Nature Inspired Optimisation for Delivery Problems
Chapter 2: Heuristics for the Vehicle Routing Problem

Neil Urquhart

May 27, 2022

These slides are designed to accompany the book "Nature Inspired Optimisation for Delivery Problems : From Theory to the Real World".

`https://link.springer.com/book/10.1007/978-3-030-98108-2`

The Vehicle Routing Problem

VRP: Definition

- The TSP requires the construction of one to route to solve it.
- The Vehicle Routing Problem (VRP) requires a network of routes to solve it.
- A set of customers require visits which are undertaken using a fleet of vehicles
- There are two fundamental aspects to the VRP:
  - The allocation of customers to vehicles/routes
  - The ordering of visits within each route

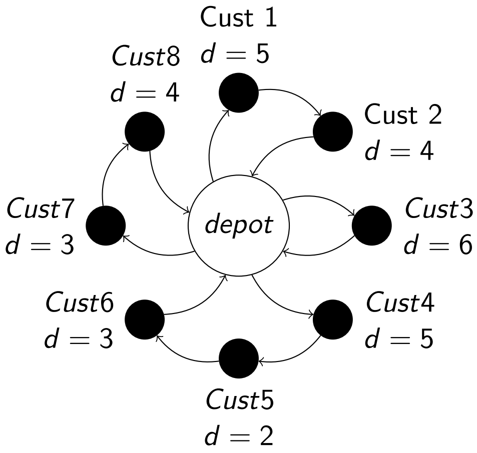
VRP: Characteristics

There are many variants of the VRP, some of their characteristics include:

- A vehicle capacity constraint, limiting the number of customers a vehicle can service.
- A demand associated with a customer, a vehicle can only service customers where their collective demand does not exceed the vehicles' capacity.
- Time windows that restrict visits to a customer to a specific time period
- A mixture of vehicles having different characteristics such as capacity, operating cost, speed etc

An Example CVRP Solution

In this example there are 8 customers with a total demand (d) of 31 and a vehicle capacity (cap) of 10. A solution based on 4 vehicles is shown.



Solving the CVRP

An Example CVRP

Let us consider the following scenario which represents a typical real-world problem:

> *ACE Doughnuts bake doughnuts at their factory. Customers order doughnuts, placing their order the previous day. The company has a fleet of vans, each of which can carry up to capacity doughnuts. Each day the company has to work out a plan that allocates deliveries to vans. The plan must ensure that:*
>
> - *No van is carrying more than capacity doughnuts*
> - *Each customer receives the number of doughnuts that they ordered*
> - *The number of vans used is minimised*
> - *The total distance travelled collectively by the vans is minimised*

The Grand Tour

- We can consider the CVRP to be a TSP, that is divided into separate routes.
- If we treat the 8 customer example shown earlier as a TSP, with the depot s as the start/finish point and the customers as cities then we might produce the following TSP solution:

$$d \rightarrow c1 \rightarrow c2 \rightarrow c3 \rightarrow c4 \rightarrow c5 \rightarrow c6 \rightarrow c7 \rightarrow c8 \rightarrow d$$

- This tour that takes in all of the customers is sometimes known as the *Grand Tour*.
- We can divide the grand tour into feasible sub tours, with respect to the vehicle capacity constraint

Dividing The Grand Tour

- We can divide the grand tour into feasible sub tours by commencing with an empty tour:

$$d \rightarrow d$$

- Customers can be added until the combined demand exceeds the capacity of the vehicle.
- In this case Customer 1 is added, then Customer 2 (giving a total demand of 9) Customer 3 cannot be added as the combined demand of 15 exceeds the capacity (10) of the vehicle.

$$d \rightarrow c1 \rightarrow c2 \rightarrow d$$

Dividing The Grand Tour ... cont

A second tour commencing with Customer 3 is created, adding Customer 4 would exceed the capacity constraint so tour 2 only visits one customer:

$$\begin{gathered} d \rightarrow c1 \rightarrow c2 \rightarrow d \\ d \rightarrow c3 \rightarrow d \end{gathered}$$

This process continues until all of the customers have been added to a tour:

$$\begin{gathered} d \rightarrow c1 \rightarrow c2 \rightarrow d \\ d \rightarrow c3 \rightarrow d \\ d \rightarrow c4 \rightarrow c5 \rightarrow c6 \rightarrow c7 \rightarrow d \\ d \rightarrow c8 \rightarrow d \end{gathered}$$

Splitting up a grand tour in this manner creates a quick solution, but the quality is dependent on the underlying TSP.

