Capacitated Arc Routing Problem

November 29, 2021

Project2 Report of

CS303 Artificial Intelligence

11911839 NieYuhe

1 Preliminary

1.1 Problem Description

Arc Routing Problem[1] is the problem which we need to choose the best path based on different conditions of routes. Routes are usually described as nodes, arcs and edges with. Basically the goal of this problem is to satisfy the demand where some arcs or edges contain, and to minimize the cost of the whole path. **Capacitated Arc Routing Problem (CARP)** is one variant of Arc Routing Problem. The agent has limited capacity when handling the demand, which means it must go back to some depot to empty it's current load before it's load is over the capacity.

1.2 Problem Application

CARP is a familiar problem that often occurs in reality. It can be used in different kinds of work such as

- Urban waste collection task
- Taxi passenger carrying and fuel consumption
- Postal route planning
- Takeaway delivery service
- Sanding or salting the streets

2 Methodology

2.1 Notation

Symbol	Description
V	vertices set
E	edge set
A	arc set
a	one specific arc
T/A_R	requested arc set
D/v_0	depot vertex
Q	vehicle capacity
$S/individual$	CARP solution
$sc(S)$	the total cost of solution S
R_i	the i_{th} route of solution S
R_{ik}	the k_{th} task of k_{th} route of solution S
$load(R_i)$	the total demand of route R_i
$dem(R_{ik})$	the demand of the k_{th} task of k_{th} route
$sc(R)$	the total cost of route R
$sc(a)$	the cost of an arc a
$dem(a)$	the demand of an arc a
$head(a)$	the head vertex of an arc a
$tail(a)$	the tail vertex of an arc a
$inv(a)$	the inverse direction of an arc a
$app(a)$	the appearance of an arc a

Table 1: Notation List

2.2 Data Structure

There are four main data structures used in this project.

2.2.1 Arc

Structure *Arc*(Figure.1) is used for recording the detail of graph message, it represent a route start from a head vertex towards a tail vertex and it's information. It contains four elements:
Attribute:

- *head*: a int number that represent the head vertex
- *tail*: a int number that represent the tail vertex
- *sc*: serving cost, cost of a vehicle traveling along the edge/arc and serving it
- *dem*: demand for the corresponding arc task

In our CARP project, since the task are considered as edge task, thus each edge task has two arc representation.

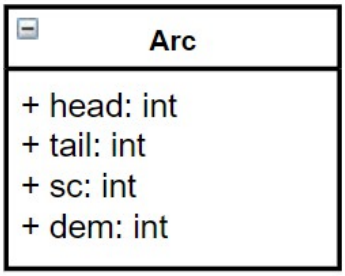

Figure 1: Arc data structure

2.2.2 Graph

Structure *Graph*(Figure.2) is used to represent the whole CARP route graph, it has several attributes and one method
Attribute:

- V: the number of vertices
- D: depot vertex index
- Er: the number of required edges
- NEr: the number of non- required edges
- Q: vehicle capacity
- C: total cost of all tasks
- *Adj*: a two dimensional int list that represent the adjacent matrix of each vertices
- *Short_Dist*: a two dimensional int list that record the minimum distance between each of two vertices
- *Arc*: a list of Arc object that record all the information of the graph that offered (because the information is about the edges, we split each edge information into two Arc object)
- *R_Arc*: a list of Arc object which has demand

Method:

- *Floyd()*: the algorithm that perform on the *Adj* to generate *Short_Dist* of the graph

