2006 Multi-Objective VRPTW Genetic Algorithm

COSC 4F90

Brett Terpstra

Department of Computer Science
Brock University

October 19, 2023

VRPTW Problem

- Vehicle Routing problem (VRP) is a specialization of the travelling salesman problem, making it NP complete.
  - This means there is (currently) no deterministic algorithm which can solve the problem in a reasonable (polynomial) amount of time.
- In VRPTW - VRP with time windows - we must also respect the opening and closing constraints of the customers we are servicing.
- The 2006 VRPTW paper, and this implementation, use a genetic algorithm in an attempt to find a 'good enough' solution to the VRPTW problem in a reasonable amount of time.

VRPTW Problem with Pareto Ranking

- In essence, we are modelling evolution using computers to solve for a favourable solution using math. The advancement of this paper is the introduction of Pareto ranking.
  - Pareto ranking means we treat the problem as a multi-objective optimization problem instead of a single objective problem.
  - A solution is considered to be an improvement over another if it is as good or better in every objective that is part of the problem.
  - In this case, our objectives are reducing the total distance and total number of vehicles.

Chromosome and Route Representation

- Chromosome are just arrays.
- Routes store the calculated best (valid) route from the chromosome, representing one vehicle.

```
struct chromosome {
    std::array<customerID_t, CUSTOMER_COUNT>
genes{};
};
struct route {
    ArrayList<customerID_t> customers;
    distance_t total_distance = 0;
};
```

Individual and Population Representation

- Chromosomes are stored alongside all valid calculated routes within structures representing an individual in the population.
- the population stores only the set of individuals.

```
struct individual {
    chromosome c;
    ArrayList<route> routes{};
    distance_t total_routes_distance = 0;
    rank_t rank = 0;
    fitness_t fitness = 0;
};
struct population {
    ArrayList<individual> pops;
};
```

Genetic Algorithm Pseudo-code

```
read_instance_data(path);
init_population();
for (int i = 0; i < GENERATION_COUNT; i++){
    reconstruct_populations();

    calculatePopulationFitness();
    rankPopulation();

    population p;
    applyElitism(p, 1);
    while (p.size() < POPULATION_SIZE)
        applyTournamentAndOrCrossover(p);

    applyMutation(p);

    rebuild_population_chromosomes(p);
    current_population = p;
}
```

Tournament Selection

- The primary cause of my weird initial results.
- `tournamentSelect()` pseudo-code:

```
ArrayList<customerID_t> buffer;
fill_with_unique_individuals(buffer);
if (select(0,1) <= 0.8)
    return best_in(buffer);
else
    return random_from(buffer);
```

- `applyTournamentAndOrCrossover` pseudo-code:

```
auto p1 = tournamentSelect();
auto p2 = tournamentSelect();
if (select(0,1) <= 0.8)
    insert(applyCrossover(p1, p2));
else
    insert(p1, p2);
```

Best Cost Route Crossover

- applyCrossover(p1, p2) pseudo-code:

```
    auto r1 = select_rand_route(p1.routes);
    auto r2 = select_rand_route(p2.routes, r1);
    auto c1, c2 = copy_to_children(p1, p2);
// step a) remove p1's route from p2's (now c2)
// remove p2's route from p1's (now c1)
    remove_from(c2, r1);
    remove_from(c1, r2);
// insert r1 back into c2 at best and feasible
   point.
    insert_to(c2, r1);
// insert r2 back into c1 at best and feasible
   point.
    insert_to(c1, r2);
```

Inserting back into a individual

- `insert_to(child, route)` pseudo-code:

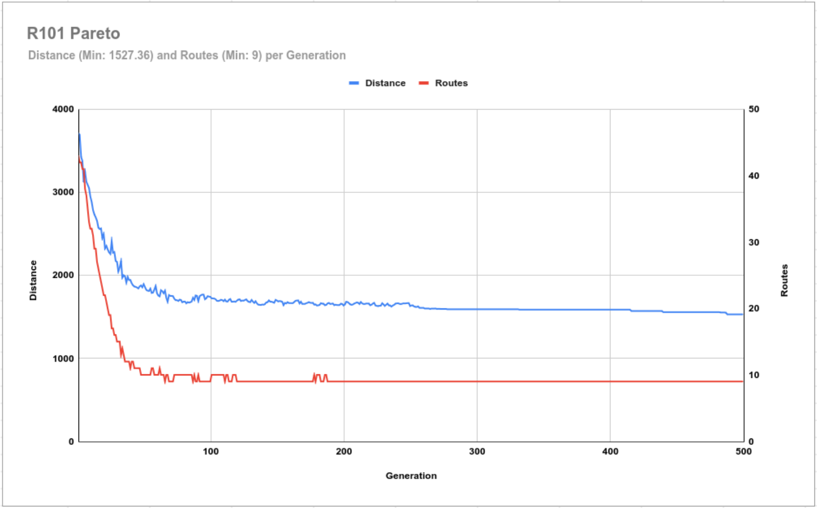
```
for (customer in route.customers){
    ArrayList<route> possibleRoutes;
    for (child_customer in child.routes.customers){
        auto r = insert_before_customer(customer, child_customer);
        if (feasible(r))
            possibleRoutes.push_back(r);
    }
    // no feasible route found, we must make a new one
    if (possibleRoutes.empty())
        return new route(customer);
    else {
        route min;
        min.distance = double::max();
        for (route in possibleRoutes)
            if (route.distance < min.distance)
                min = route;
        // the actual implentation uses route_cache
        // to track the insertion location
        child.routes.insert_with_place(min);
    }
}
```

Weighted Sum vs Pareto Ranking

- In general, Pareto ranking performed as good to significantly better than weighted sum fitness.
- After fixing the index issue in tournament selection the R101 results got significantly better with Pareto ranking.
  - The results are significantly better than the Solomon best, which is suspicious...
- More on this later.

