PROJECT: CARP SOLVER

COURSE: ARTIFICAL INTELLIGENCE

NAME ZHANG, SHIQI

ID 11510580

1. Introduction

The capacitated arc routing problem (CARP) constitutes an important class of vehicle routing problems. It has applications in areas such as household refuse collection, street sweeping, inspection of electrical and pipeline networks for faults, mail delivery, etc. Inspired by Memetic Algorithm with Extended Neighborhood Search (MAENS) algorithm, which was published by Tang, Mei and Yao [2]. I used genetic algorithm as framework. In the formulation of first generation, a greedy method is applied for generating given size (usually ten times of the population scale) solutions and the top K solutions will be selected as the first generation based on the total cost of each solution. During the parent selection part, roulette model is used for weight-based random selection. In the cross over part, both OX [1] and SBX [2] are applied for generating new offspring. For chromosome mutation, I used three traditional local search methods: single-insertion, double insertion and swap [3] and one novel MS operator [2]. There exists the possibility of capacity violation in local search part. Due to the high potential of these infeasible candidates, I use a penalty parameter to evaluate the overall cost of each solution no matter feasible or not, which can allow the admirable genes in the infeasible chromosomes to be preserved. Meanwhile, in the generation filter part, a threshold guarantees that the number of infeasible chromosomes is fixed in each generation and avoid the overwhelming ratio of infeasible chromosomes. All the detail algorithms will be shown in part three.

2. Problem Definition

CARP can be described 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 $v \in V$. 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:

a) Each route must start and end at v_0;

b) The total demand serviced on each route must not exceed Q;

c) Each task must be served exactly once (but the corresponding edge can be traversed more than once)

3. Algorithm Overview

The total procedure of my solution is shown as follows. The detail of each algorithm is in part four.

```
Input: undirected graph, capacity, generation size, task
Output: optimal routes, minimum cost

procedure CARPSOLVER (S)
    repeat
        generate Dijkstra table

        // initiate first generation
        repeat
        chromosome := random-based greedy search(task)
        until ten times of generation size
        filter best chromosome in generation size

        // add children
        for i := 1 to generation size do
            p1,p2 := roulette-based parent selection
            random select following two cross over method
            child := OX operator(p1,p2)
            child := SBX operator(p1,p2)
            // mutation
            if random number less than mutate rate
                single-insertion(child)
                double-insertion(child)
                swap(child)
                MS-operator(child)
                find the best child
            endif
        endfor

        // shrink with violation toleration and infeasible solution threshold
        next generation := shrink(parent generation, current generation)

    until timeout
    return best solution
end procedure
```

Algorithm.1 Overall Algorithm

4. Algorithm Detail

4.1 Dijkstra Algorithm

Before the implementation of finding the route plan, certain graphical algorithm should be applied for searching the minimum cost between every to node. Generating this minimum cost table is the first priority preliminary. Here, I choose Dijkstra Algorithm as the shortest path search algorithm.

4.2 Random Based Greedy Search Algorithm

Generally, people use random selected genetic sequences as first generation for genetic algorithm. Here I applied a greedy way to tighten the initial search area. It may occur that the solution is trapped in the local optimum, however, the operations in crossover and mutation parts can help to find more possible potential solutions. Here, the greedy-based first generation not only aims at reducing the redundant searching but also acts as the baseline of the final solution.

Compared with traditional path scanning [4] way to guide the next step. We here append a random tiny float to break tie while facing selection. The random-based guidance, therefore, can generate bunches of solutions in the same cost. In the process of experiments, it shows better performance than the traditional path scanning method.

4.3 Ulusoy's Splitting Algorithm

Ulusoy's splitting algorithm [4] works on a given giant tour S. It minimizes total cost and, as a secondary objective, the number of vehicles. It runs in $O(\tau)$ space only, by avoiding an explicit generation of the auxiliary graph H. Two labels are used for each node i of H: V i (cost of the shortest path from 0 to i in H) and N i (number of arcs on that path, i.e. number of trips). Due to two indexes i and j, the algorithm enumerates all subsequences of S that correspond to feasible trips and computes their loads and costs. At the end, the total cost F(S) and the minimum number of vehicles K for that cost can be read in V and N τ. The pseudo-code is shown as follows:

```
procedure split(S)
    V(0),N(0) := 0
    for i := 1 to τ do V(i) := ∞ endfor
    for i := 1 to τ do
        load, cost := 0; j := i
        repeat
            load := load + q(S(j))
            if i = j then
                cost := D(σ,S(i)) + w(S(i)) + D(S(i),σ)
            else
                cost := cost - D(S(j-1),σ) + D(S(j-1),S(j)) + w(S(j)) + D(S(j),σ)
            endif
            if (load ¡= W) and (cost ¡= L) then
                VNew := V(i-1) + cost
                if (VNew ¡ V(j)) or ((VNew = V(j)) and (N(i-1) + 1 ¡ N(j)) then
                    V(j) := VNew
                    N(j) := N(i-1) + 1
                endif
                j := j + 1
            endif
        until (j ¿ τ) or (load ¿ W) or (cost ¿ L)
    endfor
end procedure
```