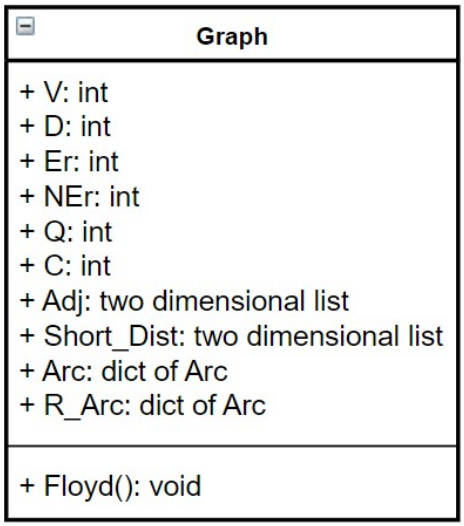

Figure 2: Graph data structure

2.2.3 PS

Structure *PS*(Figure.3) is used to generate the initial individual solution according to the Graph. It is the integration of the Path-Scaning algorithm, which contains several methods.
Method:

- *generate_individual()*: the algorithm that used to generate a individual
- *better()*: use five strategies to determine which arc task is better

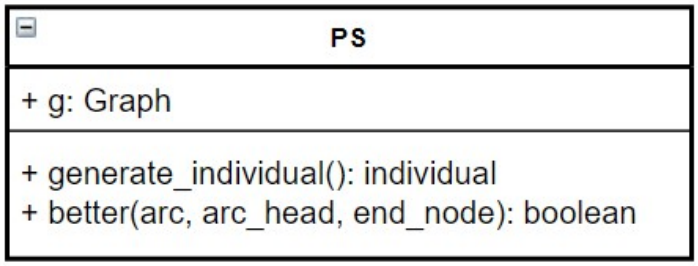

Figure 3: PS data structure

2.2.4 GA

Structure *GA*(Figure.4) is a genetic algorithm frame which is used to produce better solution. It contains common genetic algorithms and five mutation algorithms.

Method - Genetic Structure:

- *init_population(pop_size)*: generate population which contains pop_size individuals
- *fitness(individual)*: calculate the fitness value of one individual
- *select(n, population)*: select n individual who has the greatest fitness value
- *reproduce(population)*: use mutation strategy to generate a new mutated population
- *replacement(old_p, new_p)*: use some strategy to generate a new population according to old population and mutated population

Method - Mutation Strategy:

- *flip(individual)*: flip one gene of the individual
- *single_insertion(individual)*: pop one gene and insert it into one of the chromosome
- *double_insertion(individual)*: pop two genes and insert them into some chromosomes
- *swap(individual)*: swap two genes
- *single_opt(individual)*: flip a subset of one chromosome's gene

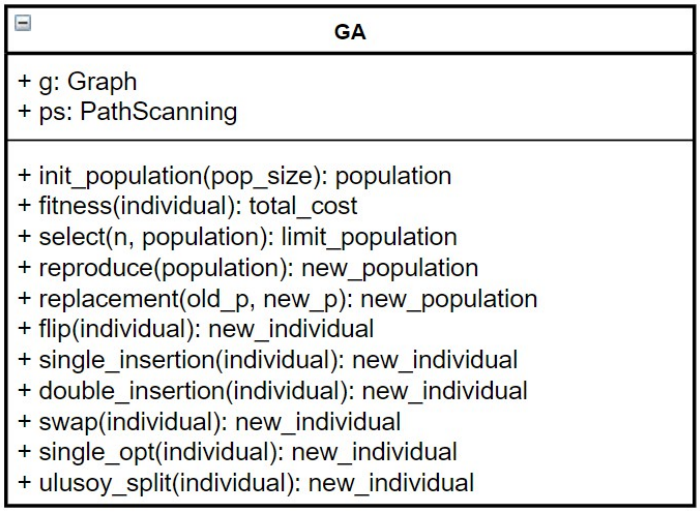

Figure 4: GA data structure

2.3 Model Design

2.3.1 Problem Formulation

CARP problem can be formulate in the following aspect:

Problem format:

- $G = (V, E)$: an undirected connected graph
- V: a vertex set
- A: an arc set, each arc $a \in A$ incurs a serve cost $sc(a)$ and a demand $dem(a)$, a head vertex $head(a)$ and a teal vertex $tail(a)$, an edge task e has two arc tasks, $e = \{a, inv(a)\}$
- T: a set of required arcs $T \subseteq A$, each required arc task $t \in T$ has a demand $dem(t) > 0$
- Q: a fleet of identical vehicles, each of capacity Q
- v_0: each vehicles starts and ends at a designated depot vertex $v_0 \in V$

Solution(aka individual) format S:

$$\begin{aligned} S &= (R_1, R_2, ..., R_m) \\ &= (0, \underbrace{R_{11}, R_{12}, ..., R_{1..}}_{R_1}, 0, \underbrace{R_{21}, R_{22}, ..., R_{2..}}_{R_2}, 0, \underbrace{R_{m1}, R_{m2}, ..., R_{m..}}_{R_m}, 0, 0) \end{aligned}$$

- m: the number of routes in S

- each R_i denote a single route, every R_i also consists of a sequence of arc task a served), the corresponding edge can be traversed more than once
- the load(the total demand of a route) is $load(R_i) = \sum_{k=1}^{length(R_i)} dem(R_{ik})$

CARP problem try to determine a set of routes for the vehicles to serve all the tasks with minimal cost while satisfying:

- each route must start and end at depot v_0
- total demand serviced on each route must not exceed Q
- each task must be served exactly once(an edge task contains two arc tasks, which one of them should be served), the corresponding edge can be traversed more than once

We can represent our target in minimizing $sc(S)$:

$$sc(S) = \sum_{i=1}^{length(S)-1} sc(R_i)$$

$$sc(R) = \sum_{i=1}^{len(R)-1} [sc(a_i) + sp(a_i, a_{i+1})]$$

$$\begin{aligned} \text{s.t. : } & \operatorname{app}(a_i) = 1, \forall a_i \in A_R \\ & \operatorname{app}(a_i) + \operatorname{app}(inv(a_i)) = 1, \forall a_i \in A_R \\ & \operatorname{load}(R_i) \leq Q \end{aligned}$$

