Transportation Logistics

Lecture held by Gilbert Laporte
Winter Term 2004 (University of Vienna)

Summarized by Guenther Fuellerer

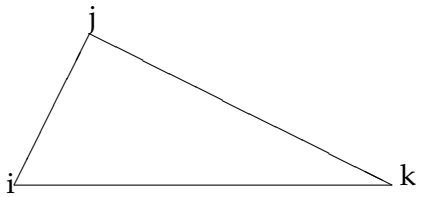
Contents

List of Figures

1 Travelling Sales Person Problem (TSP)

input: distance matrix (c_{ij})
output: Hamiltonian Tour (no repetition)
A graph consists of vertices, that can be connected by arcs (directed graph, which means that $c_{ij} \neq c_{ji}$ and that the distance matrix is asymmetric) or edges (undirected, which means that $c_{ij} = c_{ji}$ and that the distance matrix is symmetric). City problems tend to be asymmetric (due to one ways) and Inter-City problems tend to be symmetric.

The Triangle Inequality is satisfied if $c_{ik} \leq c_{ij} + c_{jk}$



The Euclidian Distance between two vertices i (x_i, y_i) and j (x_j, y_j) is given by:
$c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

1.1 Applications of the TSP

1. Design of a TSP Tour
2. Vehicle Routing Problem (each route is a TSP)
3. Manufacturing
    a) Drilling Problems
    b) Placement Problems
    c) Transition Costs between Jobs

4. Crystallography (each face hast to be x-rayed and the optimal sequence has to be determined)

5. Examination Scheduling

A	B	D	F	G
C	E	J	H	I

Given 10 exams the objective is to minimize the number of students who have two exams in a row. C_{ij} represents the number of students having exams i and j, therefore the objective is the creation of a chain which minimizes the sum of c_{ij} between successive exams.

1.2 Different Solving Strategies

1.2.1 Enumeration

Given a problem with n vertices there are $(n-1)!$ combinations in the **asymmetric case**: $|1|n-1|n-2|\ldots|1|$
In the symmetric case for instance the sequence 1-5-2-4-3 is equal to 3-4-2-5-1 and therefore the number of different combinations is reduced to $\frac{(n-1)!}{2}$

1.2.2 Heuristics

c_ij	1	2	3	4	5	6
1	0	4	1	3	6	9
2	8	0	5	7	3	1
3	7	6	0	2	4	4
4	6	2	1	0	6	1
5	9	8	2	5	0	4
6	4	1	3	3	5	0

Nearest Neighbor Heuristic

1-3-4-6-2-5-1 gives costs of 17
5-3-4-6-2-1-5 gives costs of 20
The Nearest Neighbor Heuristic is quick and is 10% - 15% away from the optimum.
The solution can be evaluated by comparing upper $(\overline{z})$ and lower bound $(\underline{z})$, where $\underline{z} \leq z* \leq \overline{z}$

A lower bound in this example can be obtained by first reducing rows and then columns:

c_ij	1	2	3	4	5	6	
1	0	4	1	3	6	9	1
2	8	0	5	7	3	1	1
3	7	6	0	2	4	4	2
4	6	2	1	0	6	1	1
5	9	8	2	5	0	4	2
6	4	1	3	3	5	0	1
	3	0	0	0	2	0	13 = sum of reduction constants

$\frac{\bar{z}}{\underline{z}} = 1.31 \Rightarrow \frac{\bar{z}}{z*} \leq 1.31$

Insertion Methods

- Start with a particular tour.
- Gradually insert other vertices between two vertices of the current particular tour.
- At each iteration the cheapest insertion is performed

1-3-4-1 cost=9

try to insert vertex 2:

between	gives	added cost
1-3	1-2-3	8
3-4	3-2-4	11
4-1	4-2-1	4

try to insert vertex 5:

between	gives	added cost
1-3	1-5-3	7
3-4	3-5-4	7
4-1	4-5-1	9

try to insert vertex 6:

between	gives	added cost
1-3	1-6-3	11
3-4	3-6-4	5
4-1	4-6-1	**-1** $\rightarrow$ cheapest insertion

Flood (1956)
In a planar problem the vertices on the *convex hull* are visited *in the same order* as in an optimal solution. Therefore it is a good idea to start an insertion method with the convex hull. The Nearest Neighbor heuristic and insertion heuristics are **construction heuristics**. Apart from them there are **improvement heuristics**:

r-opt Heuristics

- Remove r edges from the tour.
- Reconnect the tour. If an improvement is realized a new tour has been constructed.

