Application of simulated annealing algorithm in the CARP problem

He Zhu, Southern University of Science and Technology

***Abstract*—Artificial Intelligence Course Project II, this paper use simulated annealing algorithm to solve the N-P hard Capacitated Arc Routing Problem.**

***Index Terms*—simulated annealing algorithm, Capacitated Arc Routing Problem, path-scanning,**

I. Introduction

CARP, which is Capacitated Arc Routing Problem in short, can be describe 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:

1) Each route must start and end at v_0;
2) The total demand serviced on each route must not exceed Q;
3) Each task must be served exactly once (but the corresponding edge can be traversed more than once)

This problem can be applied to the real-life problems of routing city garbage trucks, routing city sprinklers, and routing winter de-icing trucks. The aim of this project is to use simulated annealing algorithm to arrange the route of vehicles to obtain a cost that is as small as possible while satisfying all services.

II. Preliminary

Before using the simulated annealing algorithm, we need to define the objects of simulated annealing. For the problem studied in this paper, since it is essentially an optimization problem of routing, we need to generate some routes first. After generating some initial routes, we can use simulated annealing algorithm to reorganization those routes to minimize the total cost while satisfying all services. Some terminology and notation using throughout this report are in table one.

The problem can be formulated as an optimization problem, which is specified by a tuple $(t_0,\ \Delta,\ \xi,\ R,\ k)$, which means we need to find one route planning Δ with the minimum cost ξ in solution space R during the process of simulated annealing with a initial temperature t_0 and a rate coefficient k

TABLE I: The symbol table.

symbol	definition
T	Initial temperature of simulated annealing
k	A coefficient of temperature in simulated annealing
N	The number of vertices in graph
δ_i	A path of one route
Q	Capacity of one vehicle
Γ	Total server demands of one route
$E_{i,j}$	An edge from vertex i to vertex j
$C_{i,j}$	Minimum cost from vertex i to vertex j
$S_{i,j}$	An server edge from vertex i to vertex j
D_S	The demand of an server edge S
$M(i,j)$	Shortest distance between i and j

III. Methodology

A. Preparation

Before start our algorithm, we have to get all information about target graph and calculate the shortest path between any two points. Here we use Floyd-Warshall algorithm and below is the pseudo-code:

Algorithm 1 Floyd

```
for k in range(N + 1) do
  for i in range(N + 1) do
    for j in range(N + 1) do
      if i = j then
        C_{i,j} = 0
      end if
      if C_{i,j} > C_{i,k} + C_{k,j} then
        C_{i,j} = C_{i,k} + C_{k,j}
      end if
    end for
  end for
end for
return C
```

The complexity of this algorithm is $O(n^3)$, but this algorithm has clear logic and concise code. The main factor which can influence its performance is the complexity of the input undirected graphs.

B. Path-scanning algorithm

First, we give a definition of these three solution spaces:

$$\forall \Delta \in G \tag{1}$$

If any route δ_i in one route planning Δ and any server edge $S_{i,j}$ satisfy:

$$\Gamma_\delta \leq \zeta \tag{2}$$

$$S_{i,j} \in \Delta \tag{3}$$

the solution Δ can be seen as a solution in solution space R and meanwhile we have the formula below:

$$G = R \cup U \tag{4}$$

As mentioned before, we should generate some routes first, which means we need to get a solution in solution space R and in this topic, we choose path-scanning algorithm to obtain that initial solution. The following pseudo-code shows the process of path-scanning:

Algorithm 2 Path-Scanning

```
free ← D_{S_{i,j}}
route ← nulllist
while free not null do
  curPos ← v_0
  curLoad ← 0
  path ← nulllist
  while True do
    candidate ← getEdge(free, curLoad)
    if not candidate then
      break
    end if
    verticesList ← getVertices(cadidate)
    nextPos ← min(verticesList)
    nextEdge ← random edge in cadidate at nextPos
    curLoad = curLoad + D_{nextEdge}
    free.remove(nextEdge)
    path.add(nextEdge)
  end while
  route.add(route)
end while
return route
```

The complexity of this algorithm is $O(n^2)$. This algorithm can help you to make a good start by giving you a excellent solution. The main factor which can influence its performance is the complexity of the input undirected graphs.

C. Simulated annealing algorithm

After we generate a solution by path-scanning, the next step is that using simulated annealing algorithm to optimize that solution by some manipulations. First, we give the description of simulated annealing algorithm by pseudo-code.