2.3.2 Basic Flowchart

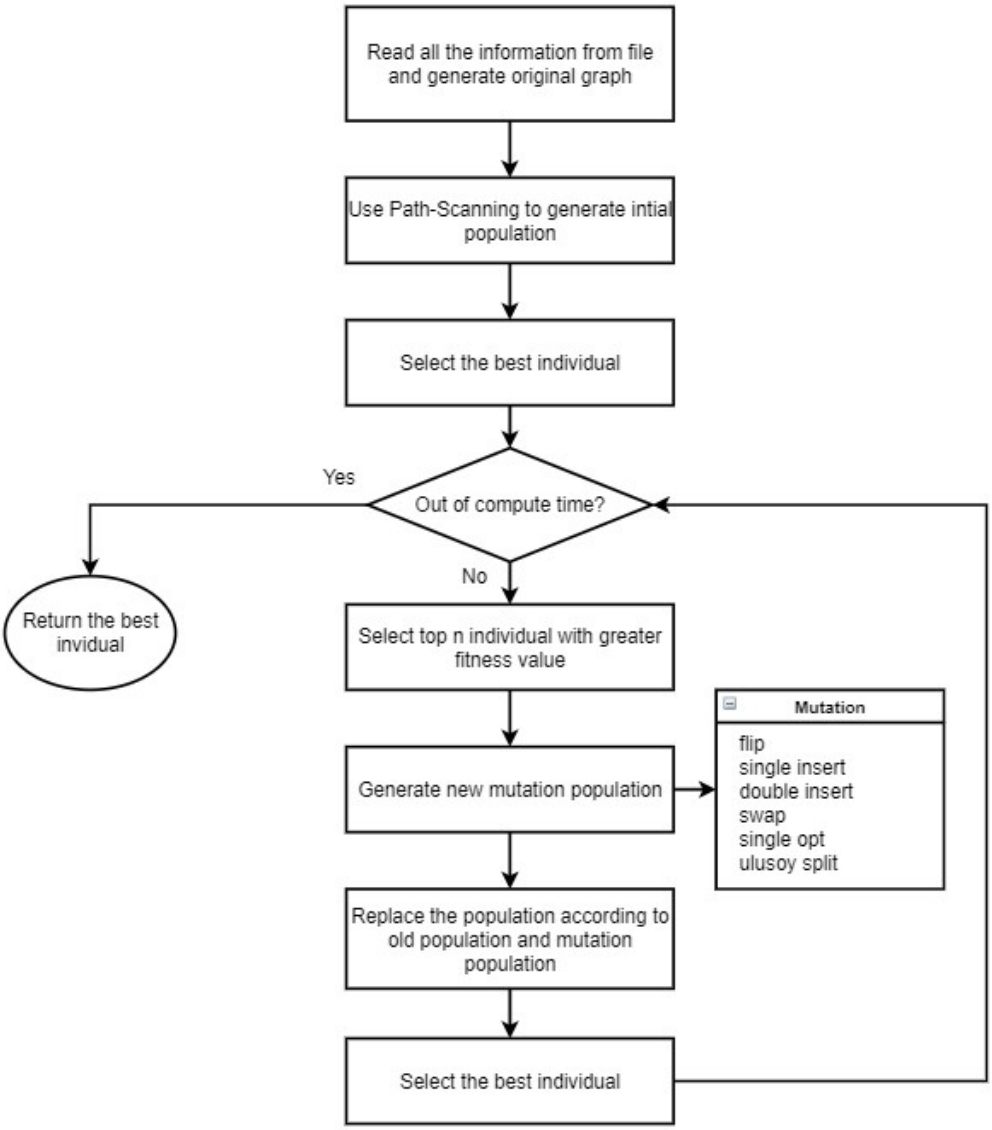

Figure 5: Framework of solving the CARP problem

2.4 Detail of Algorithm

2.4.1 Path Scanning

Algorithm 1 PATH SCANNING

Input: A_R (Requested Arc Test)
Output: individual

create empty individual list
while $A_R \neq \emptyset$ **do**
  $i \leftarrow i + 1; R_i \leftarrow \emptyset; load(i), cost(i) \leftarrow \emptyset; end \leftarrow depot;$
  loop
    $\overline{h} \leftarrow \infty;$
    for each $a \in A_R \mid load(i) + dem(a) \leq Q$ **do**
      if $Dist[end][head(a)] < \overline{d}$ **then**
        $\overline{d} \leftarrow Dist[end][head(a)];$
        $\overline{a} \leftarrow a;$
      else if $Dist[end][head(a)] = \overline{d}$ and BETTER$(a, \overline{a},$ END$)$ **then**
        $\overline{a} \leftarrow a;$
      end if
    end for
    add $\overline{a}$ at the end of route R_i;
    $A_R \setminus \{\overline{a}, inv(\overline{a})\};$
    $load(i) \leftarrow load(i) + dem(\overline{a});$
    $cost(i) \leftarrow cost(i) + sc(\overline{a}) + \overline{d};$
    $end \leftarrow tail(\overline{a});$
    until $A_R = \emptyset$ or $\overline{h} = \infty$
  end loop
  add route R_i to the end of individual;
end while
return individual;

Algorithm 2 BETTER

Input: a, $\overline{a}$, end
Output: True or False

Randomly choose the following five rules;
case1: **return** Dist[end][head($\overline{a}$)] + sc($\overline{a}$) < Dist[end][head(a)] + sc(a);
case2: **return** Dist[depot] + [tail($\overline{a}$)] > Dist[depot] + [tail(a)];
case3: **return** Dist[depot] + [tail($\overline{a}$)] < Dist[depot] + [tail(a)];
case4: **return** (dem($\overline{a}$) / sc($\overline{a}$)) < (dem(a) / sc(a));
case5: **return** (dem($\overline{a}$) / sc($\overline{a}$)) > (dem(a) / sc(a));

2.4.2 Mutation Strategy

There are six basic strategies when doing mutation to generate new individual.

