA1	UHGS	SAHiD	RCO-SAHiD
Beijing-1	812647	774502	813907	**760578**y	775523	765538
Beijing-2	1303570	1168190	1353567	**1129810**y	1167480	1148259
Beijing-3	1777852	1591540	1678224	**1534878**y	1586180	1563874
Beijing-4	2126151	1920330	2053938	**1836866**y	1910880	1879617
Beijing-5	2581910	2293120	2396483	**2199275**y	2273080	2234352
Beijing-6	2968102	2705060	2774161	**2561113**y	2664510	2632250
Beijing-7	3331900	3015790	3147294	**2851602**y	3013590	2925015
Beijing-8	3584696	3323850	3415275	**3136727**y	3283530	3203032
Beijing-9	3934270	3653630	3890129	**3462953**y	3621490	3541842
Beijing-10	4206005	4002040	4066188	**3765614**y	3935540	3852428
Mean	2662710.3	2444805.2	2558916.6	2323941.6	2423180.3	2374620.7
G-E-S	10-0-0	10-0-0	10-0-0	0-0-10	10-0-0	

To further demonstrate the efficacy of embedding RCO, Fig. 6 shows the convergence curves of RCO-SAHiD and SAHiD over the 25 independent runs on the Hefei-1, Hefei-10, Beijing-1 and Beijing-10 instances, where the x-axis is the computational time in seconds, and the y-axis is the average total cost of the best-so-far solution. These four instances are selected as the representative instances of the corresponding datasets, and similar patterns are observed on other instances.

From the figure, one can see that the convergence curves of RCO-SAHiD are almost always below that of SAHiD, and there is a decent gap between the two convergence curves. The only exception is the Hefei-10 instance, for which

SAHiD converged slightly better than RCO-SAHiD. This is also consistent with Table 3, which shows that Hefei-10 is the only instance where there is no statistical difference between RCO-SAHiD and SAHiD.