Algorithm.2 Ulusoy's Splitting Algorithm

4.4 Order Crossover(OX)

Given two parents P1 and P2 with τ tasks, In OX, the sequence for C1 is followed by P1(p),...,P1(q) P2(q+1),...,P2(τ), P2(1),...,P2(p−1), with restriction that tasks from P2 are taken only if missing in C1. While applying OX operator, we need to flatten the routes list into one single route. After crossover, based on the current order, Ulusoy's splitting algorithm can help to divide the task sequence into proper allocation with minimum cost.

4.5 Sequence Based Crossover(SBX)

Given two parent solutions S1 and S2, SBX randomly selects two routes R1 and R2 from them, respectively. Then, both R 1 and R 2 are further randomly split into two sub routes, say R1 = (R11,R12) and R2 = (R21,R22). After that, a new route is obtained by replacing R12 with R22. Finally, it is possible that some tasks appear more than once in the new route, or some tasks in R 1 are no longer served in the new route. In the former case, the duplicated tasks are removed from the new route. In the latter case, the missing tasks are re-inserted into the new route.

4.6 Local Search Based Mutation:

Compared with traditional mutation strategy which only change one genetic point. I applied local search methods with weaker search strength. It means the current solution point will only random select one neighbor rather than traversal all its neighbors. After doing three traditional operator and MS operator, we select the best from these four candidates.

The reason why we skip traversal is that the essential of genetic algorithm is to create as many solutions as possible. If we care too much about the quality of neighbors, the iteration time will be slow down. The detail of simplified local search methods is shown in 4.7 and 4.8.

4.7 Traditional Local Search Operator

1) Single Insertion: In the single insertion move, a task is removed from its current position and randomly re-inserted into another position of the current solution or a new empty route.

2) Double Insertion: The double insertion move is similar to the single-insertion except that two consecutive tasks are moved instead of a single task.

3) Swap: In the swap move, two candidate tasks are selected and their positions are exchanged.

4.8 Merge and Split(MS) Operator

Traditional small scale local search is easy to be trapped into the local optimum. MS operator can be regarded as a large step search method, while jumping into a brand-new area, re-do the small scale local search. The process of MS operator can be divided into three parts. Firstly, merge k routes (practically k=2) into one single route. Secondly, order this route with minimum cost. Traditionally path scanning is applied here, however, considering the high performance of random-based greedy search in first generation initiation, we use random-based algorithm to order the task sequence. Finally, we need to split the sequence by Ulusoy's splitting algorithm and then connect all the routes together. The pseudo-code is shown as follows:

```
procedure MS (Chromosome)
    route1 := random select one route from Chromosome
    route2 := random select one route from Chromosome
    total := merge route1 and route2 together
    for given times:
        random-based greedy search(total)
    endfor
    best := select the best one
    new routes := split(best)
    append together
end procedure
```

Algorithm.3 Merge and Split Algorithm

4.9 Violation Toleration

During the cross over and mutation parts, there exists the risk of capacity violation. It means that the capacity is over the standard capacity for a car in certain route. This solution can have a more lower cost but it is infeasible. Considering the high potential of these infeasible solutions, we can keep some of them for guiding the searching direction. Meanwhile, a penalty function need to be applied on the total cost, which can help to assess both feasible solutions and infeasible solutions.

$$\text{total cost} = \text{real cost} + p * \text{over capacity}$$

In this formula, real cost means the current real cost. P is the penalty parameter which will increase by iterations and it means the toleration of the infeasible solutions. Over capacity is the current real capacity minus the standard capacity.

5. Experiments

The operating system is Win10, Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz 3.41GHz, 8GB. The multi-processing number is eight. Generation size is 200. And use random seed.

Every dataset has been tested for 10 times with the limit time 300 seconds and all the experimental environments are same. The result of seven different datasets is shown as follows:

data file	Best	worst	average
gdb1	316	316	316
gdb10	275	275	275
val1A	173	173	173
val4A	400	400	400
val7A	279	279	279
egl-e1-A	3548	3618	3614
egl-s1-A	5135	5348	5237

Table 1. The Detail of Result

6. References

[1] Oliver, I. M., Smith, D. J., & Holland, J. R. C. (1987). A Study of Permutation Crossover Operators on the Traveling Salesman Problem. *International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*.

[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] B. L. Golden. (1983). Computational experiments with algorithms for a class of routing problems. *Computers & Operations Research, 10*(1), 47-59.

[4] Ulusoy, G. (1985). The fleet size and mix problem for capacitated arc routing. *European Journal of Operational Research, 22*(3), 329-337.