Flip Strategy
Randomly pick one arc a and flip it's direction. That is, change a to inv(a).

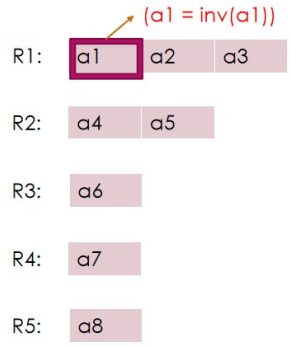

Figure 6: Flip Mutation

Single Insertion Strategy

Randomly pick one arc a and insert it into other position.

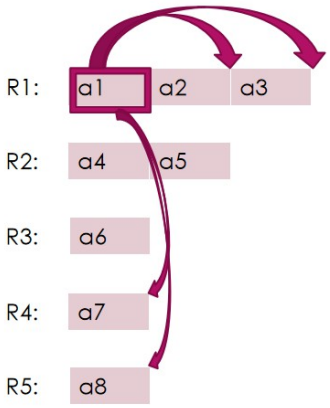

Figure 7: Single Insertion Mutation

Double Insertion Strategy

Randomly pick two arcs and insert them into other positions.

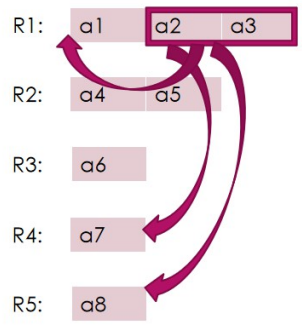

Figure 8: Double Insertion Mutation

Swap Strategy

Randomly pick two arcs and swap them.

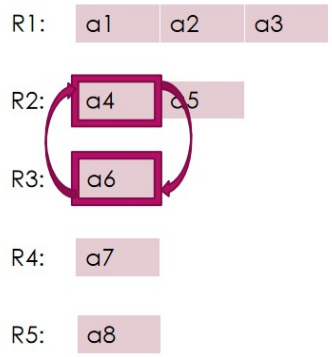

Figure 9: Swap Mutation

Single Opt Strategy

Choose choose a sub-route of a route, flip all the arcs in the sub-route.

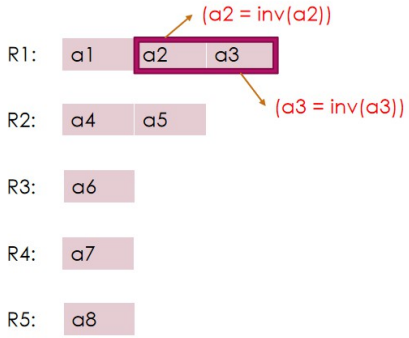

Figure 10: Single Opt Mutation

Ulusoy Split Strategy

Ulusoy Split[2] algorithm is first proposed by G.Ulusoy. Given a giant tour T which is a list of all required edges with chosen directions and connected implicitly by shortest path, Ulusoy Split partition the tour info some feasible vehicle routes.

Here is a compact implementation of ulusoy-split algorithm, which is proposed by Lacomme, Prins and Ramdane-Cherif.

Algorithm 3 UlusoySplit

Input: individual
Output: split_individual

$T \leftarrow$ CreateGrandTour(individual);
create list V and P with both size equal to $len(T)$;
set V_0 and P_0 to 0, and $V_1, V_2, ...V_n$ to $+\infty$;
for $i \leftarrow 1$ to $len(T)$ **do**;
    $load, cost \leftarrow 0$; $j \leftarrow i$;
    loop
        $load \leftarrow load + sc(T_j)$;
        if $j = i$ **then**
            $cost \leftarrow Dist[depot][tail(T_i)] + sc(T_i) + Dist[tail(T_i)][depot]$;
        else
            $cost - Dist[tail(T_{j-1})][depot] + Dist[tail(T_{j-1})][head(T_j)] + sc(T_j) + Dist[tail(T_j)][depot]$;
        endif
        if $load \leq Q$ **then**
            if $V_{i-1} + cost < V_j$ **then**
                $V_j \leftarrow V_{i-1} + cost$;
                $P_j \leftarrow i - 1$;
            endif
            $j \leftarrow j + 1$;
        endif
        until $(j > t) or (load > Q)$
    end loop
end for
split_individual $\leftarrow$ split by P;