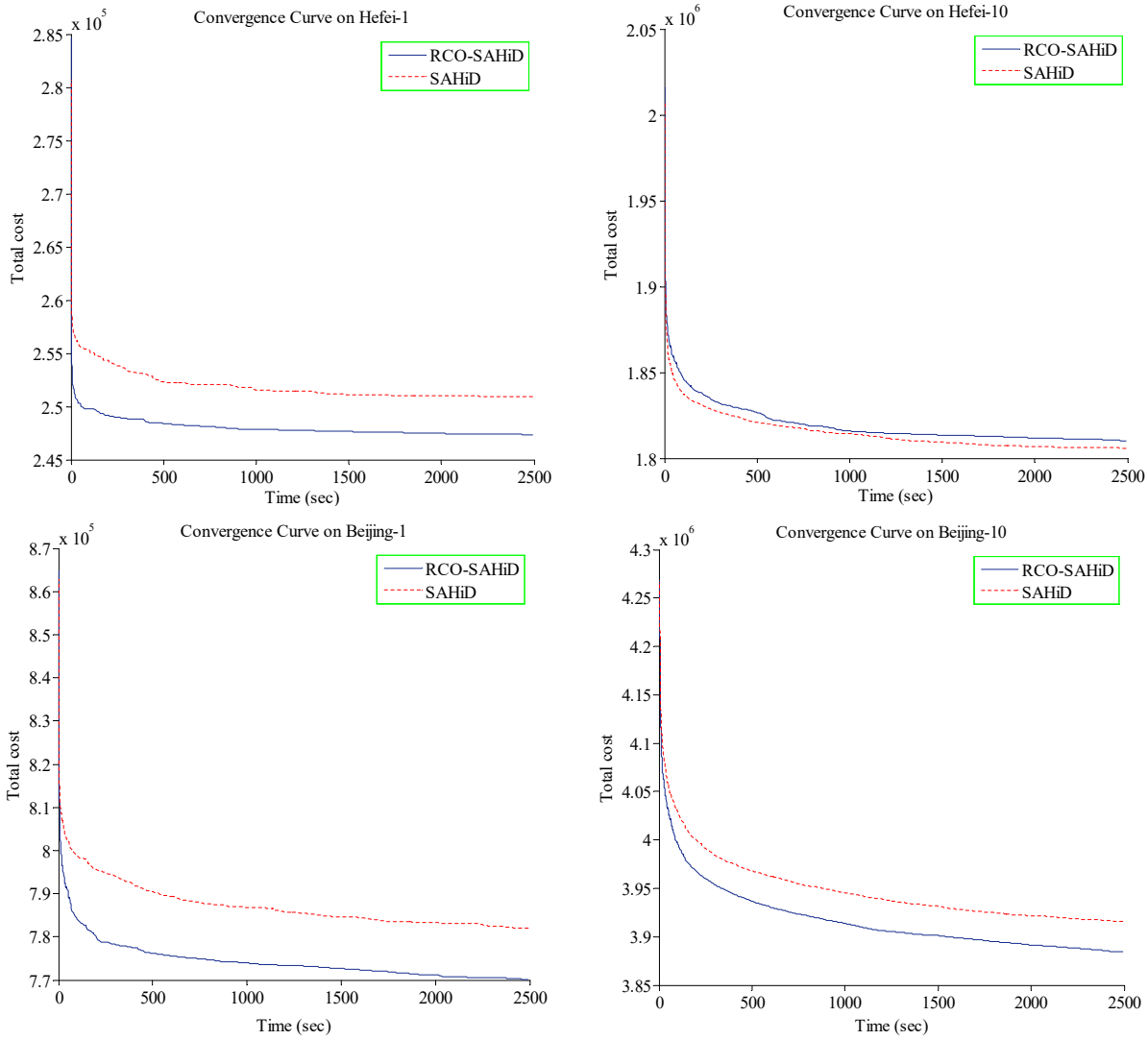

Figure 6: Convergence curves of RCO-SAHiD and SAHiD on the Hefei-1, Hefei-10, Beijing-1 and Beijing-10 instances.

In summary, the results in Experiment 2 clearly demonstrate the effectiveness of the RCO operator in improving the performance of SAHiD. Although RCO-SAHiD did not perform so well as UHGS, this is mainly due to the superiority of UHGS over SAHiD in terms of search capability, rather than the problem decomposition. Since UHGS is not a divide-and-conquer approach, we expect that embedding RCO into UHGS can further improve its performance.

4.6. Results on Experiment 3

Experiment 3 is to compare RCO-SAHiD with SAHiD [38], UHGS [41], Fast-CARP [43] and PS [47] on 12 largest KW instances. RCO-SAHiD, SAHiD and UHGS were ran 25 times independently. The runtime of 81 seconds per 1000 nodes is set for the compared algorithms on each instance. The results of Fast-CARP and PS are copied from the original literatures ([43] and [47]), as their codes are not available.

Compared with Experiment 2, Experiment 3 has a much larger problem size and much tighter time budget. For example, in K1_g-2, there are 8556 tasks, while only 81 11640=1000 = 943 seconds is allowed. In other words,

Experiment 3 has a strong requirement for an algorithm to search effectively in a huge search space within a very limited time budget.

Table 7 shows the average performance of the compared algorithms on the 12 KW instances. For each instance, the minimal mean total cost is marked with "y". Under Wilcoxon rank sum test with significance level of 0.05, if an algorithm is significantly worse (better) than RCO-SAHiD, then its result is marked with underline (in bold).

Table 7: The average performance over 25 independent runs of the compared algorithms on the KW dataset. For each instance, the minimal mean total cost is marked with "y". Under Wilcoxon rank sum test with significance level of 0.05, an algorithm is marked with underline (in bold) if it is significantly worse (better) than RCO-SAHiD.

