Solving Capacitated Arc Routing Problems with Evolutionary Simulated Annealing

Zubin Zheng, *Sophomore, SUSTech*

***Abstract*—In this project, we use ESA to solve CARP. ESA can perfectly solve small to medium-sized instances since its use of simulated annealing and evolutionary processes can avoid falling into local optima. However, ESA can not fully solve the large-scale difficult CARP. The experimental results are basically consistent with the analysis in the Methodology, perfectly hitting the optimal values of five instances in OJ.**

***Index Terms*—Capacitated arc routing problem (CARP), combinatorial optimization, evolutionary simulated annealing, local search.**

I. Introduction

THE ARC routing problem is originated in the Swiss mathematician Leon-hard Eulerin's study about Königsberg bridges problem in the 18th century [1]. It is a combinatorial optimization problem that requires determining the optimal routing plan for the fleet subject to some constraints [2]. The research on routing problems can be divided into two categories: vehicle routing problem (VRP) with points as service objects and arc routing problem (ARP) with arcs as service objects. Unlike the former, the basic feature of ARP is that the fleet starts from a designated depot and operates on all edges that require service, rather than serving at the vertex.

The capacitated arc routing problem (CARP), which is the most typical form of the arc routing problem, is considered in this report. It can be informally described as follows: Given an undirected graph consisting of a set of vertices and edges, each edge has a predefined traversal cost, some edges are associated with a service cost and demand, which must be serviced by vehicles. A fleet of identical vehicles is based at the designated depot vertex and has a limited capacity. The objective of CARP is to find a minimum cost routing plan for vehicles such that:

- 1) each required edge is serviced on one and only one of the routes;
- 2) each route must start and end at the designated depot vertex;
- 3) the total demand serviced on the route of a vehicle must not exceed the vehicle capacity [3].

CARP is a NP-hard problem [4]. The exact algorithms are expected to obtain the optimal solution but the consumed time is unreasonable since it can not solve all instances when the scale of instances is too large in a polynomial time. For these reasons, non-exact algorithms including heuristic and meta-heuristic methods show more advantages over exact algorithms. Representative heuristic methods include Augment-Merge, Path-Scanning and Ulusoy's route-first cluster-second method. Among the meta-heuristic methods, neighborhood search approaches and population-based algorithms are popular to solve CARP, and the latter typically achieve superior performance, such as memetic algorithms include MAENS [5] as the state-of-the-art method for the classical test instance sets [3].

Since Golden and Wong proposed the Capacity Arc Routing Problem in 1981 [1], CARP has been widely applied in daily life, especially in municipal services, such as road sprinkler path planning, garbage collection vehicle path planning, road deicing vehicle path planning, school bus transportation path planning, and so on.

The purpose of this report is to show how to solve CARP with Evolutionary Simulated Annealing (ESA). Inspired by the frame of evolutionary simulated annealing described in [6] for solving UFLP, we transfer and apply the ideas of algorithms to solve CARP by modifying the corresponding parameters and replace the movements to neighborhoods by five distinct common move operators. The experimental results indicate that ESA can solve CARP to a certain extent.

The rest of this report is organized as follows. Section II introduces preliminary including the formal problem definition of CARP, explanation of terminology and notation used throughout this report. Section III as methodology describes the general workflow, detailed algorithm ESA design and analysis. Section IV presents the experimental studies, which include the pre-experiment and main experiment in the seven open test instances in OJ and robustness experiment in egl set. Finally, conclusions will be presented in Section V.

