Project 2: Solving the Multi-Depot Vehicle-Routing Problem using Genetic Algorithms

Per Magnus Veierland
permve@stud.ntnu.no

February 20, 2017

CHROMOSOME REPRESENTATION

The chromosome is represented as a list of fixed length equal to the total number of depots t. Each entry in the list consists of a list of variable length of at least the number of maximum vehicles m, with each entry representing a route assigned to the associated depot. Each of the routes is a variable length list consisting of the integer indexes of the customers visited on the route, depots not included. These route lists can be empty. Flattening out this nested list structure will always yield a list with a length equal to the total number of customers n.

CREATION

Creating the initial chromosome population is done in a two-step process consisting of a) *grouping* the set of customers such that each customer is assigned to its nearest depot, b) *routing* and *scheduling*, where for each depot customers are assigned and sequenced within routes.

Assuming that there are two depots d_A and d_B, and a customer c_i is to be assigned, the grouping works by assigning the customer to the closest depot as determined by the Euclidean distance.

If $\text{DISTANCE}(c_i, d_A) < \text{DISTANCE}(c_i, d_B)$, then assign c_i to d_A. If $\text{DISTANCE}(c_i, d_A) > \text{DISTANCE}(c_i, d_B)$, then assign c_i to d_B. Otherwise, if $\text{DISTANCE}(c_i, d_A) = \text{DISTANCE}(c_i, d_B)$, then assign c_i to a depot randomly.

$$\text{DISTANCE}(a, b) = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2} \tag{1}$$

After grouping all customers to a depot, the *Clarke and Wright Savings* algorithm [1] is used to assign customers to individual routes. The algorithm begins with assuming that a separate route is used for each customer, and that the cost of each route is $2 \cdot \text{DISTANCE}(c_i, d)$. For each pair of customers c_i, c_j assigned to a depot, the saving of combining their two routes is computed as $\text{DISTANCE}(c_i, d) + \text{DISTANCE}(c_j, d) - \text{DISTANCE}(c_i, c_j)$, as the cost saved is one of the distances between the depot and each of the customers, and the added cost is the distance between the two customers.

After computing a list of the savings for combining each pair of customers within a depot, this list is sorted in descending order such that the list begins with the greatest saving. For each entry in the savings list, the following evaluation is made:

a) If c_i and c_j have neither been assigned a route and they can be added to a route within the duration and load constraints of their depot, a new route consisting of c_i and c_j is created.

b) If exactly one of c_i and c_j has been assigned to a route, and it is adjacent to a depot within the route, the other customer is added into the route between the customer and the depot, if this modification does not violate the given duration and load constraints for their depot.

c) If both c_i and c_j have been added to two different routes and are both adjacent to the depot in their routes, the two routes are merged into a single route with c_i and c_j in the middle, if the resulting route does not violate the duration and load constraints for their depot.

Any remaining customers are assigned individual routes. The sequencing of the resulting routes are such that they work well as an input to the genetic algorithm, and no separate sequencing algorithm was found necessary.

Algorithm 1 BEST COST ROUTE CROSSOVER

```
1: function BEST-COST-ROUTE-CROSSOVER(p1, p2)
2:     let c1 ← COPY(p1), c2 ← COPY(p2)
3:     let depot_index ← RANDOMCHOICE(|Depots|)
4:     let r1 ← RANDOMCHOICE(c1[depot_index])
5:     let r2 ← RANDOMCHOICE(c2[depot_index])
6:     for c in r1 do
7:         remove c from each route in c2
8:     end for
9:     repeat lines 6-8 replacing r2 for r1 and c1 for c2
10:    for c in r1 do
11:        let insertion_points ← []
12:        for depot_routes in c2 do
13:            for route in depot_routes do
14:                let route_cost ← ROUTECOST(route)
15:                for insert_position in RANGE(LEN(route) + 1) do
16:                    let modified ← COPY(route)
17:                    insert c at position insert_position into modified
18:                    if WITHINLIMITS(modified) then
19:                        let modified_cost ← ROUTECOST(modified)
20:                        let insertion_cost ← modified_cost − route_cost
21:                        append (insertion_cost, route, insert_position)
22:                    end if
23:                end for
24:            end for
25:        end for
26:        sort insertion_points in ascending order by insertion_cost
27:        let (insertion_cost, route, insert_position) ← insertion_points[0]
28:        insert c1 at position insert_position into route
29:    end for
30:    repeat lines 10-29 replacing r2 for r1 and c1 for c2
31:    return c1, c2
32: end function
```

CROSSOVER OPERATOR

Best-Cost-Route-Crossover as developed in [2] was selected as the genetic operator for crossover to combine the chromosomes of individuals for inheritance, see Algorithm 1. When applying it, a random route is selected from the same depot for both parents. The customers belonging to each of the selected routes are removed from the other parent. Further, the removed customers of each parent are reinserted into the their respective feasible location with the lowest insertion cost. Insertion cost is given by the cost of a route after inserting a customer, minus the cost of a route before inserting a customer.

In 10% of cases on a per customer basis, a random feasible insertion point is chosen instead of the best possible insertion point.

Mutation operator

Inspired by the crossover operator, a *Best-Cost-Customer-Mutation* genetic operator was developed to mutate individuals as a source of genetic diversity. In it, a random customer is selected and removed from the chromosome. In 90% of cases, the customer is reinserted at the feasible location with the lowest insertion cost in the chromosome. In the other 10% of cases, the customer is reinserted at a random feasible insertion point.