2.4.3 Genetic Algorithm Framework

Algorithm 4 GeneticAlgorithm

Input: pop_size
Output: best

$Origin_{pop} \leftarrow$ InitialPopulation();
$Cur_{pop} \leftarrow$ top pop_size Fitness(individuals) in the $Original_{pop}$;
best $\leftarrow$ min(Fitness(Cur_{pop}));
while Not Terminate **do**
    $Mute_{pop} \leftarrow$ Reproduce(Cur_{pop});
    $Cur_{pop} \leftarrow$ Replacement($Mute_{pop}$, Cur_{pop});
    best $\leftarrow$ min(Fitness(Cur_{pop}));
end while
return best

3 Empirical Verification

3.1 Data Set

In addition to the data set CARP platform supplies. I have used some of the CARP benchmark sets which are provided by vidlat on github[3]. It mainly contains the following test cases:

Benchmark	Graph Size	Vertice Number
kshs1	tiny	5-10
gdb	small	10-15
val	small	20-30
egl-e	middle	50-100
egl-s	large	140-200

Table 2: Different types of Benchmark

3.2 Performance Measure

3.2.1 Test Environment

Personal Judge:

- Software: *Python* (Editor: *Pycharm Professional 2021.1.1*)
- Hardware: *Lenovo-Rescuer Inteli7-9760H CPU @ 2.60GHz(12CPUs), 2.6GHz*

CARP platform:

- Operation System: Debian 10
- Server CPU: 2.2GHz*2,8-core total
- Python version: 3.9.7

3.2.2 Measurement Criteria

CARP benchmark sets have their best solution criteria. In my personal judge, I limit the execution time to at most 10 minutes, and use the same random seed = 10 to test the solution result. The experimental results will be showed in section 3.4.

In addition to this, CARP online judge platform also provides me a measurement criteria of the efficiency of my code. It not only shows the result of my code, but also provides the rank compare with other students.

3.3 Hyperparameters

There are some hyperparameters which are used for the program.

Hyperparameter	Value
ORIGIN_SIZE	300-1000 according to graph size
POP_SIZE	30
BREAK_INTERVAL	3

Table 3: Hyperparameter

ORIGIN_SIZE is the size of generate initial population. Through path-scanning, we will obtain a good initial solution. So it is important to generate more initial solution and choose some with the best fitness value to form the first generation of population. Generate initial population with a large size in a large graph will cost many time, so we will dynamically change this hyperparameter according to the graph size.

POP_SIZE is the size of population when perform genetic algorithm to form each generation. When the size is small, it is hard to maintain variant genotype of individual. When the size is large, the reproduce and mutation will cost more time in generate new population. In practice, 30 is a optimal hyperparameter which consider both robustness and time cost.

BREAK_INTERVAL is the time interval to break out of the iteration when close to the time limit. Since the time cost of each iteration of new generation will not exceed 2 seconds, we choose 3 seconds in prohibiting time limit overflow.

