Capacitated Arc Routing Problems Report

Using part of MAENS to solve the problem

Kai WANG 11612909
School of Computer Science and Engineering
Southern University of Science and Technology
Email: 11612909@mail.sustc.edu.cn

***Abstract*—This is a report for the second project in AI class, the capacitated arc routing problems (CARP). The problem CARP is a popular question which has been researched by lots of people. The challenge is how to find the optimal solution in the shortest possible time. For solving this problem, I researched the method MAENS put forward by Dr. TANG. I learned a lot from this method and implemented part of it.**

***Index Terms*—MAENS, pathscanning, crossover, insert**

I. PRELIMINARY

A. Problem Description

The capacitated arc routing problem (CARP) is one of the most typical form of thr arc routing problem which can be described as follows: a graph G = (V, E), with a set of vertices denoted by V, a set of edges denoted by E.Each edges has a deadheading cost (the cost of a vehicle traveling along the edge/arc without serving it). And edges in subset A of se E have demand which need to be served with serving cost. By the way, edge and its reverse edge have the same attributes and only need to be served once. A solution to the problem is a routing plan that consists of a number of routes for the vehicles, and the objective is to minimize the total cost of the routing plan subject to the following constraints : [1]

1) each route starts and ends at the depot;
2) each task is served in exactly one route;
3) the total demand of each route must not exceed the vehicle's capacity Q.

B. Problem Applications

- Garbage collection;
- Route planning of sprinklers;
- China postman problem;

II. METHODOLOGY

A. Notation

- **DEPOT**: position of the depot;
- **Required**: a list of all edges need to be served;
- **Q**: the capacity of a vehicle;
- **MAX**: 99999, a constraint factor used in programing;
- **TIME**: maximum time that can be used;
- **SEED**: random seed given;
- **bestpop**: a list of $bestsize$ best solutions found by memetic algorithm ;
- **psize**: size of the population ;
- **opsize**: size of the offspring generated by each population ;
- **utrail**: the maximum time to find a new solution;
- **bestsize**: size of the shared best solutions set ;

Identify applicable funding agency here. If none, delete this.

B. Data Structure

- **table**: a three-dimensional array which stores all edges cost and arcs demand;
- **dijk**: a two-dimensional array which stores the shortest path between any two vertices;
- **bestpop**: a list of $bestsize$ best solutions found by memetic algorithm ;
  Table and **dijk** are transfered to each process. **bestpop** is shared by all processes.

C. Model design

Simply, the CARP is a path planning question with constraints. Therefore, I need to find some feasible solutions at first. And then to improve it as better as possible.

In the first part, the fondamental method is *pathscanning*. As we all known, *pathscanning* is a method using greedy algorithm which means we always choose the nealest task to serve. When there are two or more tasks have the same distance from current position, I using the *better* function to choose one of them. By the way, to avoid falling into local optimum, I apply some random choices in *pathscanning*. As result, it performs well.

In the second part, to improve the solutions, I used the *memetic algorithm (MA)* introdused in MAENS. In general, *MA* can converge to high-quality solutions more efficiently than their conventional evolutionary counterparts. MA is an outstanding algorithm which can help the solution jump out of the local optimum. Offsprings also can inherit good genes from their fathers. It makes it easier for solutions to evolve in a better direction.

For getting a better solution in a limitted time, multiprocessing is used in my program. After reading the file and getting the shortest route cost of any two vertices using *Dijisktra* algorithm, start multiprocessing. In every process, firstly using *pathscanning* to find $psize$ different solutions for the first population. In each generation, using *crossover* algorithm to produce a new generation with 6 times size of father population. And then choose the best $psize$ solutions as the new population.

It should be noted that all processes share a set *bestpop* which includes the best $bestsize$ solutions found by all processes. When poduce a new population, generation set needs to be combined with the *bestpop* and then choose the best $psize$.