II. Preliminary

Given a graph $G(V,E)$ with a set of vertices V, a set of edges E, a set of required edges $TASK \subset E$ and a vehicle with a capacity of Q that is based at the depot vertex $v_d \in V$, each edge $e \in E$ is represented by the order pair with its beginning vertex i and ending vertex j as $e = (i,j)$. Each edge e is characterized by four attributes: the beginning vertex u, the ending vertex v, the cost c and the demand d. If the demand d equals to 0, then the egde is not a required edge. A required edge is claimed to be served if and only if the corresponding edge is included in one of the routing plan. We

This project was supported by Department of Computer Science and Engineering, Southern University of Science and Technology of China(SUSTech). This project was posted in CS311 Artificial Inteligence(H) as project 2 on April 7, 2023, and was updated on April 30, 2023. This project was finished on May 6, 2023, before deadline May 12, 2023.

Z. Zheng is a sophomore in Southern University of Science and Technology of China(SUSTech), Shenzhen, Guangdong Province, China. (e-mail: 12112328@mail.sustech.edu.cn).

use the terminology $task$ to represent a required edge. Let n be the number of We have a solution to CARP as:

$$s = (R_1, R_2, ..., R_m)$$

where m is the number of routes. The k^{th} route $R_k = (0, task_{k1}, task_{k2}, ..., task_{kl_k}, 0)$, where $task_{kt}$ and l_k denote the t^{th} task and the number of tasks served in R_k, and 0 denotes a dummy task which is used to separate different routes. The cost and demand of the dummy task are both 0 and its two endpoints are both v_d(the depot). Moreover, since each task is an undirected edge and it can be served from either direction, so each task in R_k must be specified from which direction it will be served. Specifically, $task_{kt}$ = (u, v).

Hence, the problem formulation of CARP is described as follows:

$$\textbf{minimize } TC(s) = \sum_{k=1}^{m} RC(R_k) \tag{1}$$

subject to

$$\sum_{k=1}^{m} l_k = |TASK| = n \tag{2}$$

$$task_{k_1 i_1} \neq task_{k_2 i_2} \tag{3}$$

$$(task_{k_1 i_1}.u, task_{k_1 i_1}.v) \neq (task_{k_2 i_2}.v, task_{k_2 i_2}.u) \tag{4}$$

where $\forall(k_1, i_1 \neq k_2, i_2)$ in constraint (3) and (4)

$$\sum_{i=1}^{l_k} task_{ki}.d \leqslant Q, \forall k = 1, 2, ..., m \tag{5}$$

$$task_{ki} \in TASK \tag{6}$$

where $RC(R_k)$ is the route cost of route R_k, which can be computed as:

$$RC(R_k) = \sum_{i=1}^{l_k} task_{ki}.c + dis(v_d, task_{k1}.u) +$$

$$\sum_{i=1}^{l_k - 1} dis(task_{ki}.v, task_{k(i+1)}.u) + dis(task_{kl_k}.v, v_d)$$

where $dis(v_i, v_j) \geqslant 0$ is the cost of the shortest path from v_i to v_j.

The objective function of CARP (1) is the total cost of all routes; Constraint (2) (3) (4) ensure that each task is served exactly once; Constraint (5) is the capacity constraint; Constraint (6) is the definition of the tasks.

III. Methodology

A. General Workflow

The proposed method ESA can be divides into 2 phases: initialization phase and evolution phase. In initialization phase, we need to preprocess the data from input to construct the graph, compute the cost of the shortest path from v_i to v_j by floyd algorithm. After that, we should construct initial population by path-scanning algorithm. In evolution phase, we will run simulated annealing(SA) algorithm with several iteration until the time is used up. In SA, we invoke five common move operators (CMO) including flip, single insertion, double insertion, swap and 2-opt to make the selected feasible solution move to its neighborhood).