Name	$\|V\|$	$\|E\|$	$\|T\|$	Q	UHGS		SAHiD		RCO-SAHiD	
					Mean	Std	Mean	Std	Mean	Std
K1_g-2	11640	12675	8566	48000	**6361639**y	62021	6555725	20127	6464425	28374
K1_g-6	11640	12675	8566	168000	**3593703**y	23857	3806616	16567	3785654	15569
K2_g-2	11636	12671	8563	48000	**6173748**y	69412	6361486	25149	6268937	27169
K2_g-4	11636	12671	8563	96000	**4353692**y	44890	4484168	26174	4463470	22877
K5_g-2	11405	12435	8267	48000	**5931093**y	70015	6143480	24601	6066358	25310
K5_g-6	11405	12435	8267	168000	**3447989**y	26460	3652255	18995	3637475	21134
O1_g-4	10283	11863	8581	96000	**3291833**y	28719	3428522	12159	3380808	11764
O1_g-6	10283	11863	8581	168000	**2700608**y	24040	2819667	13203	2800130	15190
O1_p-2	9957	11492	8220	48000	**2515957**y	23390	2622827	11350	2611434	11455
O1_p-4	9957	11492	8220	96000	**2210256**y	18760	2314768	12627	2300794	14922
O6_g-2	9563	11073	7831	48000	**3511678**y	35021	3657354	12754	3595682	13912
O6_g-6	9563	11073	7831	168000	**2269116**y	15096	2371623	13594	2361511	11524
W-D-L					0-0-12		12-0-0			

From Table 7, one can see that RCO-SAHiD statistically significantly outperformed SAHiD on all the 12 KW instances. It is consistent with the results on the Hefei and Beijing datasets, which demonstrates the effectiveness of the proposed RCO operator. Again, UHGS performed the best on the KW datasets. However, we observed that it is much slower than our algorithm. Specifically, even the initialisation stage can take much longer time than the given time budget (e.g. 3774.5 seconds for the O1_p-4 instance, while the given time budget is 804.2 seconds).

Table 8 shows the best performance of the compared algorithms on the KW dataset, where the minimal total cost of each instance is marked with "y", and is marked with underline (in bold) if it is greater (smaller) than RCO-SAHiD. For Fast-CARP and PS, the results are obtained from [43] directly. Since their results were obtained by a single run after sophisticated parameter tuning (i.e. Fast-CARP), or the best ones obtained by 105 runs with different evaluation criteria and degrees of randomization (i.e. PS), we treated them as the best performance in the comparison. The patterns shown in Table 8 are consistent with those in Table 7. In addition, Fast-CARP performed slightly better than RCO-SAHiD on some KW instances. However, it was very carefully tuned, by testing a large number of combinations of parameters (about 200 combinations). On the other hand, SAHiD is almost parameter-free, and is easier to use in practice.

Fig. 7 shows the convergence curves of RCO-SAHiD and SAHiD over 25 independent runs on four representative KW instances (K1_g-2, K1_g-6, O6_g-2 and O6_g-6), where the x-axis is the computational time in seconds, and the y-axis is the average total cost of the best-so-far solution. These four instances are selected as the representative

Table 8: The best total cost of the compared algorithms on the Beijing dataset. For each instance, the minimal total cost is marked with "*y*". An algorithm is marked with underline (in bold) if its total cost is greater (smaller) than RCO-SAHiD.

Name	PS	Fast-CARP	UHGS	SAHiD	RCO-SAHiD
K1_g-2	7549094	6501210	**6274277***y*	6509006	6405640
K1_g-6	4886345	**3739724**	**3541665***y*	3771667	3760040
K2_g-2	7434491	6249733	**6033790***y*	6319036	6224123
K2_g-4	5629112	4434203	**4293975***y*	4445174	4416990
K5_g-2	7136324	6031579	**5822825***y*	6081146	6008824
K5_g-6	4707990	**3578627**	**3406571***y*	3609300	3596594
O1_g-4	4173408	**3278666**	**3260611***y*	3404825	3364902
O1_g-6	3596504	**2724848**	**2664709***y*	2794956	2766860
O1_p-2	3465916	**2509047**	**2484393***y*	2600948	2585479
O1_p-4	3089185	**2194629**	**2173807***y*	2285241	2268133
O6_g-2	4345961	**3531246**	**3459355***y*	3639059	3574446
O6_g-6	3100213	**2276829**	**2239678***y*	2352559	2343073
Mean	4926211.9	3920861.8	3804638.0	3984409.8	3942925.3
G-E-S	12-0-0	4-0-8	0-0-12	12-0-0	

instances of the corresponding datasets, and similar patterns are observed on other instances.

From the figure, one can see that the convergence curves of RCO-SAHiD are always significantly below that of SAHiD, and there is an apparent gap between the two convergence curves. This indicates that no matter when to stop the search, RCO-SAHiD will always provide a better solution than SAHiD.

