International Journal of Scientific & Engineering Research Volume 10, Issue 3, March-2019
ISSN 2229-5518

A newly developed algorithm based on tabu search for the Capacitated Arc Routing Problem (CARP)

Yi Zhao, Gexing Wang, Shuying Zhang

Abstract— The Capacitated Arc Routing Problem (CARP) is a difficult optimization problem in vehicle routing with unacceptable complexity by brute force. CARP is to find a solution that a given set of specified roads must be serviced by vehicles. Many search algorithms are introduced to solve CARP problem, including SA (simulated annealing algorithm), tabu algorithm, GA (Genetic Algorithm), MA (Memetic Algorithm). In this paper, a newly developed algorithm is proposed based on traditional tabu search algorithm. The computational results are shown in the experiment part of this paper, showing this algorithm can find high quality solutions efficiently.

Index Terms— tabu search, CARP, vehicle routing, optimization, decision making, genetic algorithm, simulated annealing

1 Introduction

The Capacitated Arc Routing Problem (CARP) is a difficult optimisation problem in vehicle routing with great amount of applications in real life. CARP is to find a solution that a given set of specified roads must be serviced by vehicles, while in the mean time not exceed the limit service capacity of the vehicles. In this paper, we only discuss the problem under the situation that all the roads (edges) in the problem are bidirectional and with same property. Many search algorithm are introduced to solve CARP problem, including SA (simulated annealing algorithm), tabu algorithm, GA (Genetic Algorithm), MA (Memetic Algorithm), etc... These algorithm either cannot find solution with high quality, or need long time to give out a solution with high quality. Through the algorithm proposed in this paper, a solution with good quality is produced within 2 minutes for a graph with 140 vertices and 190 edges.

2 Capacitated Arc Routing Problem

2.1 Problem Definition

The Capacitated Arc Routing Problem may be described as follows: for an undirected graph G = (V, E), with a set if required edges R ⊆ E. Vehicles are needed to serve the required edges. A set of identical vehicles, each of capacity Q, starts at the depot node D in the graph G. Each edge (vi, vj) in the graph comes with a cost cij indicates the cost of a vehicle goes through it. Each required edge (vi, vj) in the graph comes with a demand dij indicates the demand of a vehicle serves it. Each required edge need only to be serve once and allows unlimited times of passing through. A vehicle route must start and finish at the given depot D and the total demand of the route cannot exceed the capacity Q. The objective of CARP is to find a minimum cost set of vehicle routes that served all the required edges.

2.2 Problem Applications

CARP has a variety of applications in real life. For example, in the winter of many countries snow can be so high that vehicles cannot pass through, which means salt need to be put on the road to accelerate the melt of snow. Another typical application for CARP is the road rubbish collection in cities. CARP gives a great abstract model for these real problems.

TABLE I
SYMBOLS IN THE ALGORITHM

Symbol	Representation
D	vehicle depot
Q	vehicle capacity
G	graph
V	number of nodes in graph
E	number of edges
R_E	number of required edges
NR_E	number of not required edges
$penalty\_factor$	the value of penalty factor in the cost estimation

3 Methodology

3.1 Notations

TABLE I list out the symbols will be used in the algorithm.

3.2 Model Design

This algorithm is divided into several parts.
• Initialization: the part that deal with the raw data(graph), put the graph in computer memory
• Initialization Solution Generator: the part that generate the initial solution by basic path scanning algorithm
• variation: the part that perform the modification to generate new solution
• search: the part that perform the searching
• multi carp: the part to perform the multiprocessing work of the alogorithm

3.1 Detail of Algorithms

Algorithm 1 Floyd shortest path Algorithm

```
for each k ∈ [1, V] do
    for each i ∈ [1, V] do
        for each j ∈ [1, V] do
            dis[i][j] = min(dis[i][k], dis[k][j])
        end for
    end for
end for
```

Algorithm 2 Path Scanning

```
L ← ∅
S ← ∅
for all edges e required do
    L ← L ∪ e
    L ← L ∪ e_inReverseOrder
        global_best_solution ← current_solution
    end if
    if current_solution is a feasible solution and has less
    cost than global_best_solution then
        best_feasible_solution ← current_solution
    end if
    if global_best_solution kept infeasible for three round
    then
        penalty_factor = penalty_factor * 2
    end if
    if global_best_solution kept feasible for three round
    then
        penalty_factor = penalty_factor / 2
    end if
            end for
            if remain_cap remains enough for the vehicle and
            e is not the shortest required edge remain then
                random choose whether e is selected with equal
                probability
            else if remain_cap remains enough for the vehicle
            and e is the shortest required edge remain then
                update remain_cap to minus the demand of the
                edge
                S ← S ∪ e
            else
                solution ← solution ∪ route
                break
            end if
        end while
        if cost for new solution is less than the best solution
        ever found then
            update the best solution to the new solution
        end if
    end while
end for
```