Based on the description above, the flowchart of solving CARP by ESA is shown as the following figure:

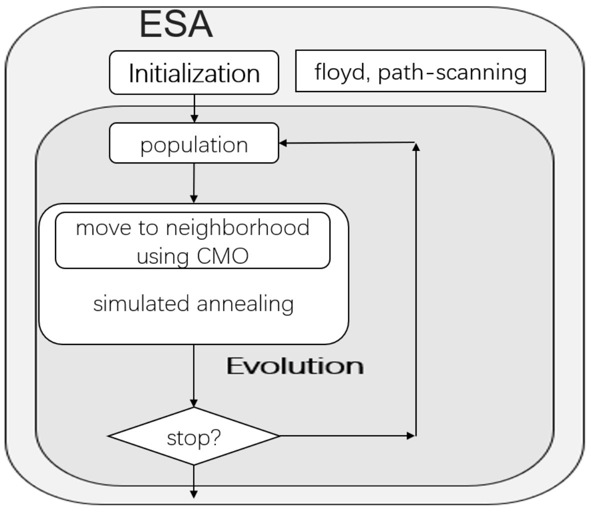

Fig. 1. ESA

B. Detailed Algorithm Design

1) In initialization phase, we apply two main algorithms: floyd and path-scanning. Since the implementation of floyd algorithm is trivial, here we only show the details of path-scanning. Firstly, we copy all required arcs in a list free. Secondly, we repeat the following steps to generate paths one by one:

- start at the depot
- repeat to add the path which is the closest to the end of current path, not yet serviced and compatible with vehicle capacity . If multiple tasks are the closest to the end of current path, five rules are used to determine the next task.
- no task can join the path and go back to the depot

where the five rules mentioned above can be described as follows:

- 1) maximize the distance from the task to the depot;
- 2) minimize the distance from the task to the depot;
- 3) maximize the term task.d / task.c;
- 4) minimize the term task.d / task.c;
- 5) use rule 1) if the current load is less than Q/2, otherwise use rule 2).

Based on the description above, the pseudo-code of path-scanning is shown as the following:

Algorithm 1 Path-Scanning with Five Rules

```
Input: $rule\_id \in \{1,2,3,4,5\}$, $MIN[][]$
    //MIN[][] is the the shortest path from $v_i$ to $v_j$ calculated by floyd algorithm
1:  $free\_list[] \leftarrow$ copy all required arcs
2:  $route \leftarrow []$, $load \leftarrow []$, $cost \leftarrow []$, $cnt \leftarrow 0$
3:  while $free\_list[] \neq \varnothing$ do
4:      $cnt \leftarrow cnt + 1$
5:      $cur \leftarrow v_d$
6:      $route.append([])$, $load.append(0)$, $cost.append(0)$
7:      while $free\_list[] \neq \varnothing$ do
8:          $next \leftarrow e.u$, randomly choose a required arc $e$ from $free\_list[]$
9:          $dis \leftarrow \infty$
10:         for a required arc $y$ in $free\_list[]$ do
11:             if $load[cnt] + y.demand \leqslant Q$ then
12:                 $d \leftarrow MIN[cur][y.v]$
13:                 if $d < dis$ then
14:                     $dis \leftarrow d$, $next \leftarrow y$
15:                 else if d == dis then
16:                     use $rule\_id$'s rule to decide if update $next$
17:                 end if
18:             end if
19:         end for
20:         if dis is unchanged then
21:             break
22:         end if
23:         $route[cnt].append($(next.u, next.v))
24:         $load[cnt] \leftarrow load[cnt] + next.demand$
25:         $cost[cnt] \leftarrow cost[cnt]$ + MIN[cur][next.u] + $next.cost$
26:         $cur \leftarrow next.v$
27:         $free\_list[]$ removes (next.u, next.v) and (next.v, next.u)
28:     end while
29:     $cost[cnt] \leftarrow cost[cnt]$ + MIN[cur][$v_d$]
30: end while
Output: $solution \leftarrow$ Solution(route, load, cost, sum(cost))
```

After running Path-Scanning with five rules, we obtain 5 different individuals as the initial solutions in the population.