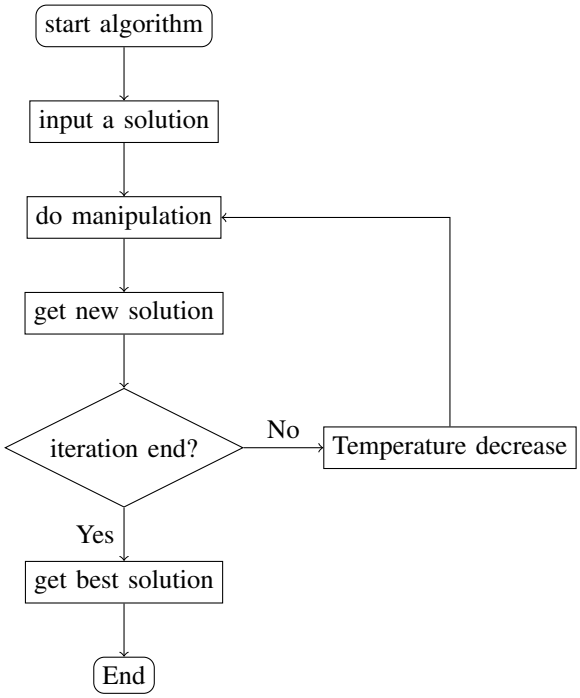
During the simulated annealing, we will do following four different manipulations to optimize the solution to get a better result and the solutions obtained are guaranteed to be in solution space R:

- Flip
- Single-injection
- Swap
- 2-opt

Here we use a flow chart to show the overall process.

Algorithm 3 SA

```
bestCost ← getCost(curRoute)
T ← T_0
ratio ← 0.99999
while t != 0 do
  newRoute = Op(curRoute)
  newCost = getCost(newRoute)
  if newCost ≤ bestCost then
    curRoute ← newRoute
    bestCost ← newCost
  else
    p = (bestCost − newCost)/kt
    randomNum ← random(0, 1)
    if randomNum < exp(p) then
      curRoute ← newRoute
      bestCost ← newCost
    end if
  end if
  T = T ∗ ratio
end while
return curRoute, bestCost
```



The above figure shows the basic flow of the process of simulated annealing in this topic. In the manipulation process, we used the four previously mentioned methods. In the process of get new solution, we will check the current temperature and use this and factor k as a basis for accepting a worse solution.

The complexity of this algorithm is $O(n)$. This algorithm have a good performance in decreasing the input cost. The main factor which can influence its time complexity is the initial temperature and the complexity of the input undirected graphs.

IV. Experiments

A. Setup

CPU: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, Main Memory: 16GB, Python version: 3.10, NumPy version: 1.23.3, IDE: PyCharm 2022.2.1. Initial temperature t_0 is 200000, coefficient k is 0.005, the selection ratio of different manipulations are 25 %.

B. Results

All results are generated by myself. The Experiments Environment is based on the basic parameter settings. In each Experiment we just restart the program to get a new cost of one single graph and repeat the process to calculate an average result and finally comparing the results with different parameters. Below are graphs of results of different experiments:

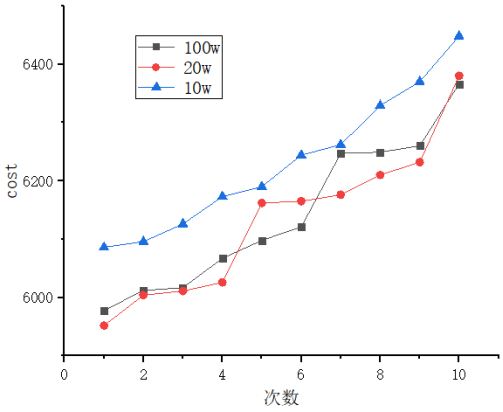


Fig. 1: Result of experiment 1

This graph show ten results of graph egl-s1-A with three different initial temperature while the coefficient is $k = 0.001$. From this result we can know that if we ignore a best result and a worst result, we can find that the results of t_0 =20w and t_0 =100w are almost the same but are better than the results of t_0 =10w.

