6. ROUTING PROBLEMS

6.1. VEHICLE ROUTING PROBLEMS

Vehicle Routing Problem, VRP:
Customers i=1,...,n with demands of a product must be served using a fleet of vehicles for the deliveries. The vehicles, with given maximum capacities, are situated at a central depot (or several depots) to which they must return. Determine a routing schedule that minimizes the total cost of the deliveries.

The TSP is a special case of VRP, which means VRP is NP-hard.

Graph formulation for a single depot problem:
Input: Graph G=(V,E), with costs (or distances) c_{ij} on the edges and demands d_i on the vertices, |V|=n, m vehicles with capacities K_j, j=1,...,m. A central depot is at vertex 0.
Problem: Determine cycles R_1, ... , R_m for the vehicles that start from vertex 0 and service all vertices such that the load of vehicle j doesn't exceed its capacity and the total cost of the cycles is minimized.

A route means a sequence of customers for each vehicle.

Variations and modifications:
- Service demand for each edge instead of vertices → multiple postmen problem.
- Number of vehicles may be a decision variable, instead of a fixed value. Initially there is an unlimited number of vehicles available. Number of vehicles or total cost to be minimzed.
- The graph may be directed or undirected.
- Pick-up problem: goods collected from the customers. Combination: customers may have either demand (import) or take away (export) needs.
- Multiple products, specialized vehicles for some products.
- Due dates / deadlines for deliveries.
- Travel time / cost of a vehicle may be constrained.

Applications:
- distribution plan for a wholesale dealer
- garbage disposal
- mail delivery, mailbox collection
- security company's rounds
- elevator maintenance
- school bus routing
- airline schedules
- snow plows

Also variable costs may be included in the cost function. These cost are proportional to the amount of load. These probelms belong to logistics and transportation, not considered here.

EXACT METHODS

Because the problem is "harder" than the TSP, exact methods are suitable for small instances only (number of customers < 100). Versions of Branch and Bound, Branch and Cut and Dynamic Programming algorithm have been tried. The Integer Programming formulation can be based on
- partition of routes, or
- construction of routes for vehicles (cf. TSP models), or
- product flow.

HEURISTICS

Many heuristics use similar principles as the TSP heuristics, keeping an eye on the capacity constraints. Most methods can be classified into the following categories:
• Constructive methods: tours are built up by adding nodes to partial tours or combining subtours, regarding capacities and costs.
• Two-phase methods: consist of 1) clustering of vertices and 2) route construction. The order of these phases may be "clustering first, route later" or "route first, clustering later".

We introduce two methods: the Clarke and Wright method is an example of the first, the Sweeping heuristic of the second category.

Assume an complete, undirected graph with symmetric costs and an unlimited number of identical vehicles with capacity K.

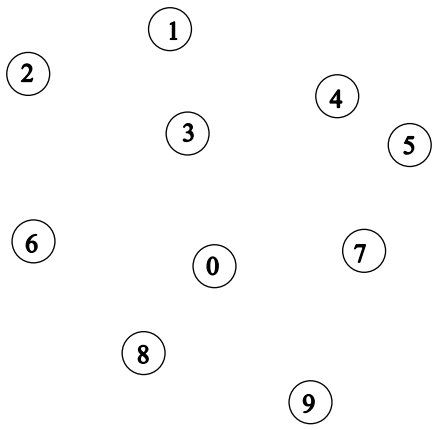
METHOD OF CLARKE AND WRIGHT

1. Starting solution: each of the n vehicles serves one customer.
2. For all pairs of nodes i,j, $i \neq j$, calculate the *savings* for joining the cycles using egde [i,j]:
   $s_{ij} = c_{0i} + c_{0j} - c_{ij}$.
3. Sort the savings in decreasing order.
4. Take edge [i,j] from the top of the savings list. Join two separate cycles with edge [i,j], if
   (i) the nodes belong to separate cycles
   (ii) the maximum capacity of the vehicle is not exceeded
   (iii) i and j are first or last customer on their cycles.
5. Repeat 4 until the savings list is handled or the capacities don't allow more merging.

The cycles if i and j are NOT united in sep 4, if the nodes belong to the same cycle OR the capacity is exceeded OR either node is an interior node of the cycle.

Improvement: Optimize the tour of each vehicle with a TSP heuristic

Example 6.1. Given a complete graph with a central depot 0 and 9 customers. (The costs are not directly proportional to the euclidean distances of the diagram.)



