Capacitated Arc Routing Problem Report

12110714 Jianan Xie

***Abstract*—CARP is known as capacitated arc routing problem, i.e., arc path problem with capacity constraints. It's a classical NP-hard arc routing problem. An algorithm called path scanning is used to find the local optimal answer and some supplemented methods are used to help the answer jump out of local optimum. It practically works not bad.**

***Index Terms*—CARP, capacitated, arc, routing.**

I. Introduction

THE routing problems can be divided into VRP(Vertex Routing Problem) and ARP(Arc Routing Problem). The arc routing problem can be roughly divided into three categories: Chinese postal routing problem, rural postal routing problem, and arc routing problem with capacity constraints(CARP).

Since Golden and Wong proposed the Capacity Constrained Arc Routing Problem (CARP) in 1981, 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. An arc routing problem is generally a planning problem in which the edges of a graph are served by a fleet of vehicles, each of which is assumed to be of the same model. Each edge must be provided by one vehicle and the service needs to be completed in one pass, all edges are allowed to be passed any number of times, each vehicle leaving the yard and returning to the starting point after the service.

The purpose of this project is to find the optimal answer of CARP in computing time.

II. Preliminary

CARP can be described as follows: consider an undirected connected graph $G = (V, E)$, with a vertex set V and an edge set E and a set of required edges (tasks) $T \subseteq E$. A fleet of identical vehicles, each of capacity Q, is based at a designated depot vertex $v_0 \in V$. Each edge $e \in E$ incurs a cost $c(e)$ whenever a vehicle travels over it or serves it (if it is a task). Each required edge (task) $\tau \in T$ has a demand $d(\tau) > 0$ associated with it.

The objective of CARP is to determine a set of routes for the vehicles to serve all the tasks with minimal costs while satisfying: a) Each route must start and end at v_0; b) The total demand serviced on each route must not exceed Q; c) Each task must be served exactly once (but the corresponding edge can be traversed more than once)

Thus, the solution to CARP can be described as:

$$s = (R_1, R_2, \ldots, R_m)$$

m is the number of routes (vehicles). The k^{th} route $R_k = (0, \tau_{k1}, \tau_{k2}, \ldots, \tau_{kl_k}, 0)$, where τ_{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 the demand of the dummy task are both 0 and its two endpoints are both v_0 (the depot). Moreover, since each task here 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, $\tau_{kt} = (head(\tau_{kt}), tail(\tau_{kt}))$, where $head(\tau_{kt})$ and $tail(\tau_{kt})$ represent the endpoints of τ_{kt}, and τ_{kt} is served from $head(\tau_{kt})$ to $tail(\tau_{kt})$.

Formally, the objective function of CARP is:

Algorithm 1 Minimize TC(s)

MINIMIZE $\mathrm{TC}(\mathbf{s})$
s.t (2) $\sum_{k=1}^{m} l_k = |T|$
(3) $\tau_{k_1 i_1} \neq \tau_{k_2 i_2}, \forall (k_1, i_1 \neq k_2, i_2)$
(4) $\tau_{k_1 i_1} \neq inv(\tau_{k_2 i_2}), \forall (k_1, i_1 \neq k_2, i_2)$
(5) $\sum_{i=1}^{l_k} d(\tau_{ki}) \leq Q, \forall k = 1, 2, \ldots, m$
(6) $\tau_{ki} \in T$

III. Methodology

The whole algorithm consists of three parts. Part I: Preparation. In this part, Floyd algorithm is used to derive the shortest distance between any two nodes on the graph, which speeds up calculating the costs during task transition. Part II: Construction. In this part, path scanning is used to derive a possible local optimal answer and five rules are applied on it to get five initial solutions. Part III: Improvement. In this part, some functions is used to improve the initial solution derived from path scanning. Hopefully, it may find the global optimal answer.

(1)Floyd algorithm is a classical method to solve MSSP (Muti Source Shortest Path) with negative weighted edges, which makes use of the idea of dynamic programming.

Algorithm 2 Floyd(G)

```
FLOYD(G)
    for  k = 1  to  n
        for  i = 1  to  n
            for  j = 1  to  n
                if  e_ij > e_ik + e_kj
                    e_ij > e_ik + e_kj
```

(2)Path scanning algorithm is a basic method to get a not bad initial solution of CARP, which supposes the vehicles will load the demands until it's full then return to the depot. It has some rules to use when multiple tasks are as close as to the vehicle, and this results in diverse initial solution to CARP. It intrinsically uses the greedy idea within single route.