The most common values for r are 2 (see figure 1.1) and 3 (see figure 1.2). The Two-opt brings you within around 5% of the optimum solution.
Croes (1958)

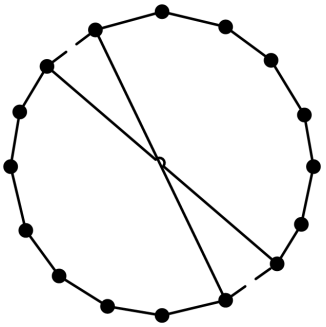


Figure 1.1: Two-opt

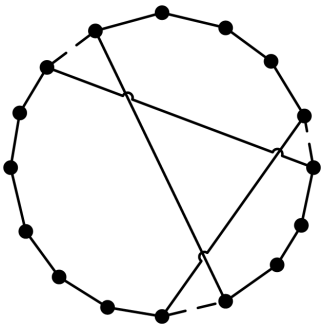


Figure 1.2: Three-opt

The optimal tour in a planar TSP does not cross itself. Therefore applying 2-opt eliminates all crossings.

Or-opt

Named after Ilhan Or (1976) or-opt (see figure 1.3) is the attempt to

- relocate chains of 3 consecutive vertices in all possible positions.
- Implement any profitable move.
- Repeat as long as profitable.
- Start with strings of 2 vertices and finally with single vertices.

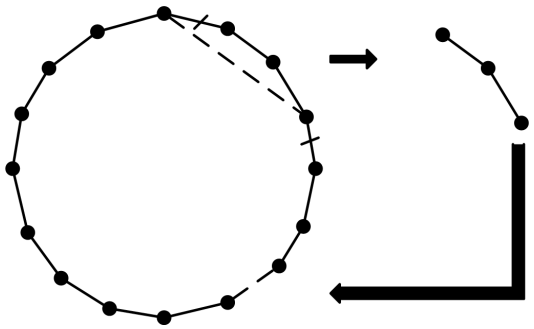


Figure 1.3: Or-opt

Checking all Or-opt moves takes $O(n^2)$ time. In practice Or-opt is not as good as two-opt BUT two-opt followed by Or-opt gives quite good results.

1.2.3 Exact Algorithms

Asymmetric Problems

A preliminary result to this problem is the Assignment Problem (AP)
How to assign jobs to people given the following training costs (c_{ij} costs of assigning person i to job j) to minimize the overall costs:

c_{ij}	1	2	3	4	5
1	4	2	5	6	1
2	4	3	2	1	8
3	3	5	6	7	5
4	2	1	1	9	9
5	6	3	7	4	3

$$x_{ij} = \begin{cases} 1 & \text{if person i is assigned to job j} \\ 0 & \text{otherwise} \end{cases}$$

$$min \sum_{i,j} c_{ij} x_{ij}$$

$$\sum_j x_{ij} = 1 \qquad \forall i \quad \text{each job is assigned to one person}$$

$$\sum_i x_{ij} = 1 \qquad \forall j \quad \text{each person is assigned to one job}$$

The relationship between the AP and the *directed* TSP is the following: The binary variable x_{ij} is equal to one if j is the successor of i in the tour:

x_{ij}	1	2	3	4	5	6
1				1		
2	1					
3						1
4					1	
5			1			
6		1				

However some AP solutions are not TSP solutions:

- In an AP x_{ii} can be 1, which is impossible in the TSP $\rightarrow c_{ii} = \infty$
- This tableau is also a valid AP solution:

x_{ij}	1	2	3	4	5	6
1					1	
2						1
3		1				
4	1					
5				1		
6			1			

that gives the following TSP solution:
$1 \rightarrow 5 \rightarrow 4 \rightarrow 1$ and $2 \rightarrow 6 \rightarrow 3 \rightarrow 2$, which is not valid.

