A Heuristic Approach to Capacitated Arc Routing Problem using Simulated Annealing

Kemiao Huang, 11610728, *Undergraduate, CSE*

***Abstract*—The capacitated arc routing problem (CARP) is a classical problem and it's popular for many years since its wide applications in society. The simulated annealing is a regular and well-known for heuristic in NP hard problem. In this paper, a simulated annealing based heuristic approach is proposed for CARP. The most difficult point for simulated annealing is to set the parameters to control the balance between time cost and mutation effectiveness. A novel time control is proposed for the simulated annealing to get the output before time out.**

***Index Terms*—Capacitated arc routing problem (CARP), dynamic time control, local search, optimization, simulated annealing**

I. Preliminaries

THE Objective of this project is to realize an effective algorithm to find a solution for CARP in certain time as good as possible.

A. Introduction

1) Problem Description: Arc Routing is the process of selecting the best path in a network based on the route. There are many types of arc routing. Each one has different goal and heuristic. All of them are NP-hard problems. The capacipated arc routing problem is one of the branches of arc routing problem. It consists of an undirected graph with required edges and non-required edges. Each edge has its demand and cost. There are a number of vehicles with maximum capacities. Each vehicle should starts from the depot node and tries to meet the demand of each edge and returns depot without exceeding the its capacity. The goal is to minimise the cost for the vehicles to meet all the demands on each edge.

2) Problem Application: It is suitable to adopt the approach based on CARP in which demands are set on arcs. For example, urban waste collection[1], post delivery, mail delivery where demand is concentrated and road sweeping where road section itself is the target of service.

II. Methodology

To get the initial solution for simulated annealing heuristic, shortest path based path scanning is used. To realize the mutation for solutions, three classical moving operators are used, which are single insertion, swap and two-opt. To control the time cost of the whole procedure, rather than use sampling to set the parameter at the first time, a dynamic fixing approach is used during the simulated annealing process.

A. Notations

The input of CARP contains the vehicle capacity denoted by Q, a mixed graph $G = (V, E, A)$, with vertices denoted by V, edges denoted by E and required arcs denoted by A. Moreover, the depot for the routing is denoted by D and the shortest-distance matrix is denoted by S.

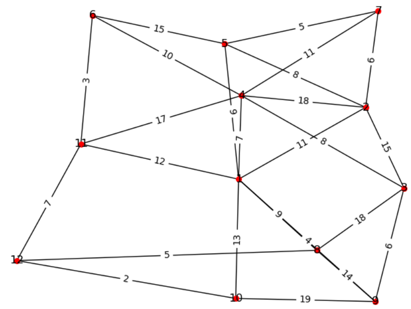

Fig. 1: The graph sample for CARP (gdb10)

B. Data Structures

1) Shortest-Distance Matrix: Since arc routing problem doesn't need the attribute for the vertices but the demand and cost of edges instead, the shortest-distance matrix is used without storing the graph information like connectivity, leaf and parent.

2) List: The python list is used for storing, inserting and popping the arcs, routes and solutions.

C. Model Design

Basically, a CARP solution contains its route list and quality (i.e. total cost), a route contains a required arc list, load and cost, an arc contains its begin vertex, end vertex, demand and cost. For easier implementation, the arcs have fixed begin and end vertices with another boolean variable for declaring the direction of the arc.

TABLE I: Class Declaration

Classes	Attributes
Arc	begin_index, end_index, load, cost, reverse
Route	required_arc_list, cost
Solution	route_list, quality

Firstly, read the input file and build the corresponding arcs and set the basic parameters. Secondly, Floyd algorithm is used for constructing the n by n shortest-distance matrix. Next, the initial solution comes out from doing path scanning for all the required arcs. Then the iteration parameter for the simulated annealing is computed according to the remaining time and start annealing. The solution comes out by annealing is the best one. Finally, print the solution in the right format.

D. Algorithms

1) Floyd Warshall Algorithm: This algorithm is used to construct the shortest path between each vertex. By contrast, Dijkstra algorithm theoretically has smaller time complexity by using Fibonacci heap or pairing heap as its priority queue[3]. However, Floyd algorithm is more easily to implement and the difference of practical time cost is very small. Therefore, Floyd algorithm is chosen for this procedure.

