Artificial Intelligence Project Report: CARP

Yuwei Xiao
12012902
Department of Computer science and Engineering

***Abstract*—This article is a project report of CS303 Artificial Intelligence. The topic of the project is *Heuristic Search for Capacitated Arc Routing Problem (CARP)*. In this report, I will introduce the CARP problem, the search methods I developed and the tuning process I made. The report will also cover the analysis and experiments of the problem.**

***Index Terms*—CARP, Heuristic Search, Genetic Algorithm, Simulated Annealing**

I. Introduction

The capacitated arc routing problem (CARP) is an important and practical problem in the OR literature. In short, the problem is to identify routes to service (e.g., pickup or deliver) demand located along the edges of a network such that the total cost of the routes is minimized. In general, a single route cannot satisfy the entire demand due to capacity constraints on the vehicles.

CARP belongs to the set of NP-hard problems; consequently numerous heuristic and metaheuristic solution approaches have been developed to solve it.

This report will primarily show an implementation of a path-scanning based CARP agent, while analysing its performance. Meanwhile exploring how the various CARP heuristics interact.

II. Preliminary

CARP has numerous applications in reality, therefore we can transfer the abstract definition of CARP into a real problem to better comprehend it. For instance, how do we design the route and collecting sequence of a garbage tractor with limited capacity? Basically, we want the tractor collect as much garbage as possible before it have to return to the depot. And for the route it take, we want the distance to be minimize.

A. Problem Formulation

Table 1 list the notations used in formulating the problem and through out the article. Meanwhile, I will use these notations to formulate the goal of agent in algorithm description.

The formulation of the problem: given a graph *G*, a depot vertex v_0, a vehicle with capacity C and a task set *T*, determining the best route *r*, in order to minimize the total cost c of finishing all tasks in *T*.

That is to say, the problem can be formulated as a tuple (G, v_0, C, T), and the objective of the agent is to minimize c while finishing all tasks in *T*.

TABLE I
Notations

Notation	Name	Explanation
G	graph	undirected graph
D	distance matrix	matrix that records the distance between nodes
c	cost	cost of the edge
d	demand	demand of the task
C	capacity	capacity of the vehicle
T	Tasks	set that contains all tasks
d_{task}	distance	distance from current vertex to task
R	best route	The best route of the search result

III. Methodology

The design of the agent for CARP generally focus on the implementation of path-scanning and its improvement. I implemented path-scanning with different strategy. I also implemented several heuristics and tried to combine them together to improve the agent's performance. The general workflow of the proposed method will be shown in subsection A. Followed by the details of the algorithm in subsection B. The heuristic functions will be introduced in subsection C. The improving methods will be shown in subsection D. And Complexity and Optimality will be discussed in subsection E.

A. General Work Flow

The algorithm can be generally divided into 3 parts: a. use Dijkstra algorithm to get the minimum distance between each vertexes; b. use Path-scanning heuristic search to get a basic solution of route; c. iteratively use one of the five methods to improve the solution until exceeding the time limit

B. Path-scanning

Path-scanning is a heuristic search method for CARP, which aims at finding a set of routes to finishing all tasks while not exceeding the vehicle's capacity. There are multiple heuristic principles of path-scanning and I have implemented several of them. The comparision between them will be shown in the coming section.

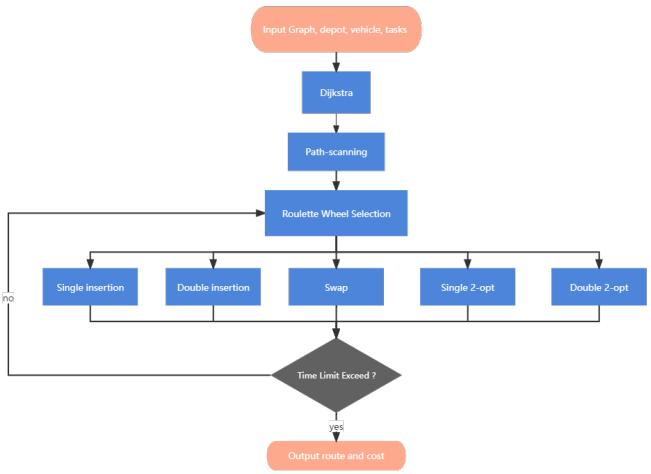

Fig. 1. **General Work Flow of my agent.**

Algorithm 1 path_scanning(G, v_0, D, C, T)

Input: The undirected graph, G; The depot vertex, v_0; The distance matrix generated by Dijkstra, D; The capacity of vehicle, C; The tasks set, T

Output: The best route according to the search result, R

R = []
while Tasks in T are not all finished **do**
  $best\_task$ = Heuristic(v_0, D, C)
  R.append(best_task)
end while
return R;

C. *Heuristics*

In path-scanning, heuristic is an important factor of improving the AI's performance. In this CARP project, I implemented 2 main kinds of heuristics and different criteria of the main heuristic. One is criteria based path scanning, the other is ellipse-rule based path scanning. These heuristics will be introduced in detail and a pseudo-code will be given. Moreover, the evaluation on different heuristics will be discussed as well.