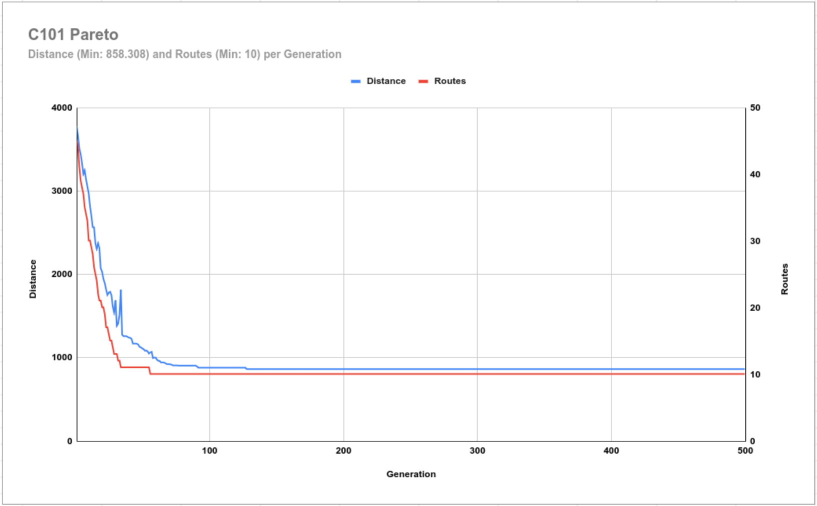
R101 Pareto Results

Solomon Best: Distance 1637.7, Routes 20



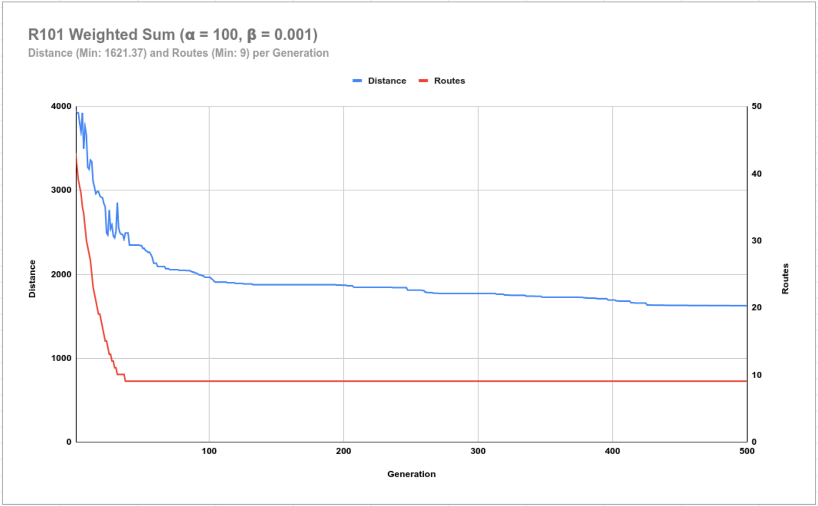
C101 Pareto Results

Solomon Best: Distance 827.3, Routes 10



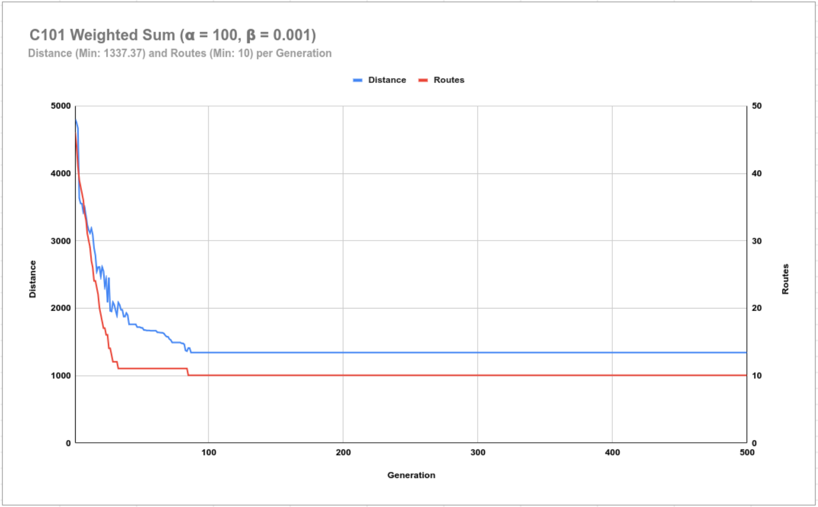
R101 Weighted Sum Results

Solomon Best: Distance 1637.7, Routes 20



C101 Weighted Sum Results

Solomon Best: Distance 827.3, Routes 10

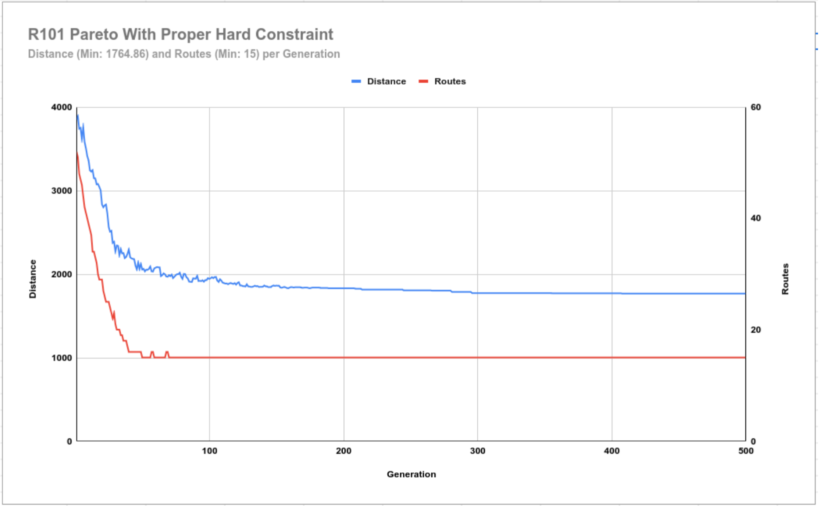


Addressing the R101 Results

- Because the c101 set contains values with a start time of 15 and an end time of 67 but service time of 90.00, it creates a situation where we cannot check the hard constraint of unloading before closing. All tests performed today are made by ignoring this issue.
- "`lastDepartTime + record.service_time > record.due`"
- If an exception is made to allow routes in c101 to exist as single customer routes, the results become non-competitive with the Solomon best.
- Next slide contains a graph of r101 including the constraint above.

R101 Pareto Ranking

Solomon Best: Distance 1637.7, Routes 20



References

- Ombuki, Beatrice & Ross, Brian & Hanshar, Franklin. (2006). Multi-Objective Genetic Algorithms for Vehicle Routing Problem with Time Windows. Applied Intelligence. 24. 17-30. 10.1007/s10489-006-6926-z.