2) In evolution phase, we run SA on the 5-individual population in each iteration. Unlike other evolutionary algorithms, ESA uses SA in local search instead of mutation operator or crossover operator and uses the evolutionary approach in global search. In SA operator, $X_n{}'$ is yielded by moving from a feasible solution X_n (X_n consists of 4 attributes: route, load, cost and sum of cost) in the direction of a neighbourhood function we call five common move operators (CMO) as follows:

$$X_n{}' = CMO(X_n) = \begin{cases} flip\left(X_n\right), 0 \leqslant \rho < i, \\ single\_insertion\left(X_n\right), i \leqslant \rho < j, \\ double\_insertion\left(X_n\right), j \leqslant \rho < k, \\ 2-opt\left(X_n\right), k \leqslant \rho < l, \\ swap\left(X_n\right), l \leqslant \rho < 1, \end{cases} \tag{7}$$

where ρ is a uniformly generated random number, $i < j < k < l$ and $i,j,k,l \in (0,1)$. $flip(X)$ tries to flip through all tasks to see if there are any improvements, $single\_insertion(X)$ removes an individual task from current position and re-inserts it after other tasks or depot in the same route or another route [5], $double\_insertion(X)$ removes two consecutive tasks from their current position and re-inserts into another position of the current solution or an another route [5], $2-opt(X)$ has two types, one for a single route which is that a sub-route is selected and its direction is reversed and the other for double routes which is that each route is first cut into two sub-routes, and new solutions are generated by reconnecting the four sub-routes [5], and $swap(X)$ simply swaps two tasks in two distinct routes.

Based on X_n and its neighbourhood $X_n{}'$, the new qualified solution X_{n+1} is determined in SA operator as follows:

$$X_{n+1} = \begin{cases} X_n', (\varDelta f\left(X\right) < 0) \cup \left(\rho < e^{-\frac{\varDelta f(X)}{T_n}}\right), \\ X_n, \ otherwise, \end{cases} \tag{8}$$

where $\varDelta f\left(X\right) = f\left(X_n{}'\right) - f\left(X_n\right)$, f is the objective value function, i.e sum of cost in solution X. T_n is the level of temperature at the nth iteration which is cooled through a particular cooling function. The equation means the new qualified solution X_{n+1} can be assigned as $X_n{}'$ if $X_n{}'$ is better or $X_n{}'$ is not better but X_{n+1} will accept it at a probability = $e^{-\frac{\varDelta f(X)}{T_n}}$ to avoid being trapped in local optimal solution.

Based on the description above, pseudo-code of solving CARP by ESA is shown as the following:

Algorithm 2 Evolutionary Simulated Annealing

```
Input: All information about CARP
1: Initialization
2: while remaining time $\geqslant$ time required to run SA once do
3:     SA()
4: end while
Output: Best $solution$ obtained by ESA
```

where SA is is shown as the following:

Algorithm 3 Simulated Annealing(SA)

```
1:  t ← Ts // set initial temperature
2:  while t > 0.1 do
3:      t ← t × cr, curIter ← 0 // cr is cooling rate
4:      while curIter < iter do
5:          curIter ← curIter + 1, $X_n \leftarrow$ a random individual
6:          $\rho \leftarrow$ a random number ∈ [0, 1], $X_n' \leftarrow$ CMO($X_n$)
7:          if f($X_n'$) < f($X_n$) then
8:              $X_{n+1} \leftarrow X_n'$
9:          else
10:             $\rho \leftarrow$ a random number ∈ [0, 1]
11:             if $\rho < e^{-\frac{\varDelta f(X)}{T_n}}$ then
12:                 $X_{n+1} \leftarrow X_n'$
13:             end if
14:         end if
15:     end while
16: end while
```

C. Analysis

The optimality of ESA: ESA uses multiple simulated annealing and use 5-individual population in evolution which is different from one single SA. SA can accept a bad solution at a probability related to current temperature; in next evolution the temperature to restore initial high temperature. These two stratagies contributes to optimality of ESA that is not easily trapped in local optima.

The complexity of ESA: Due to the randomness of evolutionary algorithms and the complexity of search operators, the analysis process of computational time complexity is often cumbersome and difficult [7]. Therefore, we do not show the complexity of ESA here.