- **Criteria-based Path-scanning:**
  This kind of path-scanning is based on 5 criteria that proposed by *Golden et al.* [1]. This criteria may be stated as follows:

  1) maximize the distance from the task to the depot
  2) minimize the distance from the task to the depot
  3) maximize the term $\frac{dem(t)}{sc(t)}$, where $dem(t)$ and $sc(t)$ are demand and serving cost of task t, respectively
  4) minimize the term $\frac{dem(t)}{sc(t)}$
  5) use criteria 1 if the vehicle is less than half- full, otherwise use criteria 2

Algorithm 2 Heuristic(v_0, D, C)

Input: The depot vertex, v_0; The distance matrix generated by Dijkstra, D; The capacity of vehicle, C; The tasks set, T

Output: The best task of the result, $best\_result$

$best\_task$ = null
for task in T **do**
  if task's demand acceptable for the vehicle **then**
    if distance to task edge is less than to best task edge **then**
      $best\_task$ = task
    end if
  else if distances are equal **then**
    if task is better than $best\_task$ by selected criteria **then**
      $best\_task$ = task
    end if
  end if
end for
return $best\_task$;

- **Ellipse-rule-based Path-scanning**
  This path-scanning method is proposed by *Luis Santos et al.* [2], which is similar to the method proposed by *Belenguer et al.* [3], which randomly selects tied edges and solves each problem k times. However, this path-scanning method use an "ellipse rule" when the load of vehicle is near the capacity. To be more specific, intuition suggests that as a vehicle's load approaches its capacity, its route should stay closer to the depot to reduce its cost to return to the depot when full. The "ellipse rule" is designed to implement this intuition. When the vehicle is near the end of a route, this rule forces the vehicle to service only edges near the shortest path between the last serviced edge and the depot.

Algorithm 3 Heuristic(v_0, D, C)

Input: The depot vertex, v_0; Distance matrix generated by Dijkstra, D; The capacity of vehicle, C; The tasks set, T

Output: The best task of the result, $best\_result$

$best\_task$ = null
for task in T **do**
  if task's demand acceptable for the vehicle **then**
    if average demand $\times\alpha \leq$ remaining capacity **then**
      if $d_{task} \leq d_{best\_task}$ **then**
        $best\_task$ = task
      end if
    end if
  else
    if task cost $\leq$ average task cost **then**
      $best\_task$ = task
    end if
  end if
end for
return $best\_task$;

D. *Improving Methods*

After dijkstra and path scanning, we can get a basic solution of CARP. However, since path scanning is a heuristic search method, it is likely to have better solution. So, I developed several improving methods to find better solution based on the solution we get from path scanning.

The improving methods are listed as follows:

- **Flip:** Randomly choose a task, which can be viewed as a tuple (v_{src}, v_{dst}), then flip it to (v_{dst}, v_{src}).
- **Swap:** Randomly choose 2 tasks and swap their positions in R.
- **Insertion** Randomly choose 1 or 2 task(s) and a new position, change the task(s) position to the new position in R.
- **2-OPT** Randomly choose 2 or 4 tasks in R, match them in a one-to-one basis and reversed the tasks between the matched tasks if the direction is not correct.

The pseudocode of the methods is shown below and because of the logic of implementing single/double insertion(2-opt) are similar, they're combined as an individual method,

Moreover, since it's hard to balance the importance of various methods, I set up several experiments and study existing work that related to the methods. The specific result of these experiments and work will be shown in the next section.

Algorithm 4 flip(R)

Input: The best route, R;
Output: The best route, R
$R_{copy} = deepcopy(R)$
flip_task = randomly select a task from R_{copy}
flip_task = reversed(flip_task)
if cost after change $\leq$ cost before change **then**
    $R = R_{copy}$
end if
return R;

Algorithm 5 swap(R)

Input: The best route, R;
Output: The best route, R
$R_{copy} = deepcopy(R)$
swap_task_1 = randomly select a task from R_{copy}
swap_task_2 = randomly select a task from R_{copy}
temp = swap_task_1
swap_task_1 = swap_task_2
swap_task_2 = temp
if cost after change $\leq$ cost before change **then**
    $R = R_{copy}$
end if
return R;

Algorithm 6 insertion(R)

Input: The best route, R;
Output: The best route, R
$R_{copy} = deepcopy(R)$
insert_task = randomly select a task from R_{copy}
position_task = randomly select a task from R_{copy}
remove insert_task from R_{copy}
append insert_task after position_task
if cost after change $\leq$ cost before change **then**
    $R = R_{copy}$
end if
return R;

Algorithm 7 2-opt(R)

Input: The best route, R;
Output: The best route, R
$R_{copy} = deepcopy(R)$
split_task_1 = randomly select a task from R_{copy}
split_task_2 = randomly select a task from R_{copy}
in_between_tasks = tasks between split_tasks in R_{copy}
in_between_tasks = reversed(in_between_tasks)
if cost after change $\leq$ cost before change **then**
    $R = R_{copy}$
end if
return R;

E. *Complexity*

- **Time Complexity Analysis**
  Since our method is consist of several parts, we should calculate the complexity respectively. First of all, the complexity of Dijkstra is $O((E+V)\log V)$. The complexity of path scanning is $O(T^2)$. And for the improving methods, the complexity of them are all $O(1)$. Considering $T \leq V$, therefore the total time complexity of our method is $O((E+V)\log V + T^2)$.
- **Space Complexity Analysis**
  The space complexity calculation is similar to time complexity calculation. The space consumption of Dijkstra is $O(V^2)$. The space consumption of path scanning is $O(2E)$. And for the improving methods, the space consumption of them are all $O(1)$. Therefore, the total space complexity of our method is $O(V^2 + 2E)$.

