A Simple Heuristic Generating and Testing Algorithm to CARP Based on Hierarchical Decomposition

Zhao Zhixiang 11711621
dept. CSE of SUStech
11711621@mail.sustech.edu.cn

***Abstract*—The capacitated arc routing problem (CARP) is a classical challenging combinatorial optimization problem with wide practice applications. Many metaheuristic methods are applied to solve CARP. In this report, a similar algorithm with the method described in *A Scalable Approach to Capacitated Arc Routing Problems Based on Hierarchical Decomposition* (SAHiD) is proposed. The experimental results show that there is still a lot of room for improvement in the algorithm.**

I. Introduction

The capacitated arc routing problem (CARP) can be narrated in three different ways. First, as we all know, for the applications of this problem in real life such as winter gritting [1], urban waste collection [2], [3], and snow removal [4], [5], There is a depot where limited recycling trucks start to work from. And the capacitated trucks need to collect a certain amount of wastes in specific roads and return to depot. The second way to narrate the problem is to describe it as a more complex traveling salesman problem (TSP). There is a salesman team. Each salesman in the team can take limited items and travel around the cities to sell them. They need to finish a set of sale tasks distributed in different cities. At last, we can treat this problem as a stock transaction problem. A hedge fund needs to invest in several companies and inject a certain amount of money for each company with several pools of funds. The cost of investment is related to the buying order of the stocks of companies. The above characters want to minimize traveling costs or investment costs and simultaneously complete the tasks of the streets that need to clean, traveling budget, or the investment of companies.

In fact, the first narrative is the normal capacitated arc routing problem. The second is the capacitated vertex routing problem. And the third is extended knapsack problem [6]. But their natures are in common.

II. Problem Definition [7]

This work is focus on solving the undirected CARP whence a undirected graph $G(V, E)$ represents the crossroad or location (resp. V) and street (resp. E). Each edge $e \in E$ is associated with a cost $c(e) > 0$ and a demand $d(e) \geq 0$. $\forall$ edges e s.t. $d(e) \geq 0$, it induces a task set T, i.e., $T = \{\tau \in E | d(\tau) > 0\}$. A vertex $v_0 \in V$ is predefined as the depot. The aim of CARP is to determine a set of routes for the vehicles to serve all the tasks with minimal total costs withe the following constraint:

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 for tasks served in each route cannot exceed the vehicle capacity Q.

III. Background and Problem Representation

Typically, there are two ways to construct a feasible solution for CARP. Both ways are divided into two stages. One is cluster-first and route-second. On the contrary, the other is route-first and cluster-second. Therefore, both ways cause the differences between the representation of the solution if we using an Evolutionary Algorithm (EA). The first way leads to a Michigan Approach to represent the solution:

- CARP can be recognized as a combination of TSP.
- Individual is represented to be the route of a vehicle (only containing the required task), which is a subproblem of CARP.
- The whole population is the solution of CARP.

The searching operators for the individual with respect to this representation:

- Externally, the individual can send or receive the tasks within the capacity and satisfying other constraints. For example, the single insertion operator, swap operator [8] can be induced into this category.
- Internally, the individual can be treated as a small size of permutation and it needs to order the tasks in the route such that minimizes the cost. For example, the flip, 2-opt operator [8] .

While the other approach, Pittsburgh Approach is wildly used in many methods such as [8] [7], since [9] can always find the optimal split from a giant tour (permutation of all tasks). And the left question is how to efficiently search the giant tour with the cost as low as possible.

The Individual is represented to be the permutation of all tasks. By Ulusoy's splitting procedure [9], a solution can be obtained by splitting the permutation into a number of routes that satisfy capacity constraints. Hence the first stage of finding the permutation of tasks can be reduced into the TSP problem and the search operators for individuals are able to be transferred from the method of solving TSP.

Algorithm 1: HD(VT)

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

```
repeat
    randomly choose the cluster number K ∈ [1, β · |VT|];
    divide VT into groups by using k-means;
    order the virtual tasks within each group;
    VT ← {permutation of tasks in each group};
until |VT| = 1;
return the permutation of tasks in VT;
```

Fig. 1. HD [7]

IV. Proposed Algorithm

As the third section in [7] said, the key challenge to CARP can be viewed as finding the optimal permutation of tasks. The proposed HD scheme in [7] is an exact method to find a good permutation of tasks efficiently. The very specific detail of hierarchical decomposition is in [7]. And the pseudocode is shown in 1.