TSP Formulation: Asymmetric Case

$$min \sum_i \sum_j c_{ij} x_{ij}$$

$$\sum_j x_{ij} = 1 \qquad \forall i \quad \text{n constraints}$$

$$\sum_i x_{ij} = 1 \qquad \forall j \quad \text{n constraints}$$

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1 \qquad 2^n - 2 \quad \text{Subtour Elimination Constraints}$$

Explanation:
Let $S \subset \{1, 2, 3, \ldots, n\}$ and $S \subset V, S \neq 0$
example: Suppose $S = \{1, 4, 5\}$
$x_{14} + x_{41} + x_{15} + x_{51} + x_{45} + x_{54}$
$0 + 1 + 1 + 0 + 0 + 1 \leq 3 - 1$ false

Methodology

Solve a relaxed problem (= original problem without the subtour elimination constraints) and obtain a lower bound $\underline{z}$
2 possible outcomes

1. Unique tour (no subtour) which is feasible and $z* = \underline{z}$
2. There are subtours $\rightarrow$ add the appropriate constraints and reoptimize

Example:
$V = \{1, 2, \ldots, 6\}$

$min \sum_i \sum_j c_{ij}$
$x_{12} + x_{13} + \cdots + x_{16} = 1$
$x_{21} + x_{23} + \cdots + x_{26} = 1$
$\vdots$
$x_{61} + x_{62} + \cdots + x_{65} = 1$

$x_{21} + x_{31} + \cdots + x_{61} = 1$
$x_{12} + x_{32} + \cdots + x_{62} = 1$
$\vdots$
$x_{15} + x_{25} + \cdots + x_{56} = 1$
We obtain:
$1 \rightarrow 5 \rightarrow 4 \rightarrow 1$
$2 \rightarrow 6 \rightarrow 3 \rightarrow 2$
To avoid these subtours we have to add two constraints:
$x_{15} + x_{54} + x_{41} \leq 2$
$x_{26} + x_{63} + x_{32} \leq 2$
After reoptimization new subtours may arise and new subtour elimination constraints have to be added.

Carpeneto and Toth (1980)

Use the same idea but they do not solve an integer problem. They use branch and bound to eliminate subtours:

1. Set all $c_{ii} = \infty$
2. Solve AP to get a lower bound
3. If there are no subtours: finished with an optimal TSP solution
4. If there are subtours: Find an upper bound $\overline{z}$(feasible solution) by using a patching algorithm (Karp) see figure 1.4
5. Take one subtour and eliminate it by branching see figure 1.5 (for example number I) In subproblem $\underline{A}$ set $c_{34} = \infty$ (and implicitly exclude A)
   In subproblem $\underline{B}$ set $c_{46} = \infty$, etc.
   To force certain arcs remove line i and column j from the cost matrix.

