CARP Report

Bruce Wan 11612201
School of Computer Science and Engineering
Southern University of Science and Technology
Email: 11612201@mail.sustc.edu.cn

1. Preliminaries

This project is how to solve and optimize the CARP problem within a limited time. In this project, this type of carp problem is a vehicle routing problem with capacity limitations. The purpose of this project is to solve the problem of several arc-based tasks for known needs, to find out how to plan the vehicle route under the condition of meeting the vehicle capacity constraints, so that the service cost is minimum. The CARP problem is proposed by Golden. And the problem has been proven to be an NP problem[1]. For this problem, there are two algorithmic approaches, one is the precise algorithm for small-scale CARP problems, and the other is for large-scale CARP problem heuristics, such as simulated annealing, evolutionary computation, and Tubu-scatter algorithm. In this project, I will use the path-scanning greedy algorithm and genetic algorithm to get the initial solution and optimize the algorithm respectively.

1.1. Software

In this project, I will use python 3.6 on the pycharm IDE to write the code. Using Numpy library, and my code will test performance on a computer with an Intel(R)-core I7-6700HQ CPU.

1.2. Algorithm

There are two main algorithm I used. The first one is path scanning algorithm based on the idea of greedy algorithm, which is used to generate the initial solution population. An other algorithm is genetic algorithm that finds better offs-s pring of the hybrid mutation based on the initial solutions.

2. Methodology

The main core algorithm idea is to use path-scanning to randomly generate 10000+ initial solutions, then use genetic algorithm to select the best initial solution from the initial solution population, and directly cross the task, also called chromosomes of the initial solution and do mutation in them.

2.1. Representation

The variables used in this project are mainly vertex, depot, required_edges, none_required_edges, vehicles, capacity, total_cost, graph, tasks_list.

- **vertex**: the number of vertex in the graph
- **other variables**:
  - **depot**: the initial point where car start to work.
  - **Required_edges**: the number of require edges
  - **None_required_edges**: number of NR-edges
  - **vehicles**: the minimal number of car in mission
  - **capacity**: the capacity of a car
  - **Total_cost**: the total cost of task.
  - **graph**:the matrix to store the graph character
  - **tasks_list**: the list to store NR-edges

- **output result**: the algorithm will output a optimal solution of the routes in this mission.

2.2. Architecture

Here list all functions in this project, they are only concise introduction of the algorithm. In the following contents and I will show you my pseudo-codes to illustrate my thoughts.

- **Functions**:
  - ***read_file***: read the file and get the data
  - ***initial_solution1***: demand/cost and some heuristic adjustment to get the initial_solution.
  - ***Initial_solution2***: randomly path-scanning algorithm
  - ***choose_solution***: generate 10000+ initial solution population and compare them with each other to choose the minimal-cost one.
  - ***gene_algorithm***:select the task, cross the task,and do the mutation
  - ***Create_worker***: establish eight processes
  - ***Finish_worker***: stop the processes after all the execution.
  - ***Get_parameter***:get the parameter from the console
  - ***Process_algorithm***: include all the execution algorithm and put them in eight processes respectively.

The last of the codes, use *main()* to test all the codes.

2.3. Detail of Algorithm

Here I will demonstrate some important algorithm in my project, they are **Floyid algorithm**, **initial_solution**, **choose_solution**, **gene_algorithm**, **process_algorithm**.

Floyid_algorithm: The algorithm mainly used to find the shortest distance between every two points.

Algorithm 1 Floyid algorithm

```
Input :graph[v][v],v
Output: dist[v][v]
1: for k from 1 to v:
2:     for i from 1 to v:
3:         for j from 1 to v:
4:             if dist[i][j]>dist[i][k] + dist[k][j]
5:                 dist[i][j] = dist[i][k] + dist[k][j]
6:             end if
7:     end for
8:  return dist
```

Initial_solution: The algorithm use path scanning to randomly generate the initial utilized solution.

Algorithm2: initial_solution

```
Input : tasks_list, graph, depot, capacity
Output: solution_list
1:  while tasks_list ! = null:
2:      while True:
3:          for task in tasks_list:
4:              find the most minimal-cost task:
5:          for task in tasks:
6:              if task_cost = minimal-cost_task:
7:                  put them in list[ ]
8:          for task in list :
9:              final_min = random.choice(list)
10:         if capacity - final_min_cost < 0:
11:             total_solution_cost += graph[current][depot]
12:             break
13:         capacity -= final_min_cost
14:         total_solution_cost+=final_min_cost + t_cost
15:         for task in tasks_list:
16:                 remove the task from tasks_list
17:         end if
18:     end for
19  end while
20  return initial_solution
```

Choose_solution: the function is used to generate solution population and choose the most minimal cost from the population and return sorted list.