In summary, the results in Experiment 3 clearly demonstrates the effectiveness of the proposed RCO operator in improving the SAHiD algorithm on large instances. Although RCO-SAHiD did not manage to outperform the state-of-the-art UHGS and Fast-CARP, due to the limited performance of the embedded SAHiD, it was still faster and less space demanding than UHGS, and less parameter sentitive than Fast-CARP.

4.7. Scalability

To investigate how the effectiveness of the RCO operator is affected by the problem size, we plot the relative performance of RCO-SAHiD to SAHiD on different problem sizes. Fig. 8 shows the relative performance of RCO-SAHiD to SAHiD (ratio between their performances) versus the number of tasks on all the tested instances, where the *x*-axis is the number of tasks, and the *y*-axis is relative performance.

From Fig. 8, one can see that embedding RCO can improve the performance of SAHiD (the ratio is smaller than 1) on almost all the instances except one Hefei instance. For the Beijing instances, the advantage of RCO becomes more obvious as the problem size increases. For the Hefei instances, on the other hand, RCO becomes less effective with the increase of the problem size. For the EGLG and KW instances, there is no particular trend, since they have very similar problem sizes. Overall, RCO can almost always lead to improvement.

To give a further analysis on the poor performance of RCO on the Hefei dataset, we investigated the topology of the original network on Hefei-10 by computing the link rank of each pair of tasks. We observed that Hefei-10 contains less high-rank links than other instances. Such link distribution decreases the effectiveness of RCO in catching poor links. As a result, RCO-SAHiD can hardly obtain high-quality solution to Hefei-10, and showed poor performance.