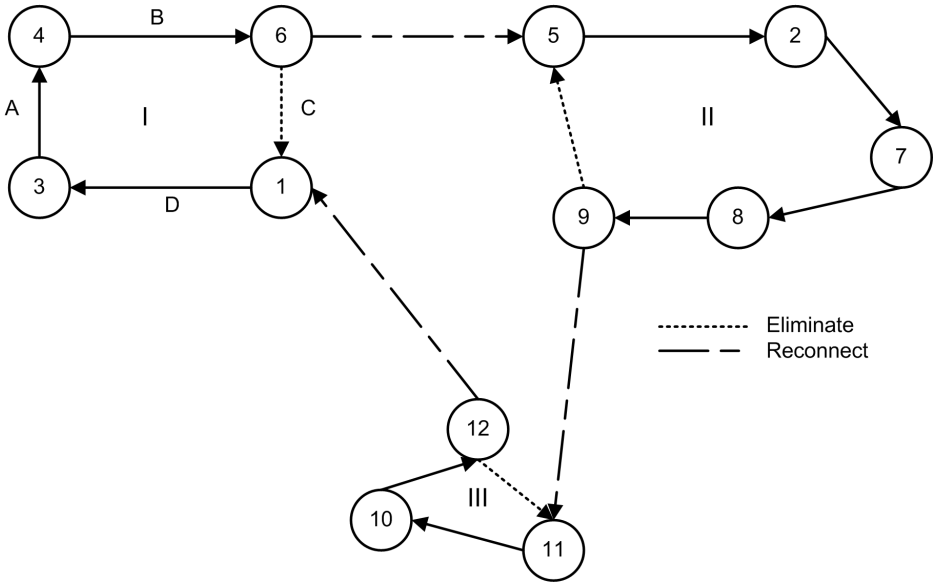


Figure 1.4: Patching Algorithm

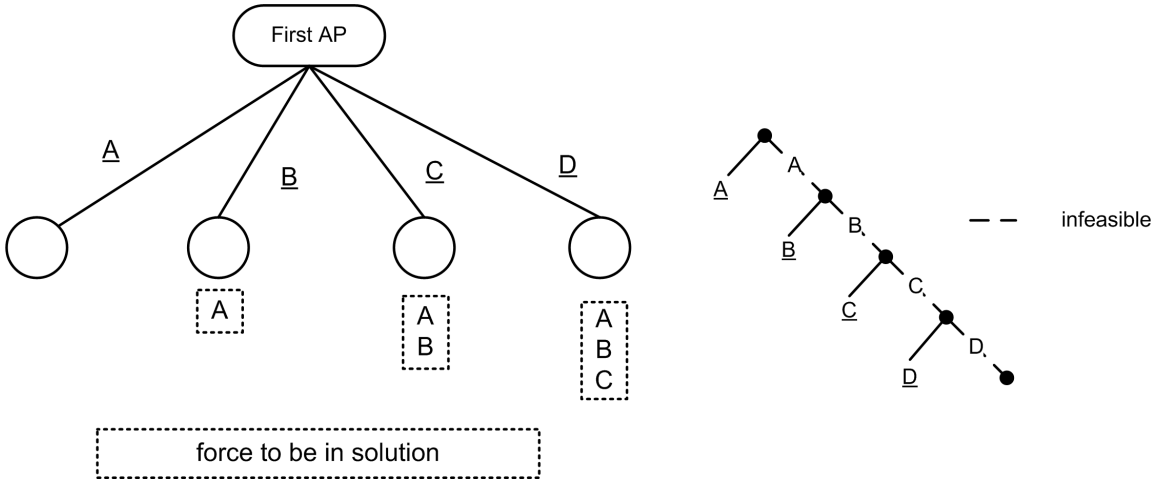


Figure 1.5: Branching I

6. Solve the AP corresponding to each subtour. Kill any branch whose $\underline{z} \geq \overline{z}$ see figure 1.6

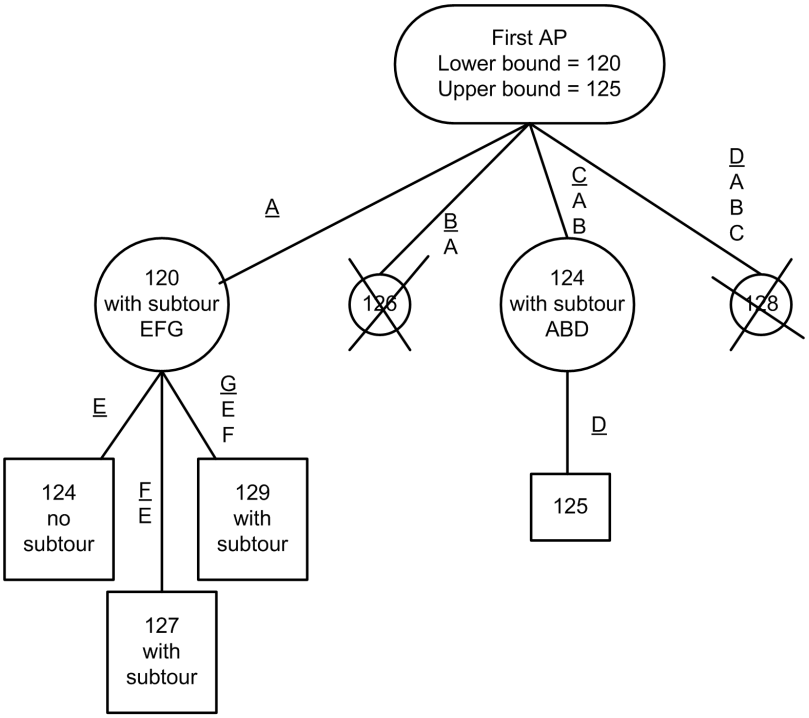


Figure 1.6: Branching II

Two more details:

- Branch on the subtour with the smallest number of arcs not already imposed in previous branches (this reduces work)
- Algorithm works well for asymmetric problems and not so well on symmetric problems because the AP constructs only subtours with two vertices (e.g. 1-2-1) and it needs lot of time to eliminate.

Finally the Carpenetho and Toth algorithm can solve large asymmetric problems easily (n=5000).