In my experience of implementing the k-means algorithm for grouping the virtual tasks, I found that sometimes the k-means algorithm won't converge whence I add a max number of iterations for assigning and updating to it.

And the original initialization of k-means is using a simple heuristic that disperses the centroids as widely as possible. i.e., 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.

However, In my algorithm, I randomly choose the centroids from all possible candidates.

The BIH implemented in my algorithm may be different from the original one. I followed the description in the [7] to achieve it step by step. Since I do have the source code, I do not know the differences between them.

In [7], the original method is HDU. After given the permutation of tasks by HD, it applies Ulusoy's splitting procedure to partition PT into a solution s'. But I fail to implement the Ulusoy's procedure. I use a naive one to replace the work of Ulusoy's. I call the new method to be HDN.

The first stage of local searching is treated as the reverse operator in TSP plus flips. That is my understanding. I am not sure if it is different from the original one.

It's worth noting that the merge-split operator is using Ulusoy's procedure as a part of it. I also use the naive splitting method to replace it. Hence the merge-split will be seriously affected by the replaced one.

Finally, I make some changes to SAHiD and come up with a little new algorithm named SAHiD*. It can be recognized as a heuristic generating and testing algorithm. The pseudo-code is shown as follows.

Algorithm 1 HDN(VT) (HD with Naive Splitting)

Input: virtual task set T
Output: a feasible solution s;

```
apply HD(VT) to generate a permutation of tasks PT;
clusters ← ∅
while there is no a allocated task do
    cluster ← ∅
    while cluster's load not exceed capacity do
        add the unallocated task with minimal index into cluster.
        if cluster's load exceeds capacity then
            add the cluster to clusters.
            break;
        end if
    end while
end while
return clusters
```

Algorithm 3: Local Search Procedure LS(s)

Procedure: **LS**(s)
Input: solution s
Output: potentially improved solution s

```
repeat
    for each sub-routes SR of each route R in s do
        reverse SR to obtain a new solution s';
        if s' is better than s then
            s ← s';
            break;
        end if
    end for
until s remains unchanged;
if s is not updated then
    apply MS operator to improve s;
    if s is updated then
        repeat
            for each sub-routes SR of each route R in s do
                reverse SR to obtain a new solution s';
                if s' is better than s then
                    s ← s';
                    break;
                end if
            end for
        until s remains unchanged;
    end if
end if
return s;
```

Fig. 2. LS [7]

V. Experimental Results and Discussion

A. results

The results are shown in table I. Note that the result of D-MEANS is picked up from all the non-dominated set with the lowest cost since the D-MEANS is the algorithm aim at multiobjective optimization.

B. Conclusion and Discussion

First I want to apologize. I made a mistake in my code. The cost is underestimated in the presentation on 5th June.