Algorithm 3 Path Scanning

PATH SCANNING($\mathbf{G}$)
tasks of double direction in a list **free_list** in preparation
end = depot //record the current position of vehicle
task = **null**
tot_cost = 0
load = 0
tot_route = $\emptyset$
while **free_list** $\neq \emptyset$
d = ∞
for each **arc** in **free_list**
if d(arc) + load< Q
if $e_{end,begin(arc)} <$ d
task = arc, d = $e_{end,begin(arc)}$
else if d(arc) + load = Q and satisfy the rule we picked
task = arc
if task $\neq$ null
tot_route←tot_route.extend(task)
tot_cost←tot_cost + e_{task}
end←end(task)
free_list removes double direction of **task**
else
end←depot
load←0
tot_cost←tot_cost + $e_{end(task),depot}$

(3)Some improvement method is applied to mutate the initial solution derived from path scanning, like flipping, single_insertion and so on. They help the initial solution to pursue global optimum. Flipping is used to flip the task in the solution to see whether we can get improvements. Single insertion is used to insert single task picked from initial solution into other positions of route to see whether we can get improvements.

Algorithm 4 Flipping(G)

FLIPPING($\mathbf{G}$)
old_cost = 0, **new_cost** = 0
add old_cost with the cost of depot to first task and last task to depot
for each arc in solution route
old_cost← old_cost + $e_{end(lastarc),begin(arc)}$
flip arc
new_cost← new_cost + $e_{end(lastarc),begin(arc)}$
add new_cost with the cost of depot to first task and last task to depot
if new_cost ¡ old_cost
return new_cost

Algorithm 5 Single Insertion(G)

SINGLE INSERTION($\mathbf{G}$)
old_cost = 0, **new_cost** = 0
random pick a **task**in solution route, random pick a **position** to insert in solution route.
add old_cost with the cost of depot to first task and last task to depot
for each arc in solution route
old_cost← old_cost + $e_{end(lastarc),begin(arc)}$
inset the picked task into picked position
for each arc in solution route
new_cost← new_cost + $e_{end(lastarc),begin(arc)}$
add new_cost with the cost of depot to first task and last task to depot
if new_cost ¡ old_cost
return new_cost

Algorithm analysis: Floyd is a classical algorithm to get shortest path from any node to another node in graph with negative weighted edges, but costs O(n) time. Path scanning is a way to find a legal solution of CARP but at most situation it's not the optimum, so some other improvement methods are in need to enhance the solution derived from it. And it costs O(—free_list—) time. Improvement method like flipping and single insertion is just one operator to improve the initial solution and both costs O(—answer route—) time.

In main method, we just do as many times improvement algorithm on current optimal answer as possible within given time limit, then get a not bad solution at the end.

IV. EXPERIMENTS

The datasets used to test the algorithm as below:

name	vertices	depot	required edges	non-required edges	capacity
egl-e1-A	77	1	51	47	305
egl-s1-A	140	1	75	115	210
gdb1	12	1	22	0	5
gdb10	12	1	25	0	10
val1A	24	1	39	0	200
val4A	41	1	69	0	225
val7A	40	1	66	0	200

egl-e1-A and egl-s1-A are more complex than other datasets which can vividly reflect the improvement done on the algorithm.

The results of algorithm running on these datasets with time limit 5s:

name	result
egl-e1-A	4238
egl-s1-A	6382
gdb1	345
gdb10	295
val1A	188
val4A	431
val7A	324

Actually, this result doesn't live up to the expectation. The improvement of flipping and single insertion is not obvious. That's because the single insertion is implemented in a self version. It means it just does insertion within one single route. Generally, it can do insertion between different routes. Besides, there are some other more efficient improvement method like swap, 2-opt and so on, which haven't been implemented. Thus, this result is just a small step of improvement on path scanning.

V. Conclusion

In this project, I get more familiar with local search problem and have deeper insight into Genetic Algorithm and Simulated Annealing Algorithm. Actually, I have tried those improvement method and spare no effort to improve genetic algorithm. But regretfully, there exist some bugs and questions I haven't get through, so these failure attempts are not shown in the final submitted project. This experience enlightens me a lot and I will make more efforts on relevant local search problem and complete this project.