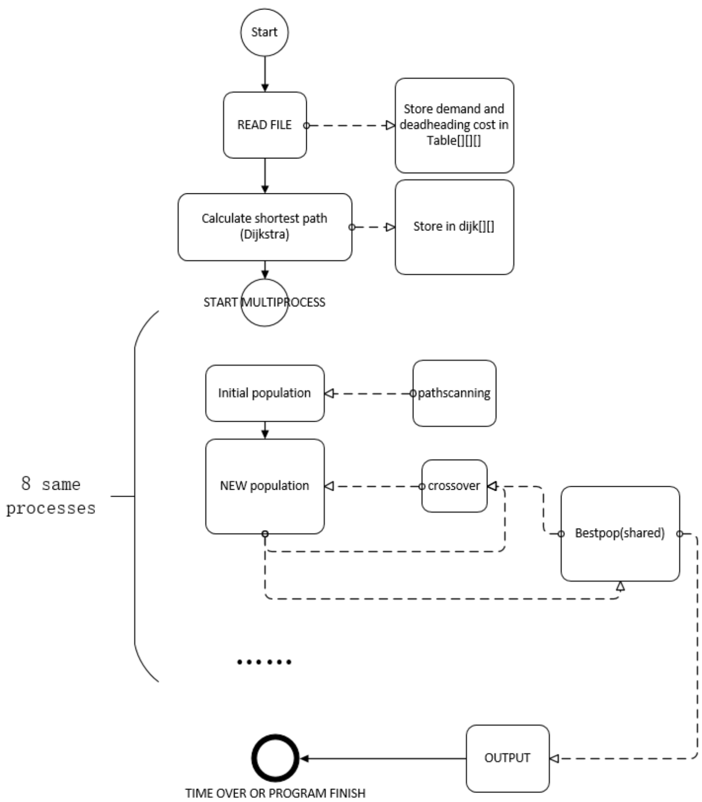

Fig. 1. Model Achitecture

D. Detail of algorithms

Pathscanning is a traditional algorthm to find a simple solution. To avoid falling into local optimum, I used two strategies in my program. The first is there is a 50% chance that a random task will be chosen as the first step in a route. And the second is when the vehicle pass the depot, discharge immediately which can be achieved by calculating the shortest path.

The better stayergy in pathscanning algoritm is also aim to increased randomness and avoid falling into local optimum.

At each iteration of MAENS, crossover is implemented by applying the sequence based crossover (SBX) operator to two parent individuals randomly selected from the current population. Each pair of parent individuals leads to a single offspring individual. [1]

Actually, the most difficult part is how to combine two genes from father and mother to create a new route and insert it to the original solution without any faults. Firstly, combine the two fragments to create a new route $R1$ and remove duplicate tasks in $R1$. After that we need to check whether it is a feasible route (whether it exceeds the CAPACITY). If so, pop tasks randomly until $R1$ is feasible. Then replace the original route in the original solution with $R1$ to create a new immature solution $Snew$. Check $Snew$, there pobabaly are some tasks not in $Snew$, put them in a set $lack$.

For each task in $lack$, try to insert it in any route of $Snew$ to check wether it is feasible after insetion. If so, intert it at the best position in that route and then try next task in $lack$.(best: least deadheading cost) When $lack$ is empty, it means $Snew$ is a new feasible solution and can be returned.

Algorithm 1 pathscanning

Output: $Route$

$free$ ¡- $Required$: a set of all tasks
$Route$ = []
k = -1
while $free$ is not empty **do**
  K += 1
  repeat
    r ← random (0,1)
    if this is the first step and r ¡ 0.5 **then**
      randomly choose a task as the first task to serve
    else
      choose the nearlest task $arcnext$ in $free$ which will not over the CAPACITY
      if there are two or more tasks satisfy the condition **then**
        use $better()$ to choose the better $arcnext$
      end if
    end if
    $Route$[k] append $arcnext$
    pop $arcnext$ from $free$
  until there are no tasks satisfy the capacity **or** the shortest route from $arcnext$ and the current position pass the $depot$
end while=0

III. Empirical Verification

A. Dataset

All datasets from the platform: $egl-e1-A$, $egl-s1-A$, $gdb1$, $gdb10$, $val1A$, $val4A$, $val7A$

And I also find some datasets in internet: $egl-s1-B$, $egl-s1-C$, $egl-s2-A$, $egl-s2-B$, $egl-s2-C$, $egl-s3-A$, $egl-s3-B$, $egl-s3-C$, $egl-s4-A$, $egl-s4-B$, $egl-s4-C$

B. Performance measure

Given a time, look at the difference between the solution given at the end of the program and the optimal solution.

Test envirment is given by CARP-Oj-Platform.