The deciding factor of its performance: population size, initial temperature Ts, cooling rate cr, inner iteration $iter$, interval division $i, j, k, l \in (0, 1)$.

IV. Experiments

A. Experimental Setup

Two datasets are used in this project. The first we used is a mixed dataset of the 7 CARP instances in OJ, which is the main dataset for efficacy test in the Experiment. There are 2 instances from the egl set, 2 instances from the gdb set and 3 instances from the val set. After that, the robustness of ESA will be tested on egl set from $github.com/zh-plus/AI-courseproject/tree/master/CARP/CARP\_samples$. The egl set was generated by Eglese based on data from a winter gritting application in placeLancashire, which consists of 24 instances based on two graphs, each with a distinct set of required edges and capacity constraints [5]. Table I shows the characteristics(the number of vertices, edges and tasks) in the mixed dataset of the 7 CARP instances:

TABLE I
The characteristics of the 7 CARP instances

instance	egl-e1-A	egl-s1-A	gdb1	gdb10	val1A	val4A	val7A
vertices	77	140	12	12	24	41	40
edges	98	190	22	25	39	69	66
tasks	51	75	22	25	39	69	66

By observation, we can divide the 7 instances into 3 class according to the number of vertices, edges and tasks: gdb1 and gbd10 are small-scale instances, val1A, val4A and val7A are middle-scale instances, egl-e1-A and egl-s1-A are large-scale instances. Furthermore, egl-e1-A and egl-s1-A will be the hardest two to solve since their scale is large and not all edges should be served.

ESA are performed on both PC and OJ. We set termination time = 60 s in PC. PC has 11th Gen Intel (R) Core (TM) i5-1135G7 @2.40GHz 2.42 GHz, 16.0 GB. The operating system is Microsoft Windows 10. The implementation of the algorithm adopts Python 3.10.10 and NumPy 1.23.4, and the compilation environment is Visual Studio Code. For the environment in OJ, the server CPU is 2.2GHz*2, 8-core total, the operation System is Debian 10 and Python version is 3.9.7.

B. Results

In order to figure out the effect of hyperparameter and different optional components in CMO, we design the following **pre-experiment** to determine the optimal parameters for the main experiment in Table II:

TABLE II
Parameters setting in pre-experiment

parameter	index	setting
POPSIZE	1	5
	2	10
	3	20
Ts	4	1
	5	50
	6	100
	7	150
cr	8	0.995
	9	0.999
iter	10	5
	11	20
	12	40
(i, j, k, l)	13	(0.2 0.4, 0.6, 0.8)
	14	(0.3, 0.35, 0.4, 0.8)

where index is a symbol for easy marking the results. When one parameter changes, others keep the same. All 7 instances in the mixed dataset of CARP are independently calculated once in PC(60s) and we only focus on costs of solutions. The result of pre-experiment in PC(60s) is shown in Table III:

TABLE III
RESULTS ON THE egl, gdb, val BENCHMARK TEST SET IN TERMS OF COSTS OF SOLUTIONS in pre-experiment

index	egl-e1-A	egl-s1-A	gdb1	gdb10	val1A	val4A	val7A
1	3602	**5416**	**316**	**275**	**173**	**400**	**279**
2	**3561**	5591	**316**	**275**	**173**	404	**279**
3	3792	5632	**316**	**275**	**173**	402	292
4	3777	5578	**316**	**275**	**173**	404	283
5	3645	**5329**	**316**	**275**	**173**	**400**	283
6	**3602**	5416	**316**	**275**	**173**	**400**	**279**
7	3719	5607	**316**	**275**	**173**	409	283
8	3768	5854	**316**	**275**	**173**	402	283
9	**3602**	**5416**	**316**	**275**	**173**	**400**	**279**
10	3894	5965	**316**	**275**	**173**	402	280
11	3702	**5388**	**316**	**275**	**173**	404	287
12	**3602**	5416	**316**	**275**	**173**	**400**	**279**
13	3739	5520	**316**	**275**	**173**	408	285
14	**3602**	**5416**	**316**	**275**	**173**	**400**	**279**

After that, we obtain the optimal parameters setting for ESA. In the **main experiment**, we set parameters as population size $POPSIZE$ = 5, initial temperature Ts = 100, cooling rate cr = 0.999, inner iteration $iter$ = 40, interval division $i = 0.3,\ j = 0.35,\ k = 0.4,\ l = 0.8$.