Algorithm 1 Floyd-Warshall

Input: V, E
Output: shortest path matrix S
$dist \leftarrow n \times n$ array of minimum distances initialized to ∞
for $v \in V$ **do**
  $dist[u][v] \leftarrow 0$
end for
for edge$(u, v) \in E$ **do**
  $dist[u][v] \leftarrow$ cost(u, v)
end for
for $k \leftarrow 1$ to n **do**
  for $i \leftarrow 1$ to n **do**
    for $j \leftarrow 1$ to n **do**
      if $dist[u][v] > dist[i][k] + dist[k][j]$ **then**
        $dist[u][v] \leftarrow dist[i][k] + dist[k][j]$
      end if
    end for
  end for
end for
return $dist$

2) Path Scanning: In order to get the initial solution as good as possible in short time, the classical approach using path scanning is used. To increase the randomness and get better solution, the traditional five rules[2] for choosing the same distance arcs are not used. Instead, the arcs with the same distance from the current source are pure randomly chosen. The formal initial solution is come out by choosing the best of ten outputs of path scanning.

3) Single Insertion: The single insertion operator is a classical operator for mutation. Since simulated annealing algorithm should keep only one solution as the current solution, the solutions without feasibility are supposed to be discarded. However, for all the operators in this paper, the output solutions are all keep with feasibility. Single insertion is randomly choose one arc and then try to insert it into another route. If it cannot, just insert into its own route with different position.

4) Swap: The swap operator is just simply swap the random two arcs and recombined with the less cost approach. Again, it tries to swap between two different routes.

5) 2-opt: In this project, the 2-opt for single route and double routes are both used. For randomness, the 2-opt for double routes have higher priority.

6) Simulated Annealing: The simulated annealing algorithm with dynamic time control is proposed in this paper. Since there are a time limit for this project, the annealing procedure should finish before time out. The iteration times for mutation at each cooling process is re-computed by the remaining time and the speed of the machine platform at each cooling period.

III. Empirical Verification

A. Dataset

Considered that there are enough testing data for this project, I didn't use the other datasets from the website. I only used datasets that the course provided.

B. Performance measurement

The solution quality and the time cost are considered to measure the performance. In the same time out limitation, the process with the lest quality has the best performance.

Algorithm 2 Path Scanning

Input: required arc list $free$
Output: route R
$R \leftarrow$ new empty route
$i \leftarrow D$
while true do
  $\overline{d} \leftarrow \infty$
  $\overline{u} \leftarrow \varnothing$
  for $u \in free$ **do**
    if $R.load + u.demand > Q$ **then**
      continue
    end if
    if $dist[i].begin < \overline{d}$ **then**
      $\overline{d} \leftarrow dist[i].begin$
      $\overline{u} \leftarrow u$
      $reverse \leftarrow$ **false**
    else if $dist[i].end < \overline{d}$ **then**
      $\overline{d} \leftarrow dist[i].end$
      $\overline{u} \leftarrow u$
      $reverse \leftarrow$ **true**
    else if $random =$ **true then**
      if $dist[i].begin = \overline{d}$ **then**
        $\overline{u} \leftarrow u$
        $reverse \leftarrow$ **false**
      else if $dist[i].end = \overline{d}$ **then**
        $\overline{u} \leftarrow u$
        $reverse \leftarrow$ **true**
      end if
    end if
  end for
  if $\overline{u} \neq \varnothing$ **then**
    $R.arc\_list.append(\overline{u}, reverse)$
    if $reverseis$**false then**
      $i \leftarrow u.end$
    else
      $i \leftarrow u.beign$
    end if
    $R.load \leftarrow R.load + u.demand$
    $R.cost \leftarrow R.cost + u.cost$
    remove u from $free$
  else
    break
  end if
end while
$R.cost \leftarrow R.cost + S[i][D]$
return R

C. Hyperparameters

In order to let the initial solution be as good as possible, 10 times path scanning are implemented. To let the annealing start from state which has enough randomness, the start temperature is set as 1000. The end temperature is set as tradition, 0.0001. Since it has adjustment for iteration times in the annealing, the cooling factor is not important but it should be set as fixed and high enough. Therefore, the cooling factor α is set as 0.99. The initial iteration times for one cooling process is set as 50. This is produced by the general testing.

D. Experiment Results

The test result in my Dell laptop is shown in the table.