Symmetric Problems

In this case we have to use integer linear programming (ILP) to solve the problem. This method is based on the work of Dantzig, Fulkerson and Johnson (1954) and has been improved by many researchers (for instance Miliotis 1977,1978).
The variables x_{ij} are only defined for $i < j$ (due to the fact that $c_{ij} = c_{ji}$)

TSP Formulation: Symmetric Case

$$x_{ij} = \begin{cases} 1 & \text{if edge ij is in the solution j} \\ 0 & \text{otherwise} \end{cases}$$

$$min \sum_{i<j} c_{ij} x_{ij}$$

degree constraints

$$\sum_{i<k} x_{ik} + \sum_{j>k} x_{kj} = 2 \qquad \forall k$$

Subtour Elimination Constraints Variant I

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1 \qquad \forall S \subset V \wedge S \neq \{\} \wedge |S| \geq 3$$

Subtour Elimination Constraints Variant II 'Conectivity Constraints'

$$\sum_{i \in S, j \in \overline{S} \text{ or } i \in \overline{S}, j \in S} x_{ij} \geq 2 \qquad \forall S \subset V \wedge S \neq \{\} \wedge |S| \geq 3$$

Given a TSP route of 1-3-6-5-2-4-1 gives the following degree constraints:

$$\begin{aligned}
k = 1 \quad & x_{12} + x_{13} + x_{14} + x_{15} + x_{16} = 2 \\
k = 2 \quad & x_{12} + x_{23} + x_{24} + x_{25} + x_{26} = 2 \\
k = 3 \quad & \underbrace{x_{13} + x_{23}}_{i<k} + \underbrace{x_{34} + x_{35} + x_{36}}_{j>k} = 2 \\
k = 4 \quad & x_{14} + x_{24} + x_{34} + x_{45} + x_{46} = 2 \\
k = 5 \quad & x_{15} + x_{25} + x_{35} + x_{45} + x_{56} = 2 \\
k = 6 \quad & x_{16} + x_{26} + x_{36} + x_{46} + x_{56} = 2
\end{aligned}$$

Two Subtours: 1-3-6-1, 2-4-5-2	
Variant I	Variant II
$x_{13} + x_{16} + x_{36} \leq 2$	$S = \{1,3,6\} \quad \overline{S} = \{2,4,5\}$
$x_{24} + x_{25} + x_{45} \leq 2$	$x_{12} + x_{14} + x_{15} + x_{23} + x_{34} + x_{35} + x_{26} + x_{46} + x_{56} \geq 2$

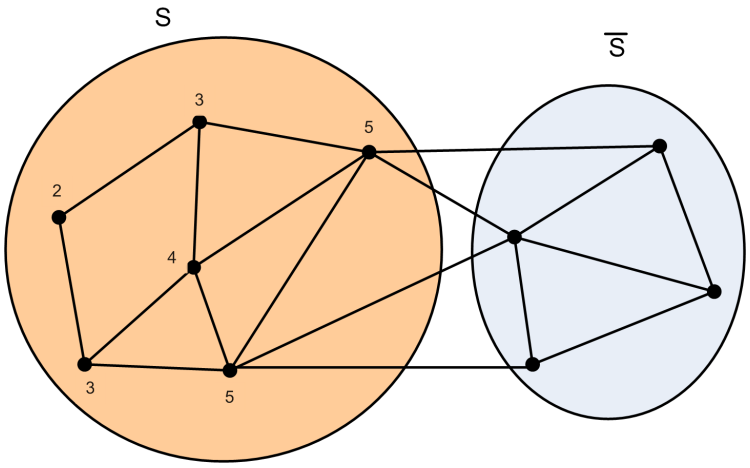


Figure 1.7: Connectivity Constraints

$$I_{(S)} = \sum_{i,j \in S} x_{ij} = \text{number of edges within S}$$

$$E_{(S)} = \sum_{i \in S, j \in \overline{S} \text{ or } i \in \overline{S}, j \in S} x_{ij} = \text{number of edges with one end in S and one in } \overline{S}$$

$$D_{(S)} = \text{total degree of all vertices in S}$$

This gives:

$$2I_{(S)} + E_{(S)} = D_{(S)}$$

$$2 \cdot 9 + 4 = 22 \qquad \text{see figure 1.7}$$

In the TSP each vertex has a degree of 2 and therefore

$$D(S) = 2|S|$$

derivation of the subtour elimination constraints:

$$I_{(S)} \leq |S| - 1 \qquad | \cdot 2$$

$$2 \cdot I_{(S)} \leq 2 \cdot |S| - 2$$

$$D_{(S)} - E_{(S)} \leq 2 \cdot |S| - 2$$

$$D_{(S)} - E_{(S)} \leq D_{(S)} - 2$$

$$E_{(S)} \geq 2$$

Solution of the TSP

$min \sum_{i<j} c_{ij} x_{ij}$

1. degree constraints
2. subtour elimination constraints OR connectivity constraints
3. $0 \leq x_{ij} \leq 1$
4. x_{ij} is integer

Constraint 3 and 4 are called 'integrality constraints'. Relaxing constraints 2 and 4, leaves constraints 1 and 3 and therefore 4 cases are possible:

1. Solution is integer and has no subtours, which is the optimal solution
2. Solution is integer but contains subtours. Therefore impose appropriate subtour elimination constraints and start reoptimization (see figure 1.8).
3. Solution is non integer (not fully integer) but has many 'connected components' (subtours or fractional subtours, see figure 1.9)

4. Solution is non integer and contains only one connected component (see figure 1.10).

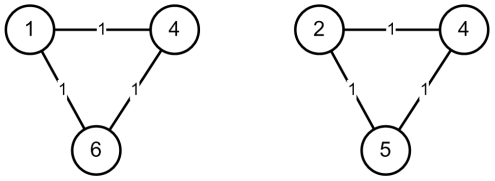


Figure 1.8: Case 2 TSP Relaxation

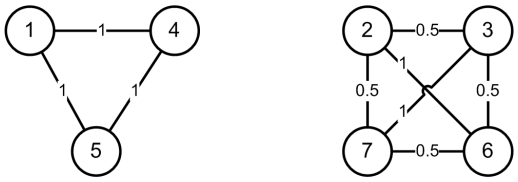


Figure 1.9: Case 3 TSP Relaxation

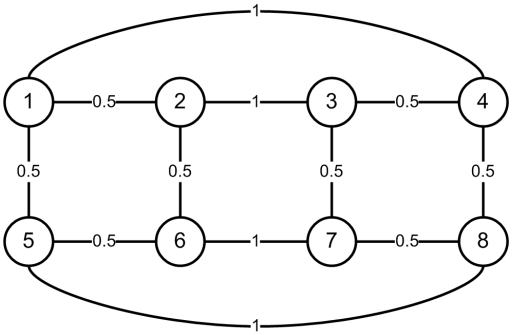


Figure 1.10: Case 4 TSP Relaxation

There are two variants of branching to solve case 3 and 4:

I Straight Algorithm
Branch on a fractional variable and eventually all subproblems will be in case:
1 → feasible solution (not necessarily the optimum due to branching, which is explained in figure 1.11)
2 → Impose appropriate subtour elimination constraints and reoptimize

II Reverse Algorithm
One tries to impose subtour elimination constraints even if the solution is non

integer, which is easy in case 3. In case 4 this is not always possible. In the example of case 3 one could impose $x_{14} + x_{15} + x_{45} \leq 2$ or $x_{23} + x_{26} + x_{27} + x_{36} + x_{37} + x_{67} \leq 3$

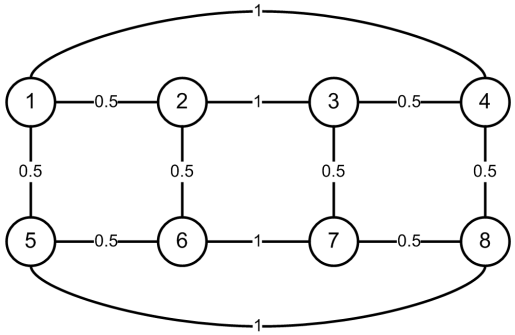


Figure 1.11: Feasible but necessarily not optimal solution

Miliotis concludes that the *reverse algorithm is better* because less branching is needed. These methods and its descendants can solve instances with 500 vertices easily. Larger sizes are rarer and more difficult.