Algorithm 3: Choose_solution

```
Input time, tasks_list, graph, depot, capacity
Output: sort_list
1: for i from 1 to 100000:
2:     result = initial_solution(tasks, graph,depot,capacity)
3:     solution_population.append(result)
4:     if execute_time>time*0.3
5:         break
6: sort_list = sort(solution_population)
7: return sort_list
```

Gene_algorithm:

Algorithm 4:Gene_algorithm

```
Input: initial_solution, total_solution_cost, deport, remain_time, capacity, graph
Output: optimal_solution
1: time_remain_limit = 1
2:while (remain_time > time_remain_limit):
3:     while True:
4:         while True:
5              choose two random number from 1 to
               length(initial_solution)
6:             if  num_1!=num_2:
7                  break
8:             end if
9:         end while
10:            choose two random number from
                   length(initial_solution[num_1]) and
                   length(initial_solution[num_2])
                   respectively
11:    for j from 1 to task_1+1:
           calculate the left total_demand_1
12:    for j from 1 to task_2+1:
           Calculate the left total_demand_2
13:    for j from task_1+1 to length(solution[num_1]):
           Calculate the right total_demand_1
14:    for j from task_2+1 to length(solution[num_2]):
           Calculate the right total_demand_2
```

```
15:        if (the demand satisfies the capacity):
16:            break
17:        cross the tasks_list1 and tasks_list2
18:        if  num.random(1,20)< 2:
               Inverse the task_head and task_tail
19:      calculate the cost of the new_solution
20:      if total_solution_cost > new_solution_cost:
21:          total_solution_cost = new_solution_cost
22:          solution = new_solution
23:      remain_time = time.time() - execute_time
24:  return solution
```

3. Empirical Verification

This project uses the data set provided by the TA, and results are verified on the carp test website. After the test results are correct, the online-judge is used to submit the work. The results are correct.

3.1. Design

About this project, I use path-scanning to get the initial solution, and then randomly generate the initial solution population with 10000 initial solutions. After selecting the best offspring, I use genetic algorithm to cross the best offspring, and mutate. Find the cost value close to the optimal solution within a limited time.

3.2. Data and data structure

Data being used in this project is five .dat files for test the codes. The data structures used in this project include numpy.array, list, tuple, and queue.

3.3. Performance

To get better results, I used eight processes to perform the genetic algorithm separately. In the specified time, the cost of the s1-dat data is 5360, and the data result of e1-dat is 3721. The test results of val1, val4, and val7 are extremely close to or equal to the true value.

In addition, my code uses some test sets in addition to online-judge. In the code test results, one solutions reach the optimal value, and three solutions are very similar to the optimal values. The comparison of the results of some test sets with the optimal solution is as follows

Data set	Average cost	Optimal cost	Average time
egl-s2-A	10995	9825	119.55
egl-e1-B	4600	4498	119.52
gdb-11	399	395	60..33
gdb-16	127	127	59.32
val-8B	408	395	59.42
val-10D	569	530	58.59

3.4. Result

For the test data s1-dat, the cost is 5360. For the test data e1-dat, the value is 3721. For the test data val1-dad-t, the value is 280. For the test data gbd1, the result is 316, for the test data val1, the result is 173. For other not-online judge data set, the algorithm's performance is not bad. According to the results of my algorithm, it is found that for small-scale data sets, most of my solutions are equal to or very close to the optimal solution. For a test set with a fixed number of points equal to 140, my solution is likely to be close to the optimal solution, and the worst will not differ from the optimal solution by more than 10%.

3.5. Analysis

This project made me understand that there are many ways to solve the optimization problem of carp. Although I only use genetic algorithm, the traditional algorithms for optimizing carp problems such as simulated annealing or tabu search and the new algorithms proposed by scholars are A good way to solve this problem. Some algorithms will perform well on large-scale carp problems. For example, Dr. Huang Junwu introduced a clustering algorithm instead of path-scanning to get the initial solution, and then used the new method proposed in the article to solve the carp problem. But on the small-scale carp problem, the greedy algorithm plus the large-scale random generation solution can quickly get a value that is very close to the optimal solution. Different kinds of algorithms have different effects on carp problems of different scales..

Acknowledgments

I am very grateful to Ning Wentao and Tang Dexuan for sharing their genetic algorithms with me. The performance and results of their algorithms are better than me. Communicating with them can get a lot of different excellent ideas in their influence. Next, I improved my genetic algorithm. Although my algorithm is not the best, I have achieved good results on the small-scale carp problem.

References

[1] Golden B L，WongR T．Capacitated arc routing problems[J]．Net—works，1981，11：305—315．