3.4 Experimental Results

3.4.1 Personal Judge

With time limit = 10 minutes and the random seed = 10, the test results are as follows:

Name	$\|V\|$	$\|E\|$	TSA_{best}	Soution
1	8	15	14661	**14661**
2	10	15	9863	**9863**
3	6	15	9320	**9320**
4	8	15	11890	**11890**
5	8	15	10957	**10957**

Table 4: kshs Benchmark

Name	$\|V\|$	$\|E\|$	TSA_{best}	Soution
1	12	22	316	**316**
2	12	26	339	**339**
3	12	22	275	**275**
4	11	19	287	**287**
5	13	26	377	**377**

Table 5: gdb Benchmark

Name	$\|V\|$	$\|E\|$	TSA_{best}	Soution
1A	24	39	173	**173**
1B	24	39	173	**178**
1C	24	39	245	**258**
2A	24	34	227	**229**
2B	24	34	259	**268**

Table 6: val Benchmark

Name	$\|V\|$	$\|E\|$	TSA_{best}	Soution
1-A	77	51	3548	**3676**
1-B	77	51	4498	**4786**
1-C	77	51	5566	**6033**
2-A	77	72	5018	**5300**
2-B	77	72	6305	**6798**

Table 7: egl-E Benchmark

Name	$\|V\|$	$\|E\|$	TSA_{best}	Soution
1-A	140	75	5018	**5782**
1-B	140	75	6388	**7045**
1-C	140	75	8518	**9156**
2-A	140	147	9956	**11269**
2-B	140	147	13165	**14488**

Table 8: egl-A Benchmark

3.4.2 CARP Platform

The result which CARP Platform returns to me is:

gdb10	gdb1	egl-s1-A	egl-e1-A	val7A	val4A	val1A
275	316	5732	3844	291	416	173

Table 9: CARP Platform Result

The estimate rank for my solution in the platform is around 30, which is the top 20% of all students.

3.5 Conclusion Analysis

3.5.1 Algorithm Evaluation

My CARP program performs well when the graph size is small or middle. It can easily find the best or better solution. However, when the graph size become large, it is hard to find the optimal solution because it's stuck in a locally optimal solution.

Advantage:

- Path-Scanning generate good initial solutions
- Mutation strategy in the genetic process try to find better solution

Disadvantage:

- Robustness deficiency: when the graph become large, the performance decrease
- Difficult to get out of the local optimal solution, although using ulusoy-split and crossover strategy, it is still hard to jump out of the local optimal solution

3.5.2 Experience

This project offers me a opportunities of implement one classical NP-Hard Arc-Routing problem. It's problem formation and strategy in using genetic algorithm provides me a training opportunity. Also, many of the algorithms are inspired from thesis I searched and lecture I learnt. This is the first time I try to reproducing the idea of the paper. It makes me experienced the feeling of scientific research. In addition to this, the benchmark data sets are found online, by using those data, I realize the strictness that one research should offers. It really a valuable chance for me to learn so many things.

3.5.3 Possible Improvement

Given me more time, I would spending more time to study the paper and the detail heuristic strategies of CARP problem. And I would talk with teachers and classmates, trying to find more efficiency way to improve my code.

3.6 Acknowledgement

Thanks for teacher Y.Zhao, I asked her many questions about CARP, like ulusoy-split. It is her patience answering that helps me a lot.

Thanks for my boy friend Z.Zhan, I share my ideas with him, and he offers me many inspirations in revising my code.

Thanks for teacher K.Tang and X.Yao, I read their thesis[4] carefully and understand the definition of CARP and some solutions they proposed.

References

[1] Wikipedia contributors, "Arc routing — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=Arc_routing&oldid=950319399, 2020, [Online; accessed 25-November-2021].

[2] Ulusoy, Gündüz, "The fleet size and mix problem for capacitated arc routing." *European journal of operational research*, vol. 22, pp. 329–337, 1985.

[3] Vidalt, "Hgs-carp," https://github.com/vidalt/HGS-CARP/tree/master/Instances, 2020.

[4] Ke Tang, Yi Mei, Xin Yao, "Memetic algorithm with extended neighborhood search for capacitated arc routing problems," *IEEE Transactions On Evolutionary Computation*, vol. 13, pp. 1151–1165, 2009.