Problem	SAHiD*	D-MEANS	Problem	SAHiD*	D-MEANS	Problem	SAHiD*	D-MEANS
gdb1	386	316	val1A	656	324	egl-e1-A	5387	3548
gdb2	468	339	val1B	532	174	egl-e1-B	5824	4525
gdb3	353	275	val1C	531	245	egl-e1-C	6808	5683
gdb4	355	287	val2A	581	227	egl-e2-A	6962	5018
gdb5	497	377	val2B	631	260	egl-e2-B	8301	6389
gdb6	379	298	val2C	731	463	egl-e2-C	9422	8485
gdb7	369	325	val3A	233	81	egl-e3-A	8310	5982
gdb8	611	352	val3B	247	87	egl-e3-B	9824	7815
gdb9	576	309	val3C	266	138	egl-e3-C	11459	10380
gdb10	362	275	val4A	1119	400	egl-e4-A	9060	6531
gdb11	691	395	val4B	1140	412	egl-e4-B	11025	9117
gdb12	693	458	val4C	1172	444	egl-e4-C	12455	11877
gdb13	539	544	val4D	1281	544	egl-s1-A	6588	5207
gdb14	203	100	val5A	1132	423	egl-s1-B	7935	6478
gdb15	168	58	val5B	1162	446	egl-s1-C	8588	8582
gdb16	218	127	val5C	1188	474	egl-s2-A	14240	10298
gdb17	261	91	val5D	1328	593	egl-s2-B	15483	13448
gdb18	344	164	val6A	702	223	egl-s2-C	17933	17009
gdb19	123	123	val6B	724	235	egl-s3-A	14972	10533
gdb20	183	156	val6C	808	321	egl-s3-B	17396	14103
gdb21	279	200	val7A	897	279	egl-s3-C	18819	17095
gdb22	367	316	val7B	911	283	egl-s4-A	17421	12781
gdb23	424	235	val7C	948	338	egl-s4-B	19915	16934
			val8A	1036	386	egl-s4-C	21583	21426
			val8B	1067	395	egl-g1-D	2080973	1059662
			val8C	1154	533			
			val9A	1030	327			
			val9B	1057	329			
			val9C	1069	335			
			val9D	1122	395			
			val10A	1185	430			
			val10B	1215	439			
			val10C	1243	447			
			val10D	1343	539			

TABLE I
COST OF 1 RUN ON THE GDB, VAL, EGL SET

Algorithm 2 Pseudo code of SAHiD*

Input: task set T, max number of ilde iteration n
Output: a feasible solution s*

```
generate an initial solution s using HDN(T) with naive splitting;
Apply LS*(s) to improve s;
s* ← s;
while iterations ≤ n do
    generate a solution s' using HDN(T);
    apply LS(s') to improve s';
    if s' is better than s* then
        s* ← s;
        iteration ← 0;
    end if
    iteration++;
end while
return s*
```

And then I fixed it and get the correct costs.

We can see the costs of SAHiD* in table **??**tab:re are all higher than the cost of D-MEANS, no matter the size of the dataset. And the main reason is failing to implement the Ulusoy's splitting procedure in my algorithm.

I first try to implement [7] but fail to implement the [9] but meet a lot of bugs. So I can only find a simple way to replace the position of [9]. Since [9] always get the optimal feasible solution from the permutation of tasks, my naive splitting method can only find a feasible solution but not optimal. That will seriously affect the result not only because the algorithm fails to do the second stage, clustering, but also because the merge-split operator does not work well as we expect.

Second, the issue of local optimal is still wandering through this kind of heuristic algorithm. In my operating experience, the algorithm will stop at one thousand and more than a few iterations (almost less than one hundred and all less than 500), which means that the algorithm finds the best solution as possible as it can at the beginning and cannot find a better solution.

And there is an unclear point in [7]. When the algorithm comes into the while-loop, it said, "generate a virtual task set VT by splitting the routes of s ". How to split the routes of s? [7] did not tell the details.

VI. FUTURE WORK

There are some future works I want to achieve:

- First I need to correctly implement the Ulusoy's splitting and then I can talk about others.
- Since [7] is individual-based, it may be improved if we design a population-based EA algorithm.

- How to solve the issue of local optimal? I think we can design a long enough search operator whence it can reach all the possible solutions. Or we can constantly inject new "gene" to keep the diversity of individuals by using the random initialization of permutation.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] J. F. Campbell and A. Langevin, "Roadway snow and ice control," in Arc Routing. New York, NY, USA: Springer, 2000, pp. 389–418.

[6] A. Jaszkiewicz, "On the performance of multiple-objective genetic local search on the 0/1 knapsack problem: A comparative experiment," IEEE Trans. Evol. Comput., vol. 6, no. 4, pp. 402–412, Aug. 2002.

[7] K. Tang, J. Wang, X. Li and X. Yao, "A Scalable Approach to Capacitated Arc Routing Problems Based on Hierarchical Decomposition," in IEEE Transactions on Cybernetics, vol. 47, no. 11, pp. 3928-3940, Nov. 2017, doi: 10.1109/TCYB.2016.2590558.

[8] 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.

[9] G. Ulusoy, "The fleet size and mix problem for capacitated arc routing," Eur. J. Oper. Res., vol. 22, no. 3, pp. 329–337, 1985.