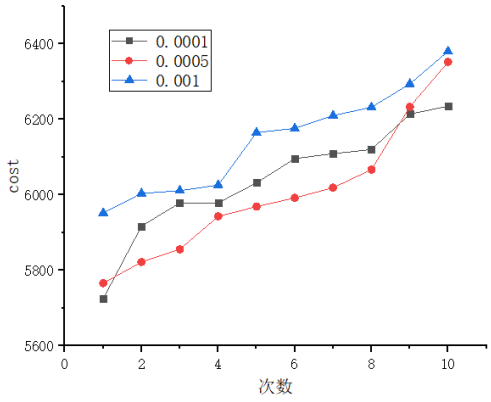


Fig. 2: Result of experiment 2

This graph show ten results of graph egl-s1-A with three different coefficients while the initial temperature is t_0 =20w. From this result we can know that if we ignore a best result and a worst result, we can find that among these three different coefficients, when the coefficient = 0.005, the result is better than others. This graph show twenty results of graph egl-s1-

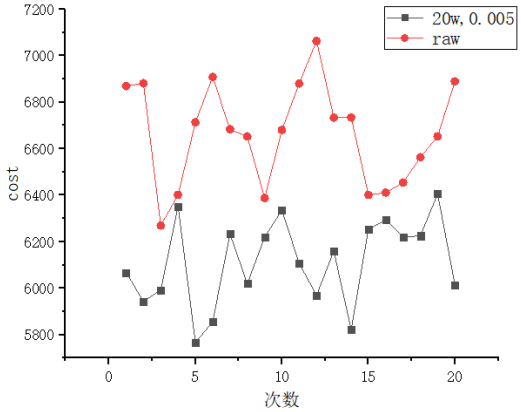


Fig. 3: Result of experiment 3

A with two different generation ways. The raw one just use path-scanning while the other one use simulated annealing base on its corresponding path-scanning result and the initial temperature is t_0 =20w and the coefficient is k =0.005. From this figure, we can find that the main factor that influence the final result is the process of simulated annealing, since even if the initial solution is better, the final solution may be worse after simulated annealing, which indicates that a worse initial solution has better potential to become better and is more possibly to generate a better result. But it has to be mentioned that the initial solution sets the lower limit of the final result.

C. analysis

The algorithm designed so far has been able to take into account the importance of route planning. Finally my input initial temperature is t_0 =20w and the coefficient k =0.005. Compared to the initial solution, the final solution can reduce the cost by about ten percent with these parameters. My path-scanning can generate an initial solution with average cost about 6600, and the best result of graph egl-s1-A is about 5500. Although I find the result of initial temperature t_0 =100w is the best, but due to the time limitation, I have to abandon it and the time cost can be ten times of initial temperature t_0 =10w which also can be known from the part C of Methodology. Overall, I think my design is good and it meet my expectations.

V. Conclusion

I use path-scanning algorithm and simulated annealing algorithm to solve the CARP. The experiment result seems good but there are still many parts that can be optimized.

1) In terms of parameter setting, I did not test if different ratio can get a better result and the experiments of input parameters is uncritical.
2) In terms of design, since we can see that different experiment with the same parameters can generate vastly

different results, to narrow down it, we can use multi-process to always get a minimum result to reduce the problems caused by randomness.

3) In terms of manipulations, we can add more effective manipulations or do those manipulations with purpose to easier get a better solution in each manipulation.
4) In terms of time complexity, I can choose Dijkstra instead of Floyd algorithm to reduce the time complexity.

1) The gradient of the game module: Let $g(\theta, a) := \nabla_a(\theta, a)$. check if derivative w.r.t. a makes sense

Let h be the local mapping from params Θ to solution A, i.e., solving $g(\theta, h(\theta)) = 0$. Let $Dg(\theta, a) = [J_1(\theta, a), J_2(\theta, a)]$, with J_1 corresponding to θ and J_2 to a.

Then, under some conditions,

$$\nabla h(\theta) = -[J_2(\theta, h(\theta))]^{-1} J_1(\theta, h(\theta))$$

REFERENCES

van Laarhoven, P. J., amp; Aarts, E. H. (1987). Simulated annealing. Simulated Annealing: Theory and Applications, 7–15. https://doi.org/10.1007/978-94-015-7744-1_2

Kirkpatrick, S., Gelatt, C. D., amp; Vecchi, M. P. (1983). Optimization by simulated annealing. Science, 220(4598), 671–680. https://doi.org/10.1126/science.220.4598.671