Symmetric costs:

c_{ij}	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	-	12	11	7	10	10	9	8	6	12
1	12	-	8	5	9	12	14	16	17	22
2	11	8	-	9	15	17	8	18	14	22
3	7	5	9	-	7	9	11	12	12	17
4	10	9	15	7	-	3	17	7	15	18
5	10	12	17	9	3	-	18	6	15	15
6	9	14	8	11	17	18	-	16	8	16
7	8	16	18	12	7	6	16	-	11	11
8	6	17	14	12	15	15	8	11	-	10
9	12	22	22	17	18	15	16	11	10	-

Demands:

i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
$\mathbf{d_i}$	10	15	18	17	3	5	9	4	6

Capacity of a vehicle: K = 40

Solution with Clarke & Wright:

Savings $s_{ij} = c_{0i} + c_{0j} - c_{ij}$
Symmetry: $s_{ij} = s_{ji}$

s_{ij}	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1		15	14	13	10	7	4	1	2
2			9	6	4	12	1	3	1
3				10	8	5	3	1	2
4					17	2	11	1	4
5						1	12	1	7
6							1	7	5
7								3	9
8									8

Ordered savings: [4,5], [1,2], [1,3], [1,4], [2,6], [5,7], [4,7], [1,5], [3,4], [2,3], [7,9], [3,5], [8,9], [1,6], [5,9], [6,8], [2,4], ...

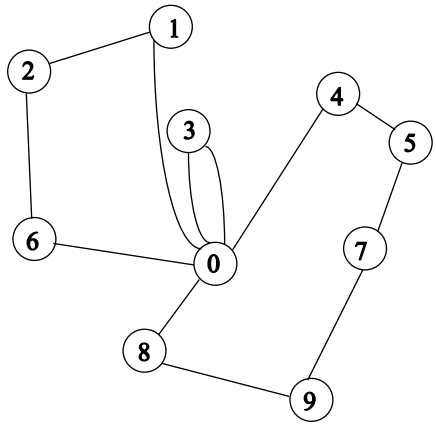
Initial solution: cycles 0-1-0, 0-2-0, ... , 0-9-0.

Edge [4,5]: Join cycles 0-4-0 and 0-5-0: result **0-4-5-0**, load $d_4 + d_5 = 20 < K$.
Edge [1,2]: Join 0-1-0 and 0-2-0: result **0-1-2-0**, load $d_1 + d_2 = 25 < K$.
Edge [1,3]: Capacity limit: $d_1+d_2+d_3 = 43 > K$.
Edge [1,4]: Capacity limit: $d_1+d_2+d_4 = 42 > K$.
Edge [2,6]: Join cycles 0-1-2-0 and 0-6-0: result **0-1-2-6-0**, load $d_1+d_2+d_6 = 30 < K$.
Edge [5,7]: Join cycles 0-4-5-0 and 0-7-0: result **0-4-5-7-0**, load $d_4+d_5+d_7 = 29 < K$.
Edge [4,7]: Condition 4(i) doesn't hold: nodes belong to same cycle.
Edge [1,5]: Condition 4(iii) doesn't hold: node 5 is an interior node of its route.
Edge [3,4]: Capacity limit: $d_3+d_4+d_5+d_7 = 47 > K$.
Edge [2,3]: Condition 4(iii) doesn't hold: node 2 is an interior node.
Edge [7,9]: Join cycles 0-4-5-7-0 and 0-9-0: result **0-4-5-7-9-0**,
load $d_4+d_5+d_7+d_9 = 35 < K$.
Edge [3,5]: Condition 4(iii) doesn't hold: node 5 is an interior node.
Edge [8,9]: Join cycles 0-4-5-7-9-0 and 0-8-0: result **0-4-5-7-9-8-0**,
load $d_4+d_5+d_7+d_9+d_8 = 39 < K$.
Edge [1,6]: Condition 4(i) doesn't hold: nodes belong to same cycle.
Edge [5,9]: Condition 4(i) doesn't hold: nodes belong to same cycle.
Edge [6,8]: Capacity limit: $(d_1+d_2+d_6)+(d_4+d_5+d_7+d_9+d_8) = 69 > K$.

Result: three routes that cannot be united because of capacity limit. Solution:

Route:	**Load:**	**Cost:**
0-3-0	18	14
0-1-2-6-0	30	37
0-4-5-7-9-8-0	39	46

Total cost f = 97



SWEEP HEURISTIC

This heuristic is of the type "clustering first, route later". Assume the customers are points in a plane with euclidean distances as costs. The distance between (x_i, y_i) and (x_j,y_j) is

$$c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

1. Compute the polar coordinates of each customer with respect to the depot. Sort the customers by increasing polar angle.
2. Add loads to the first vehicle from the top of the list as long as the capacity allows. Continue with the next vehicle until all customers are included. Now the customers have been clustered by vehicles.
3. For each vehicle, optimize its route by a suitable TSP method.

Graphically, in step 1 we rotate a ray centered at the depot.The starting angle can be chosen arbitrarily.

Source: http://neo.lcc.uma.es/radi-aeb/WebVRP/

Comment: Also for the TSP, clustering techniques can be used for splitting large scale TSP instances to smaller parts.

6.2. THE POSTMAN PROBLEM

In a weighted graph, find a minimum weight closed walk that includes every edge at least once.