All 7 instances in the mixed dataset of CARP are independently calculated 3 times in PC(60s) and judged once in OJ. Multiple quantitative assessment measures, $Best$, $Mean$, Std and GAP are used to evaluate the obtained results. Where $Best$ is the optimal value obtained by ESA. $Mean$ and Std are the average and standard deviation, respectively. GAP is the gap between the mean of best values obtained by the algorithm and the optimal value. Here we consider the best value from the experiment in [5] as the the optimal value for each instance. The formula of GAP is given by the following:

$$GAP = \frac{Mean - OPT}{OPT} \tag{9}$$

The result of main experiment in PC(60s) and OJ compared with the results in MAENS [5] is shown in Table IV:

TABLE IV
RESULTS ON THE EGL, GDB, VAL BENCHMARK TEST SET IN TERMS OF COSTS OF SOLUTIONS IN MAIN EXPERIMENT

instance	Mean	Std	Best(PC)	BEST(OJ)	Best(MAENS)	GAP
egl-e1-A	3706.66	74.12	3602	3602	**3548**	0.04
egl-s1-A	5403.33	53.82	5332	5668	**5018**	0.07
gdb1	316.00	0.00	**316**	**316**	**316**	0.00
gdb10	275.00	0.00	**275**	**275**	**275**	0.00
val1A	173.00	0.00	**173**	**173**	**173**	0.00
val4A	400.66	0.94	**400**	**400**	**400**	0.001
val7A	284.33	4.10	**279**	**279**	**279**	0.02

Finally, the robustness of ESA is tested on egl set, since we set the same parameters with main experiment, do not adjust for this dataset. All instances in egl set are independently calculated once in PC(60s) and we focus on costs of solutions and the GAP between ESA and MAENS. The result of **robustness experiment** in PC(60s) is shown in Table V:

TABLE V
RESULTS ON THE EGL BENCHMARK TEST SET IN TERMS OF COSTS OF SOLUTIONS IN ROBUSTNESS EXPERIMENT

instance	vertices	edges	tasks	Best(ESA)	Best(MAENS)	GAP
egl-e1-A	77	98	51	3602	**3548**	0.015
egl-e1-B	77	98	51	4574	**4498**	0.016
egl-e1-C	77	98	51	5869	**5595**	0.048
egl-e2-A	77	98	72	5413	**5018**	0.078
egl-e2-B	77	98	72	6625	**6317**	0.048
egl-e2-C	77	98	72	8793	**8335**	0.054
egl-e3-A	77	98	87	6416	**5898**	0.087
egl-e3-B	77	98	87	8497	**7775**	0.092
egl-e3-C	77	98	87	10801	**10292**	0.049
egl-e4-A	77	98	98	7009	**6456**	0.085
egl-e4-B	77	98	98	9638	**8998**	0.071
egl-e4-C	77	98	98	12494	**11561**	0.080
egl-s1-A	140	190	75	5332	**5018**	0.062
egl-s1-B	140	190	75	6648	**6388**	0.040
egl-s1-C	140	190	75	9107	**8518**	0.069
egl-s2-A	140	190	147	11824	**9895**	0.194
egl-s2-B	140	190	147	15198	**13147**	0.156
egl-s2-C	140	190	147	18769	**16430**	0.142
egl-s3-A	140	190	159	12306	**10257**	0.199
egl-s3-B	140	190	159	15913	**13749**	0.157
egl-s3-C	140	190	159	19869	**17207**	0.154
egl-s4-A	140	190	190	15501	**12341**	0.256
egl-s4-B	140	190	190	19318	**16337**	0.182
egl-s4-C	140	190	190	24535	**20538**	0.194
Average	-	-	-	11002.125	**9756.8**	0.105