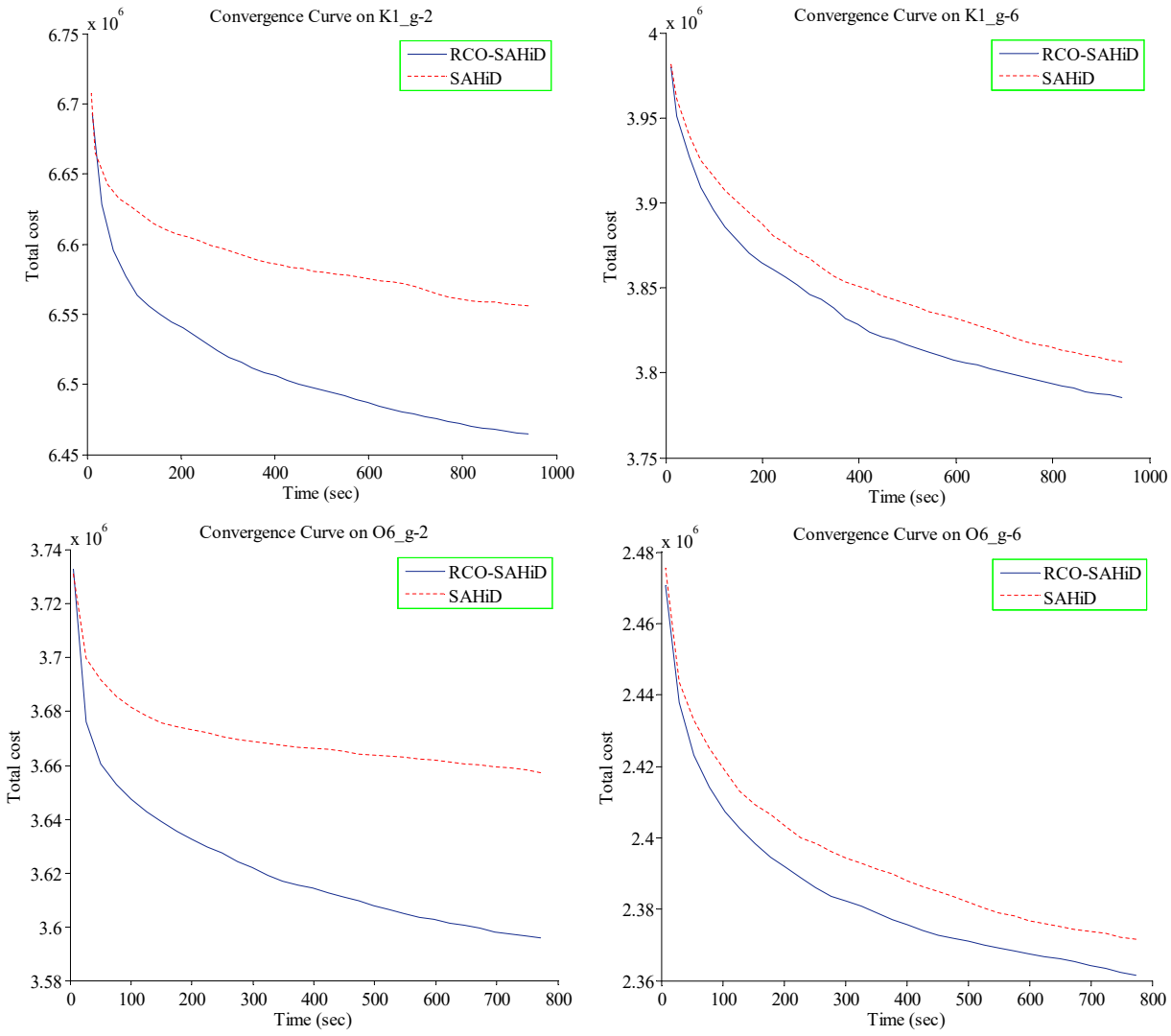

Figure 7: Convergence curves of RCO-SAHiD and SAHiD on the K1_g-2, K1_g-6, O6_g-2 and O6_g-6 instances.

5. Conclusions and Future Work

LSCARP is a hot topic of research on CARP, and a number of competitive algorithms have been proposed for solving it. In this paper, to better decompose the problem, a novel operator named the Route Cutting Off (RCO) operator is proposed for splitting the routes of the best-so-far solutions during the search process. The RCO operator is based on the idea that within the same route of the best-so-far solution, some links between tasks are good, and some others may be poor. We designed a task rank matrix, based on which the RCO operator classifies the links into good links and poor links. Then, it cuts off the two types of links with certain probabilities, in order to provide a better task subset for clustering, and thus lead to a better decomposition.

To verify the effectiveness of the proposed RCO operator, we propose two divide-and-conquer algorithms, namely RCO-RDG-MAENS and RCO-SAHiD, which are obtained by embedding the RCO operator into RDG-MAENS [21] and SAHiD [38], respectively. The experimental results clearly showed that the RCO operator managed to improve the performance of both RDG-MAENS and SAHiD, especially when the problem size is very large and the time budget is very limited. Note that the RCO operator is very flexible, and can be easily embedded into any divide-and-conquer approach that decomposes the problem by clustering the tasks.

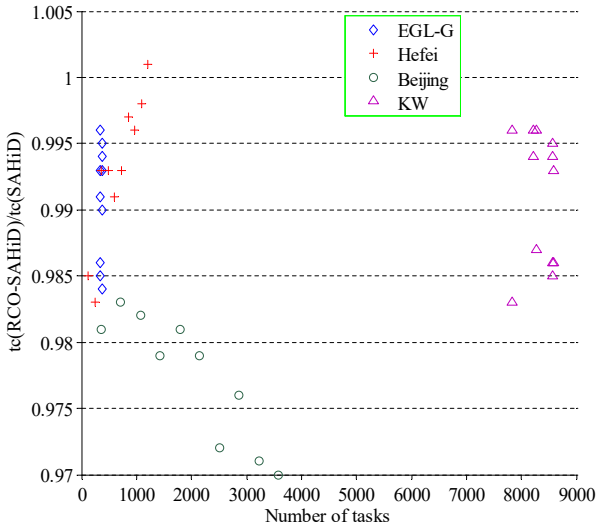

Figure 8: The relative performance of RCO-SAHiD to SAHiD versus the number of tasks on the tested datasets.

The possible future directions can be as follows. First, the current link classification can be improved. Currently, we simply use the average task rank of the solution as a threshold, and consider a link to be good if its rank is lower than the average task rank. The threshold is set in a rather arbitrary way, and may be improved in the future. Second, currently each route can have at most two links (one good link and one poor link) cut off during RCO. However, it may be more desirable to cut off more poor links than the good links, especially for long routes. In the future, we will consider adaptive schemes for deciding the number of cut-off links. Third, we will improve the robustness of the RCO operator in different graph topology.