A graph is an *Eulerian graph* if it has an Eulerian cycle (= Eulerian circuit).
Graph G has an Eulerian cycle if and only if every vertex has an even degree. An Eulerian cycle, when it exists, is an optimum postman's tour, because every edge is traversed exactly once.

An Eulerian cycle can be found with the following method:

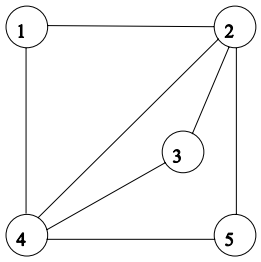
EULERIAN CYCLE

1. Choose a starting node $i:=i_0$ with a degree ≥ 2.
2. If an untraversed edge [i,j] incident to i exists, go from i to j (edge [i,j] is added to the path). Set i:=j and repeat step 2.
If all edges from i are traversed, $i=i_0$ and a cycle is formed.
3. Remove edges of the previous cycle. If untraversed edges remain, go to 1.
4. Connect the cycles at shared nodes. The resulting walk is an Eulerian cycle.

For instance, a depth-first search can be used when choosing the next vertex. Different choices may produce different solutions because there are more than one Eulerian cycles in a almost all graphs.

Example 6.2. Find an Eulerian cycle in the following graph.

An Eulerian cycle exists, because every vertex has an even degree.



Start with node 1.
First cycle: 1-2-3-4-1
Choose starting node for the next cycle: 2
Second cycle: 2-4-5-2

Shared nodes 2 and 4. Choose connection at node 2.
Insert the second cycle between 2-3:

1-2-4-5-2-3-4-1 is an Eulerian cycle.

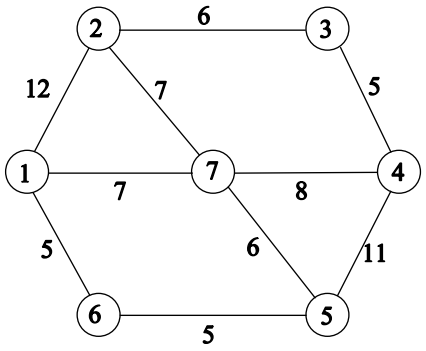
Assume a connected, undirected graph G=(V,E) with nonnegative weights c_{ij} on the edges. The following algorithm uses as a subroutine a shortest path algorithm and a minimum weight perfect matching algorithm. Both of these, and the Eulerian cycle algorithm, are polynomial and the resulting postman's algorithm is also polynomial.

POSTMAN ALGORITHM

1. Identify odd-degree vertices, denote this set by V_O. If none exists, go to 5.
2. Find a shortest path between all pairs of odd degree vertices $i,j \in V_O$
3. Form a complete graph K for V_O with weights w_{ij} = length of the shortest path determined in step 2.
4. Find a minimum weigth perfect matching in K. Duplicate the implied edges of the matching (the shortest paths) in G. This augmented multigraph, call it G_E, is an Eulerian graph.
5. Form an Eulerian cycle in G_E. This is the optimum postman's tour.

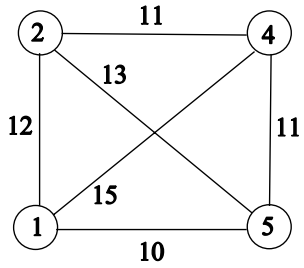
In the augmented graph all edges have an even degree and an Eulerian cycle can be formed. This algorithm finds a minimum length duplication of edges that makes an Eulerian cycle possible.

Example 6.3. Find a postman's route in the following graph G.



Vertices of odd degree: $V_O = \{1,2,4,5\}$.

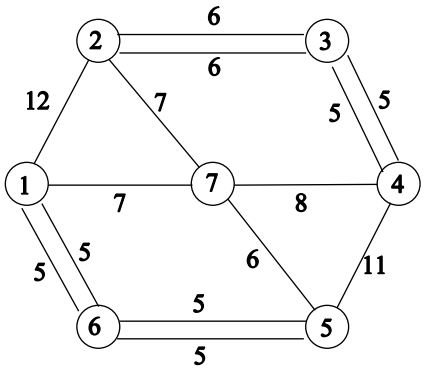
Lengths of shortest paths shown in graph K.
Minimum weight perfect matching:
{[1,5], [2,4]}



Complete graph K

Add edges of the shortest paths 1-6-5 and 2-3-4 to G.

Augmented graph G_E:



Construct an Eulerian cycle in G_E e.g. with following choices:
First cycle: 1-2-3-4-3-2-7-4-5-6-1
Second cycle: 1-6-5-7-1

Connect the cycles at node 1 to form an Eulerian cycle: 1-2-3-4-3-2-7-4-5-6-1-6-5-7-1.
This is the optimum postman's tour, with cost 93. There are alternative optimum tours: in all of these the edges 1-6, 6-5, 2-3, 3-4 are are traversed twice and the cost is 93.

The method can be modified for directed graphs. Unfortunately, there is no polynomial time exact algorithm for mixed graphs, because that problem is NP-hard.