Algorithm 3 Single Insertion

Input: input solution old_sln
Output: new solution new_sln
  $remove\_route \leftarrow$ a random route from $old\_sln.route\_list$
  $remove\_arc \leftarrow$ a random arc from $remove\_route.arc\_list$
  for r in $old\_sln.route\_list$ **do**
    if $route.load + remove\_arc.demand \leq Q$ **then**
      $insert\_route \leftarrow r$
    end if
  end for
  if $insert\_route \neq \varnothing$ **then**
    $pos \leftarrow$ a random position in $insert\_route$
  else
    $pos \leftarrow$ a random position in $remove\_route$
    insert $remove\_arc$ to pos with the direction that minimizes cost
  end if
  update the loads and costs of $route\_list$
  $new\_sln.route\_list \leftarrow route\_list$
  return new_sln

Algorithm 4 Swap

Input: input solution old_sln
Output: new solution new_sln
  $route1, route2 \leftarrow$ two random routes from $old\_sln.route\_list$
  $arc1 \leftarrow$ a random arc from $route1.arc\_list$
  for arc in $route2.arc\_list$ **do**
    $load1 \leftarrow route1.load + demand2 - demand1$
    $load2 \leftarrow route2.load + demand1 - demand2$
    if $load1, load2 < Q$ **then**
      $arc2 \leftarrow arc$
    end if
  end for
  if $arc2 = \varnothing$ **then**
    $arc2 \leftarrow$ another random arc in $route1$
  end if
  swap $arc1$ with $arc2$ with directions that minimize the costs
  update the loads and costs of $route\_list$
  $new\_sln.route\_list \leftarrow route\_list$
  return new_sln

Algorithm 5 2-opt

Input: input solution old_sln
Output: new solution new_sln
  // 2-opt for two routes
  $route1, route2 \leftarrow$ two random routes from $old\_sln.route\_list$
  cut $route1, route2$ into four halves
  if four halves can be combined without exceeding Q limit **then**
    $new\_route1, new\_route2 \leftarrow$ the recombination with less cost
    update the loads and costs of $route\_list$
    $new\_sln.route\_list \leftarrow route\_list$
  else
    // 2-opt for one route
    $sublist \leftarrow$ a random slice of $route1$
    reverse $sublist$ and insert back into $route1$
    update the cost of $route\_list$
    $new\_sln.route\_list \leftarrow route\_list$
  end if
  return new_sln

Algorithm 6 Simulated Annealing

Input: initial solution $init\_sln$
Output: best solution $best\_sln$
  $cur\_sln \leftarrow init\_sln$
  $best\_sln \leftarrow init\_sln$
  $coe \leftarrow start\_temp/end\_temp$
  $N \leftarrow$ a suitable check frequency
  $cur_temp \leftarrow start\_temp$
  for i from 1 to N **do**
    start timer
    $coe \leftarrow coe/\sqrt[N]{coe}$
    $cnt \leftarrow 0$
    while $cur\_temp > coe \times end\_temp$ **do**
      for m from 1 to M **do**
        $new\_sln \leftarrow$ apply operators on cur_sln
        $\Delta cost \leftarrow new\_sln.quality - cur\_sln.quality$
        if $\Delta cost < 0$ **then**
          $cur\_sln \leftarrow new\_sln$
          if $new\_sln.quality < best\_sln.quality$ **then**
            $best\_sln \leftarrow new\_sln$
          end if
        else if random $< exp(-\Delta cost/cur\_temp)$ **then**
          $cur\_temp \leftarrow new\_sln$
        end if
      end for
      $cur\_temp \leftarrow cur\_temp \times \alpha$
      $cnt \leftarrow cnt + M$
    end while
    stop timer
    // Fix the iteration times in each future cooling process
    $M \leftarrow$ (remaining time $\times$ cnt) / (time cost $\times$ (-log(coe, α))
  end for
  return $best\_sln$

TABLE II: Test Results

Dataset	Time out	Time cost	Initial	Final	Optimal
val1A	30	27.88	200	180	173
val4A	30	27.10	470	405	400
val7A	30	26.12	334	284	277
gdb1	30	25.68	341	316	316
gdb10	30	25.67	293	275	275
egl-e1-A	30	25.96	4464	3608	3548
egl-e1-A	60	52.11	6612	5437	5018

E. Conclusion

The simulated annealing is just an heuristic approach for CARP. In this project, the time out controlling is as good as it was expected. However, it still cannot produce the best solution for a big dataset even the time out limitation is very loose. The biggest reason is that the operators used in the project are not so effectiveness and they are easily generate the local optimal solution at last. If a more suitable operator is used, the result should be better.

IV. References

[1] K. Tang, Y. Mei and X. Yao, ”Memetic Algorithm With Extended Neighborhood Search for Capacitated Arc Routing Problems,” in IEEE Transactions on Evolutionary Computation, vol. 13, no. 5, pp. 1151-1166, Oct. 2009.
[2] A. Corberan, F. Laporte, ”Arc Routing Problems, Methods, and Applications,”, 2014

[3] W. Zhang, C. Jiang and Y. Ma, ”An Improved Dijkstra Algorithm Based on Pairing Heap,” 2012 Fifth International Symposium on Computational Intelligence and Design, Hangzhou, 2012, pp. 419-422.