Acknowledgment

This work was supported by Anhui Provincial Natural Science Foundation (No. 1808085MF173), Natural Science Key Research Project for Higher Education Institutions of Anhui Province(Nos. KJ2016A438, KJ2017A352), the National Key R & D Program of China under Grant 2017YFC1601800, and the National Natural Science Foundation of China (No. 61673194).

References

[1] R. Baldacci and V. Maniezzo. Exact methods based on node-routing formulations for undirected arc-routing problems. *Networks*, 47(1):52–60, 2010.

[2] E. Bartolini, J.-F. Cordeau, and G. Laporte. Improved lower bounds and exact algorithm for the capacitated arc routing problem. *Mathematical Programming Series B*, 137(1–2):409–452, 2013.

[3] J.M. Belenguer and E. Benavent. A cutting plane algorithm for the capacitated arc routing problem. *Computers and Operations Research*, 30(5):705–728, 2003.

[4] E. Benavent, V. Campos, A. Corberán, and E. Mota. The capacitated arc routing problem: lower bounds. *Networks*, 22(7):669–690, 1992.

[5] 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*, 147(3):629–643, 2003.

[6] J. Brandão and R. Eglese. A deterministic tabu search algorithm for the capacitated arc routing problem. *Computers and Operations Research*, 35(4):1112–1126, 2008.

[7] Y. Chen and J.K. Hao. Two phased hybrid local search for the periodic capacitated arc routing problem. *European Journal of Operational Research*, pages 1–24, 2018.

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

[9] E.W. Dijkstra. A note on two problems in connection with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[10] R.W. Eglese and L.Y.O. Li. A tabu search based heuristic for arc routing with a capacity constraint and time deadline. *Meta-Heuristics: Theory & Applications, Kluwer Academic Publishers, Boston*, pages 633–650, 1996.

[11] H.A. Eiselt and M. Gendreau. Arc routing problems, part II: the rural postman problem. *Operations Research*, 43(3):399–414, 1995.

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

[13] C.K. Goh and K.C. Tan. A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 13(1):103–127, 2009.

[14] B.L. Golden and R.T. Wong. Capacitated arc routing problems. *Networks*, 11(3):305–316, 1981.

[15] H. Handa, L. Chapman, and X. Yao. Robust route optimization for gritting/salting trucks: a CERCIA experience. *IEEE Computational Intelligence Magazine*, 1(1):6–9, 2006.

[16] L. Kiilerich and Wøhlk S. New large-scale data instances for CARP and new variations of CARP. *INFOR: Information Systems and Operational Research*, 56(1):1–32, 2018.

[17] P. Lacomme, C. Prins, and W. Ramdane-Cherif. Competitive memetic algorithms for arc routing problems. *Annals of Operations Research*, 131(1):159–185, 2004.

[18] H. Longo, M.P. de Aragão, and E. Uchoa. Solving capacitated arc routing problems using a transformation to the CVRP. *Computers and Operations Research*, 33(6):1823–1837, 2006.

[19] R. Martinelli, M. Poggi, and A. Subramanian. Improved bounds for large scale capacitated arc routing problem. *Computers & Operations Research*, 40(8):2145–2160, 2013.

[20] 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*, pages 1013–1020, 2013.

[21] Y. Mei, X. Li, and X. Yao. Cooperative coevolution with route distance grouping for large-scale capacitated arc routing problems. *IEEE Transactions on Evolutionary Computation*, 18(3):435–449, 2014.

[22] Y. Mei, X. Li, and X. Yao. Variable neighborhood decomposition for large scale capacitated arc routing problem. In *Proceedings of the 2014 IEEE Congress on Evolutionary Computation*, pages 1313–1320, 2014.

[23] Y. Mei, K. Tang, and X. Yao. A global repair operator for capacitated arc routing problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 39(3):723–734, 2009.

[24] Y. Mei, K. Tang, and X. Yao. A memetic algorithm for periodic capacitated arc routing problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 41(6):1654–1667, 2011.