Algorithm 3 Merge split

```
rand_a = random number
rand_b = random number different from rand_a
select the number rand_a and rand_b route from the
solution
get the edges in the routes
redo the path path scanning
```

Algorithm 4 Single insertion

```
for every route r in solution do
    for every edge e in route do
        insert e and e_inReverseOrder to any available
        positions in the solution
    end for
end for
```

Algorithm 5 Double insertion

```
for every route r in solution do
    for every consecutive two edges (e1, e2) in route do
        insert (e1, e2) and (e1, e2)_inReverseOrder to any
        available positions in the solution
    end for
end for
```

Algorithm 6 Swap

```
for every pair of edges in solution do
    reverse the position of e1 and e2
end for
```

Algorithm 7 Calculate estimated cost

```
maximum_exceed ← 0
for every route in solution do
    if total_demand of the edges in the route exceed Q then
        if the total_demand has larger exceed than maxi-
        mum_exceed then
            maximum_exceed ← total_demand - Q
        end if
    end if
end for
estimated_cost ← total_cost of the solution +
penalty_factor * maximum_exceed
```

ISSN 2229-5518

Algorithm 8 Search

```
penalty_factor = 2
get initial solution by path scanning
current_solution = initial_solution
global_best_solution = initial_solution
best_feasible_solution = initial_solution
while True do
    perform merge split to current_solution
    perform single insertion to current_solution
    perform double insertion to current_solution
    perform swap to current_solution
    current_solution ← best_solution from the four operator
    if current_solution has less cost than global_best_solution then
        global_best_solution ← current_solution
    end if
    if current_solution is a feasible solution and has less cost than global_best_solution then
        best_feasible_solution ← current_solution
    end if
    if global_best_solution kept infeasible for three round then
        penalty_factor = penalty_factor * 2
    end if
    if global_best_solution kept feasible for three round then
        penalty_factor = penalty_factor / 2
    end if
    if global_best_solution kept infeasible for three round then
        penalty_factor = penalty_factor * 2
    end if
    if penalty_factor ¡= 0.25 or penalty_factor ¿= 16 then
        current_solution = best_feasible_solution
    end if
    if time exceed given time or round have been more than 900 * ⌈ R_E ⌉ or best_feasible kept same for ten rounds then
        give solution by best_feasible_solution
    end if
end while
```

3 EXPERIMENTS

3.1 Dataset

This paper used gdb, val as additional dataset. gdb dataset has nodes from 7 to 27, required edges from 11 to 55 val dataset has nodes from 24 to 50, required edges from 34 to 97

3.2 Performance Measure

The performance is measured by comparing the cost of the algorithm proposed by paper and the best known.
Test enviornment:
Huawei magicbook
Ryzen 2500u 4 cores 8 threads
8G DDR4
512G SSD
The running time of all experiments are limited to 120 seconds and the the random seed is set to be 19.

3.3 Hyperparameters

The hyperparameters used in this paper include the penalty factor is initialized to be 2.
Through multiple test after applying different hyperparameter, this initial value of penalty factor is able to act well on the small dataset as to minimize to probability to go into the area of infeasible continuously and is not that big to affect the final result.

3.4 Experiment Results

The experimemt result is included in the TABLE II and TABLE III and TABLE IV in the appendix, shows the result on the dataset gdb and val and egl, respectively

CONCLUSION

The experiment result shows that the algorithm is able to almost or exactly get the best solution in relatively small graph (nodes from 7 to 50, edges from 11 to 97), the fluctuate on some gragh may due to the lack of multiple experiments and the only random seed is choosen, it may also indicates the merge split mentioned in this paper is still searching locally, more wide search operator is needed. The result also shows that this algorithm has the potential to access the best solution in larger graph (like graph in egl dataset), while the converge spped is relatively slow, it may due to the over-careful choosing of the parameters. A larger rate may change this situation.

REFERENCES

[1] Brandao, J., & Eglese, R. (2008). ”A deterministic tabu search algorithm for the capacitated arc routing problem.”, Computers & Operations Research. 35(4), 1112-1126.

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

[3] Golden, B. L., & Wong, R. T. (1981). ”Capacitated arc routing problems.”, Networks. 11(3), 305-315.

APPENDIX

TABLE II
RESULTS ON GDB DATASET

name	best known	average cost implemented
2	339	339
3	275	275
4	287	287
6	298	298
11	395	395
12	458	484
15	58	58
16	127	127
17	91	91
19	55	55

TABLE III
RESULTS ON VAL DATASET

name	best known	average cost implemented
1C	245	255
2C	457	463
3C	138	139
4A	400	400
4B	412	412
5C	473	476
6C	317	325
7C	334	337
8A	386	386
9A	323	326

TABLE IV
RESULTS ON EGL DATASET

name	best known	average cost implemented
e1-A	3548	3614
S1-A	5018	5185