C. Analysis

Pre-experiment indicated that population size $POPSIZE$, initial temperature Ts, cooling rate cr, inner iteration $iter$, interval division i, j, k, l can affect the perfermance of ESA as we analyze in the Methodology part. In gdb1, gdb10, val1A, ESA hits the optimal value regardless of the parameter settings, perhaps these datasets are small in scale and easy to solve. The main challenge is in egl-e1-A, egl-s1-A, val4A and val7A. The population size $POPSIZE$ = 5 or 10 is suitable, too large population size will reduce algorithm performance. The initial temperature Ts = 50 or 100 suitable, not too low or too large. cooling rate cr = 0.999 contributes egl-e1-A, egl-s1-A to solve, inner iteration $iter$ should not too small. For the interval division (i, j, k, l), uniform distribution (0.2 0.4, 0.6, 0.8) is not better than (0.3, 0.35, 0.4, 0.8), it may indicate that flip, 2-opt play a greater role than single insertion and double insertion in the algorithm.

Main experiment indicated that ESA is good enough to sovle the small-scale and middle-scale CARP since it can hit the optimal value in the last five datasets. Although ESA can not hit the optimal value in egl-e1-A and egl-s1-A these two large-scale dataset, compared with MAENS which is the state-of-the-art algorithm to solve CARP, the GAP is only 0.04 and 0.07, which is very close to the optimal value.

Robustness experiment showed that ESA has good robustness. The minimal GAP is 0.015, the maximal GAP is 0.256, and the average GAP is only 0.105 which is slightly bigger than the GAP = 0.07 in main experiment.

V. CONCLUSION

ESA can perfectly solve small to medium-sized instances, and its use of simulated annealing and evolutionary processes can avoid falling into local optima. However, ESA cannot fully solve the large-scale difficult CARP, and there are many parameters, making it difficult to adjust parameters. The experimental results are basically consistent with the analysis in the Methodology, and the expected results have been achieved, perfectly hitting the optimal values of the five instances. We learned that how to read parameters from the terminal, how to read files, the trick that use of class to collect all information, the metric in the performance of an algorithm in this project. In the future, we will focus on two parts. The first is to search for more useful movement operators and embed them into ESA to improve its performance. After that, we will modify ESA to make it have great performance in solving large-scale CARP.

REFERENCES

[1] Corberán, Ángel, and Gilbert Laporte, eds. *Arc routing: problems, methods, and applications*. Society for Industrial and Applied Mathematics, 2015

[2] M. Dror, *Arc Routing, Theory, Solutions, and Applications*. Boston, MA: Kluwer, 2000.

[3] Chen, Yuning, Jin-Kao Hao, and Fred Glover. *A hybrid metaheuristic approach for the capacitated arc routing problem.* European Journal of Operational Research 253.1 (2016): 25-39.

[4] Golden B. L. and Wong R. T. *Capacitated arc routing problems*. Networks 1981; 11(3): 305-315.

[5] Ke,Tang, Yi Mei, and Xin Yao. *Memetic algorithm with extended neighborhood search for capacitated arc routing problems.* IEEE Transactions on Evolutionary Computation 13.5 (2009): 1151-1166.

[6] Yigit, Vecihi, M. Emin Aydin, and Orhan Turkbey. *Solving large-scale uncapacitated facility location problems with evolutionary simulated annealing.* International journal of production research 44.22 (2006): 4773-4791.

[7] Oliveto, Pietro S., Jun He, and Xin Yao. "Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results." International Journal of Automation and Computing 4 (2007): 281-293.