F. *Optimality*

The optimality of heuristic method is hard to define. Because small step operators, such as our improving methods, are tend to fall into local optimality. While big step operators, e.g. path scanning, may miss the global optimality when approaching the global optimality. Therefore, we have to import methods like simulated annealing and genetic algorithm to help us get closer to global optimality.

IV. Experiments

In this section, performance and efficiency of the algorithm will be tested. For the performance part, we test the heuristics

and criteria by applying them respectively to the datasets. For the efficiency test, we test the running time of different path scanning methods.

*1) **Environment**:* The main testing environment is Windows 10 Home Edition with AMD Ryzen 7 5800H 3.20GHz. Python edition 3.8.8., Numpy 1.18.0 and Numba 0.53.1.

*2) **Performance Test**:* This section will show the result of applying heuristics and criteria method to several different datasets of CARP.

TABLE II
PERFORMANCE TEST RESULT OF PATH SCANNING WITH SPECIFIC CRITERIA

	C1	C2	C3	C4	C5	ER
gdb1	473	483	454	392	443	392
gdb10	446	396	402	436	423	335
val1A	290	316	313	310	286	257
val4A	713	743	732	741	728	653
val7A	571	639	570	587	534	477
egl-e1-A	5197	5519	5195	5618	5127	5003
egl-s1-A	7310	6818	6689	7606	7025	6689

TABLE III
PERFORMANCE TEST RESULT OF IMPROVING METHODS

	C1	C2	C3	C4	C5	ER
gdb1	375	369	390	369	340	334
gdb10	317	320	317	323	356	310
val1A	235	240	235	230	227	205
val4A	597	587	627	627	571	568
val7A	473	504	473	453	444	425
egl-e1-A	4630	4454	4666	4370	4560	4205
egl-s1-A	6487	6264	6217	6783	6273	6106

*3) **Efficiency Test**:* In this section, we test the path scanning methods' efficiency in different scale of graph. The results are measured by seconds, indicating the running time of different path scanning methods.

TABLE IV
EFFICIENCY TEST RESULT OF 5 ROUNDS

	Path scanning	Path scanning ER	Path scanning improved
gdb	0.002	0.002	0.002
val	0.004	0.005	0.005
egl	0.007	0.007	0.008

*4) **Summary**:* **From the performance test**, *Path scanning ER* found the minimum cost route better than criteria based path scanning. However, the differences between criteria-based and ER-based aren't significant. The improving methods turn out to be essential according to its impact on reducing cost of the route.

From the efficiency test result, we can see that ER-based path scanning is slightly more time-consuming than criteria-based path scanning. But the difference is ignorable as the scale of the graph increases.

V. CONCLUSION

In this article, I focused on solving CARP by using path scanning based methods. I read some papers, others' solutions and code about CARP and path scanning during this period. Furthermore, I implemented several methods to better solve CARP. At the beginning, I was focused on finding global optimal solution, which turned out to be bad because of falling into local optimal frequently. Then I use random method to help me jump out of local optimal. In addition, I used the idea of **Genetic Algorithm** and **Simulated Annealing**. In the end, I was satisfied with my results.

From this project, I learnt a lot about the optimality of a algorithm and how can we do to make the algorithm closer to global optimal. It's the same as in reality, although we often found ourselves stuck in local optimal and lose something along the way, we can always do something to jump out of it and go towards the final goal.

REFERENCES

[1] Arakaki, Rafael Usberti, Fabio. (2018). An efficiency-based path-scanning heuristic for the capacitated arc routing problem. Computers Operations Research. 103. 10.1016/j.cor.2018.11.018.

[2] S. Wang, Y. Mei and M. Zhang, "An Improved Multi-Objective Genetic Programming Hyper-Heuristic with Archive for Uncertain Capacitated Arc Routing Problem," 2021 IEEE Symposium Series on Computational Intelligence (SSCI), 2021, pp. 1-8, doi: 10.1109/SSCI50451.2021.9660154.

[3] Santos L, Coutinho-Rodrigues J, Current JR. Implementing a multi-vehicle multiroute spatial decision support system for efficient trash collection in Portugal. Transportation Research Part A: Policy and Practice 2008;42(6):922–34

[4] Golden BL, Wong RT. Capacitated arc routing problems. Networks 1981;11: 305–15.

[5] Belenguer JM, Benavent E. A cutting plane algorithm for the capacitated arc routing problem. Computers and Operations Research 2003;30(5):705–28.