Algorithm 2 better

Input: $arc1$, $arc2$
Output: better task arc

// randomly choose one of five strategies
r ¡- random int in range [0,4]
if r == 0 **then**
  if distance from the $arc1$ to the depot ¿= $arc2$ **then**
    arc = $arc1$
  else
    arc = $arc2$
  end if
end if
if r == 1 **then**
  if distance from the $arc1$ to the depot ¡ $arc2$ **then**
    arc = $arc1$
  else
    arc = $arc2$
  end if
end if
if r == 2 **then**
  dem(t) is demand of task t
  sc(t) is deadheading cost of task t
  if dem($arc1$)/sc($arc1$) ¿= dem($arc2$)/sc($arc2$) **then**
    arc = $arc1$
  else
    arc = $arc2$
  end if
end if
if r == 3 **then**
  dem(t) is demand of task t
  sc(t) is deadheading cost of task t
  if dem($arc1$)/sc($arc1$) ¡ dem($arc2$)/sc($arc2$) **then**
    arc = $arc1$
  else
    arc = $arc2$
  end if
end if
if r == 4 **then**
  if vehicle is less than half-full **then**
    using strategy r = 0
  else
    using strategy r = 1
  end if
end if=0

C. Hyperparameters

$psize = 30$
$ubtrial = 50$
$opsize = 6 * psize$
$bestsize = 15$

D. Experimental results

See table 1.

Amoung these seven datasets, $egl - e1 - A$, $gdb1$, $gdb10$, $val1A$, I can get the optimal solution.

Algorithm 3 MA

Input: $psize$, $opsize$, $utrial$,$bestsize$
Output: A feasible solution

//Initialization
set the current population $pop = \emptyset$
while $|pop|$ ¡ $psize$ **do**
  Set the trial counter ntrail = 0
  repeat
    Generate an initial solution S_ini using $pathscanning()$
    $ntrial$ += 1
    $psize = |pop|$
  until S_init is not a clone of any solution S $\in$ pop or $ntrial$ = $ubtrial$
  if S_int is a clone of a solution S $\in$ pop **then**
    break
  end if
  $pop \leftarrow pop \cup S_init$
end while
while stopping criterion is not met **do**
  Set an intermediate population $pop_t = pop$
  for i $\in$ range[1, psize] **do**
    Randomly select two different solutions S1 and S2 as the parents from pop
    Apply the $crossover()$ operator to S1 and S2 to generate S_x
    Sample a random number r from the uniform distribution between 0 and 1
  end for
  if S_x is not a clone of any $S \in pop_t$ **then**
    $pop_t = pop_t \cup S_x$
  end if
end while=0

Algorithm 4 crossover

Input: $mother$, $father$
Output: son

$r1$ ¡- random int in range[0,length of $mother$)
$r2$ ¡- random int in range[0,length of $father$)
$route1$ from $mother \leftarrow mother[r1]$
$route2$ from $mother \leftarrow mother[r2]$
$r11$ ¡- random int in range[0,length of $route1$)
$r22$ ¡- random int in range[0,length of $route2$)
$gene11$ ¡- $route1[0 : r11]$
$gene12$ ¡- $route1[r11 :]$
$gene21$ ¡- $route1[0 : r22]$
$gene22$ ¡- $route1[r22 :]$
$son1$ = $combine(mother, gene11, gene22)$
$son2$ = $combine(father, gene21, gene12)$
son ¡- one has less total cost of (son1, son2)
=0

And for the biggest graph $egl - s1 - A$, I can also get a good solution.

The disadvantages of my program is that it doesn't perform

TABLE I
EXPERIMENTAL RESULTS

Dataset	time(s)	optimal	cost
$egl-e1-A$	120	3548	3548
$egl-s1-A$	120	5018	5145
$egl-s1-B$	120	6338	6684
$egl-s1-C$	120	8518	8880
$egl-s2-A$	120	9895	10694
$egl-s2-B$	120	13147	14536
$egl-s2-C$	120	16430	18088
$egl-s3-A$	300	10257	11132
$egl-s3-B$	300	13749	14905
$egl-s3-C$	300	17207	18775
$egl-s4-A$	300	12341	13773
$egl-s4-B$	300	16227	17876
$egl-s4-C$	300	20538	21991
$val1A$	60	173	173
$val4A$	60	400	406
$val7A$	60	273	284
$gdb1$	60	316	316
$gdb10$	60	275	275

well on small data sets and slow. For big datasets, it can give a solution that does not deviate much from the optimal solution if you give him enough time.

ACKNOWLEDGMENT

I would like to thanks my classmates who discussed algorithms with me which help me finish this project. And I also want to thanks for TA Yao Zhao, TA Zhiyu Fan who teached algorithms and guided me to research. Last I would like to thank forward to all the student assistances who will assess my codes and reports.

REFERENCES

[1] Tang K, Mei Y, Yao X, “Memetic algorithm with extended neighborhood search for capacitated arc routing problems[J],” IEEE Transactions on Evolutionary Computation, 2009, 13(5): 1151-1166.