Selection mechanism

The chosen parent selection mechanism is tournament selection, where to select a parent from the population a group of individuals are randomly chosen from the population. In ϵ of cases, a random individual is selected from the group as a parent, and in $1-\epsilon$ of cases the individual in the group with the best fitness is selected as a parent.

Parameters

Using a grid search, the best parameters was found to be a population size of 200, a generation number of 100, a crossover rate of 0.8, a mutation rate of 0.2, a tournament group size of 10, and a tournament randomness factor ϵ of 0.1,

Feasibility

The creation process may produce infeasible solutions, as they may exceed the maximum number of allowed vehicles. The *Best-Cost-Route-Crossover* genetic operator only produces feasible offspring as only route insertions are made which respect the duration and load constraints provided and it cannot create routes. The *Best-Cost-Customer-Mutation* genetic operator only produces valid modifications, as any route insertion made must respect the duration and load constraints provided and it cannot create routes. In any case where the crossover or mutation operator fails to insert a customer to create a feasible route, the operators will return their input chromosomes.

Infeasible solutions are handled by penalizing them heavily in the fitness function. Thus they are allowed to exist in the population and contribute to diversity, while unlikely being the best candidate.

Fitness function

The primary objective of the assignment is to minimize the total distance travelled by all vehicles. To reflect this, the fitness of an individual i is given by the sum of the distances of its routes $r \in routes_i$. To simplify the notation it is assumed that both depot endpoints are contained in r, although this is not how the chromosome encodes it.

Infeasible solutions are penalized by a factor of p^{E_i}, where p is a constant set to 1.5 and E_i is the number of vehicles used in the solution i exceeding the maximum number allowed for each depot.

$$E_i = \sum_{d \in depots_i} \max\left(0, |routes_{di}| - MaxVehicles\right) \tag{2}$$

$$\text{Fitness}(i) = p^{E_i} \cdot \sum_{r \in routes_i} \sum_{j=1}^{|r|-1} \text{Distance}(r_j, r_{j+1}) \tag{3}$$

As the genetic algorithm code is implemented to only support the maximizing of a fitness function, the reciprocal of Equation 3 is used to transform maximizing into minimizing.

Elitism

A low degree (0.01%) of elitism is employed to ensure that the best solutions are not lost from the population during evolution. This is handled by sorting the population by fitness and always adding the top individuals to the next generation before generating offspring to fill the rest of the next generation.

In addition to preserving the top solutions, elitism helps pressure the evolution process by sustaining a portion of top solutions within the population.

Solution

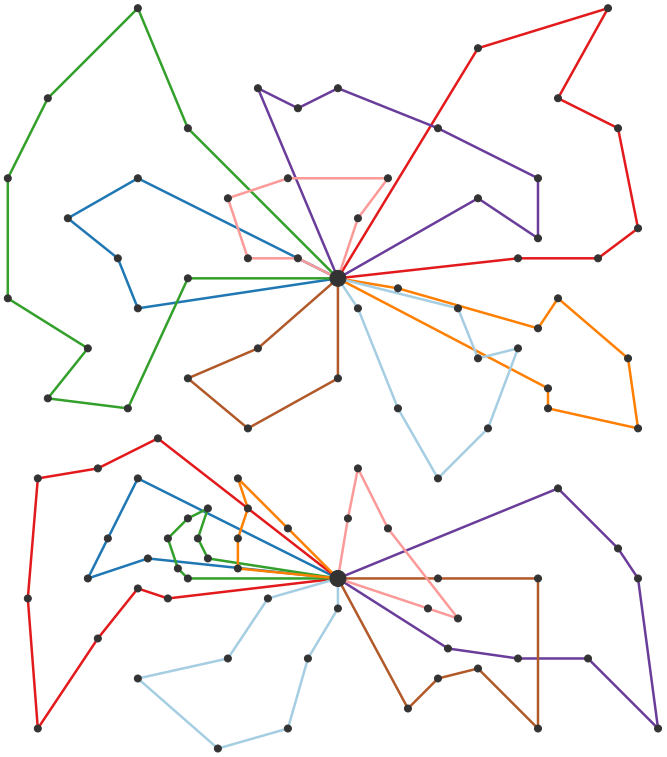

Figure 1: Visualization of problem p04.

```
1076.56
1   1    58.97    99    0 85 16 61 5 0
1   2    98.17    99    0 60 84 17 86 38 44 91 100 0
1   3    88.00    96    0 54 55 25 67 39 56 74 0
1   4    43.19    90    0 37 98 93 99 96 59 92 0
1   5    66.62   100    0 21 4 23 75 22 41 0
1   6    33.52    85    0 13 6 94 95 97 0
1   7    41.94    75    0 58 53 40 72 73 0
1   8    57.73    80    0 87 42 14 43 15 57 2 0
2   1    62.43    96    0 19 47 48 82 0
2   2    98.84    96    0 33 78 34 35 71 65 66 0
2   3    80.85    97    0 63 90 32 20 9 81 51 0
2   4   121.10    98    0 7 83 45 8 46 36 49 64 11 0
2   5    46.35    52    0 52 18 89 27 0
2   6    75.47   100    0 68 80 24 29 79 3 1 0
2   7    43.45    97    0 31 88 62 10 30 70 0
2   8    59.93    98    0 69 28 26 12 77 76 50 0
```

Figure 2: Solution to problem p04 (106.8% of best known cost).

References

[1] Geoff Clarke and John W Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.

[2] Beatrice Ombuki, Brian J Ross, and Franklin Hanshar. Multi-objective genetic algorithms for vehicle routing problem with time windows. *Applied Intelligence*, 24(1):17–30, 2006.