The Grand Tour Algorithm

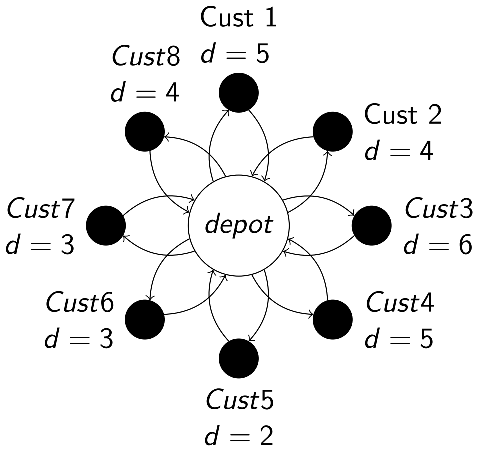
```
begin
currentTour = []
currentDemand = 0
for $visit\ in\ grandTour$ do
    if $(currentDemand + visit.demand) > CAPACITY$ then
        result.add(currentTour) currentTour = []
         currentDemand = 0
    end
    currentTour.append(visit) currentDemand =+ visit.demand
end
return result
```

The Clarke-Wright Savings Algorithm

- Clarke and Wright [Clarke & Wright 1964] proposed an algorithm for solving the CVRP based on the savings made by combining tours.
- The algorithm is an iterative algorithm which commences from a starting point based on each customer having their own tour.
- The solution is improved by finding pairs of customers that can be placed next to each other within a tour, resulting in fewer tours and less overall distance travelled.

The Clarke-Wright Savings Algorithm: Starting Point

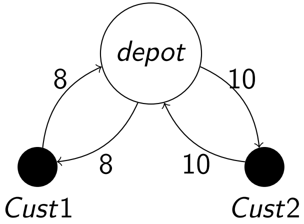
The initial solution places each customer in their own tour.



From this point, the algorithm searches for savings that can be made by combining pairs of customers into the same route.

The Clarke-Wright Savings Algorithm: Calculating savings...

Each possible pair of customers is considered and the potential saving of having them in the same tour is calculated. Firstly the distance *d* between them via the depot is calculated:



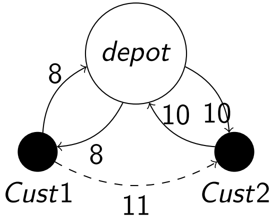
$$d = distance(c_1, d) + distance(d, c_2)$$

$$d = 8 + 10$$

$$d = 18$$

The Clarke-Wright Savings Algorithm: Calculating savings...

The direct distance $'d$ is calculated:



$$d' = distance(c_1, c_2)$$
$$d' = 11$$

Finally the saving s is calculated as follows:

$$s = d - d'$$
$$s = 7$$

As long as s is greater than 0 then there is a saving to be made (we will check the vehicle capacity constraint at a later stage).

The Clarke-Wright Savings Algorithm

The algorithm calculate each possible this ensures that the savings that have the biggest benefits are considered first which maximises their chances of being successfully applied to the solution.
If we consider a saving that places c_x next to c_y, we need to establish that the following conditions are met before the saving may be applied:

- c_x is at the end of a route
- c_y is at the start of a route
- The combined demand of the routes that incorporate c_x and c_y is less than $vcap$.

The Clarke-Wright Savings Algorithm

```
**Procedure** ClarkeWright()
savings = []
solution = buildOnePerCust()
**for** $visitv_x inproblem$ **do**
    **for** $visitv_y inproblem$ **do**
        d = distance($v_x$, $depot$) + distance($depot$, $v_y$)
        d'= distance($v_x$, $v_y$)
        s = d-d'
        **if** $s > 0$ **then**
            savings.add(new(saving($s$, $v_x$, $v_y$))
        **end**
    **end**
**end**
```

Cont ...

The Clarke-Wright Savings Algorithm

```
Cont ...
sort(savings)
for $savingsinsavings$ do
    $r_x$ = route(s.x) $r_y$ = route(s.y)
end
if $r_x.last == s.x \&\& r_y.first == s.y$ then
    if $d(r_x) + d(r_y) <= cveh$ then
        merge($r_x, r_y$)
    end
end
Return solution
```

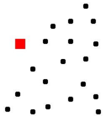
Summary

Summary

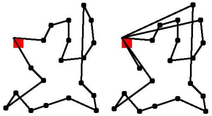
- The Grand Tour algorithm is simple and relatively quick, it does not produce a particularly good result.
- The Grand Tour takes a TSP solution as its input, that solution has been optimised only on distance.
- When the grand tour is split into individual routes, customers can only be adjacent to those that they were adjacent to in the TSP solution or adjacent to a depot.
- The Clarke-Wright Algorithm provides a means of constructing a solution based on improvements
- Clarke-Wright commences with a feasible, but inefficient solution, and iteratively improves it.

Summary: Examples

The P-n20-K2 problem instance.



The grand tour (left) and sub tours created by splitting (right)



The same problem solved using the Clarke-Wright algorithm.