[25] Y. Mei, K. Tang, and X. Yao. Decomposition-based memetic algorithm for multiobjective capacitated arc routing problem. *IEEE Transactions on Evolutionary Computation*, 15(2):151–165, 2011.

[26] M.C. Mourão and L. Amado. Heuristic method for a mixed capacitated arc routing problem: A refuse collection application. *European Journal of Operational Research*, 160(1):139–153, 2005.

[27] S. Nguyen, M.J. Zhang, M. Johnston, and K.C. Tan. Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming. *IEEE Transactions on Evolutionary Computation*, 18(2):193–208, 2014.

[28] F.B.D. Oliveira, R. Enayatifar, H.J. Sadaei, and J.Y. Potvin. A cooperative coevolutionary algorithm for the multi-depot vehicle routing problem. *Expert Systems with Applications*, 43(C):117–130, 2016.

[29] A. Ostertag, K.F. Doerner, R.F. Hartl, E.D. Taillard, and P. Waelti. Popmusic for a real-world large-scale vehicle routing problem with time windows. *Journal of the Operational Research Society*, 60(7):934–943, 2009.

[30] M. Polacek, K.F. Doerner, R.F. Hartl, and V. Maniezzo. A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. *Journal of Heuristics*, 14(5):405–423, 2008.

[31] C. Schlebusch and S. Irnich. Cut-first branch-and-price-second for the capacitated arc-routing problem. *Operations research*, 60(5):1167–1182 ., 2012.

[32] 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*, 46(4):1000–1013, 2016.

[33] 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*, pages 1–17, 2017.

[34] 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*, pages 1–22, 2017.

[35] R.H. Shang, H.N. Ma, J. Wang, L.C. Jiao, and R. Stolkin. Immune clonal selection algorithm for capacitated arc routing problem. *Soft Computing*, 20(6):2177–2204, 2016.

[36] K.C. Tan, Y.J. Yang, and C.K. Goh. A distributed cooperative coevolutionary algorithm for multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 10(5):527–549, 2006.

[37] K. Tang, Y. Mei, and X. Yao. Memetic algorithm with extended neighborhood search for capacitated arc routing problems. *IEEE Transactions on Evolutionary Computation*, 13(5):1151–1166, 2009.

[38] 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*, 47(11):3928–3940, 2017.

[39] M. Tang and X. Yao. A memetic algorithm for VLSI floorplanning. *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, 37(1):62–69, 2007.

[40] G. Ulusoy. The fleet size and mix problem for capacitated arc routing. *European Journal of Operational Research*, 22(3):329–337, 1985.

[41] T. Vidal. Node, edge, arc routing and turn penalties: Multiple problems-one neighborhood extension. *Operations Research*, 65(4):992–1010, 2017.

[42] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[43] S. Wøhlk and G. Laporte. A fast heuristic for large-scale capacitated arc routing problem. *Journal of the Operational Research Society*, 69(12):1877–1887, 2018.

[44] L.N. Xing, P. Rohlfshagen, Y.W. Chen, and X. Yao. An evolutionary approach to the multidepot capacitated arc routing problem. *IEEE Transactions on Evolutionary Computation*, 14(3):356–374, 2010.

[45] Z.Y. Yang, K. Tang, and X. Yao. Large scale evolutionary optimization using cooperative coevolution. *Information Sciences*, 178(15):2985–2999, 2008.

[46] T.T. Yao, X. Yao, S.S. Han, Y.C. Wang, D.P. Cao, and F.Y. Wang. Memetic algorithm with adaptive local search for capacitated arc routing problem. In *IEEE 20th International Conference on Intelligent Transportation Systems*, pages 1–6, 2017.

[47] H. Zbib. Variants of the path scanning construction heuristic for the no-split multi-compartment capacitated arc routing problem(Working Paper). *Aarhus University*, 2017.

[48] R. Zhang and C. Wu. A divide-and-conquer strategy with particle swarm optimization for the job shop scheduling problem. *Engineering Optimization*, 42(7):641–670, 2010.

[49] Y.Z. Zhang, Y. Mei, K. Tang, and K.Q. Jiang. Memetic algorithm with route decomposing for periodic capacitated arc routing problem. *Applied Soft Computing*, 52(3):1130–1142, 2017.