2 Vehicle Routing Problem (VRP)

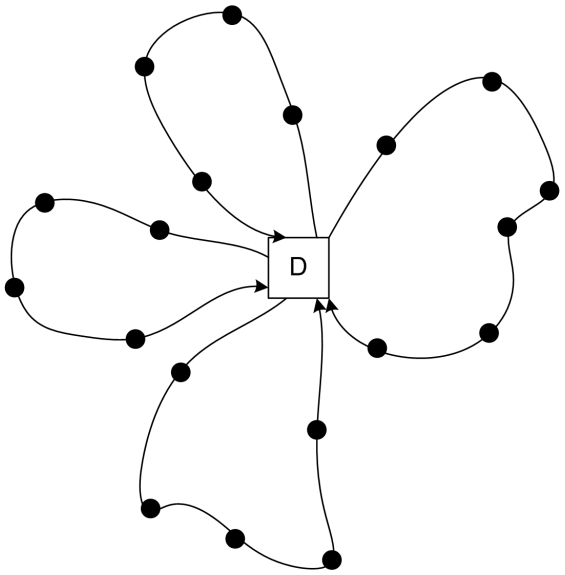


Figure 2.1: Classical Vehicle Routing Problem

2.1 Applications of the VRP

1 Most Common Case
Each customer has a demand q_i (to be delivered or collected (e.g.: garbage collection)) and each vehicle has a capacity Q (see figure 2.1)

2 Upper Bound L
on the length of each route (e.g.: L=1000km, L=8 hours)

3 Time Windows
Customer i must be served between time a_i and time b_i (time window $[a_i, b_i]$

4 Combination of Collection and Delivery

5 Multiple Depots (see figure 2.2)

6 Heterogenous Vehicle Fleet
The vehicle fleet can be grouped due to different loading constraints.

7 Specialized vehicles
Large vehicles cannot reach all customers and therefore the huge vehicles carry the items as far as possible, then the items are unloaded and loaded on the small vehicles, that finally deliver the ordered items.

8 Periodic VRP
The planning horizon is expanded to a time period (e.g.: one week), which can be coded this way: $\{1,2,\ldots,7\}$, 1 means Sunday,and 7 means Saturday. Customers specify delivery combinations $\{2,5\}$ or $\{3,6\}$ or $\{4,7\}$ or $\{2,3,5\}$. Given that delivery dates vehicle routes for the whole week have to be planned.

9 Vehicles are not available the whole day

10 Different Objectives:

- Minimize the routing cost (km).
- Minimize the number of vehicles.
- 'soft time windows' (If these time windows are hurt the solution is feasible, but penalty costs are caused)

2.2 Heuristics

Criteria to assess the quality of a heuristic

1 Speed (5% of the planning horizon)

2 Flexibility

3 Accuracy

4 Simplicity

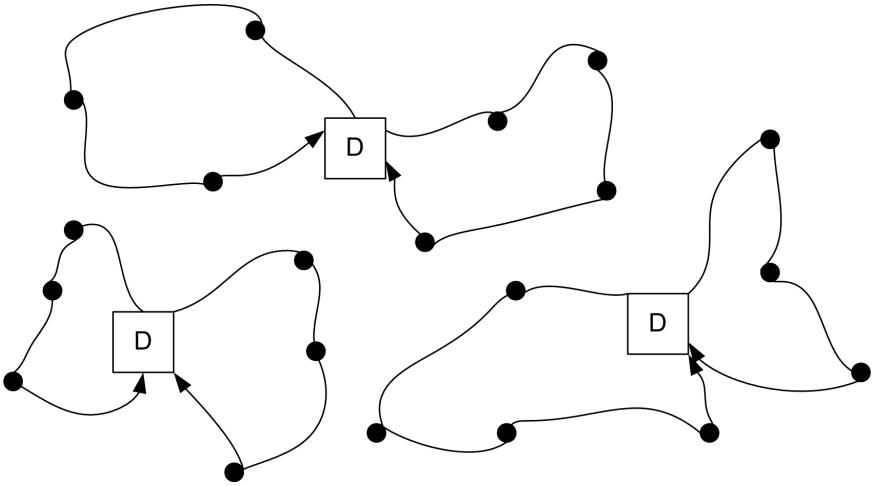


Figure 2.2: Multiple Depot VRP

2.2.1 Construction Heuristics

Savingsheuristic (Clarke and Wright, 1964)

Savings can be calculated by $s_{ij} = c_{i0} + c_{0j} - c_{ij}$ see figure 2.3

Figure 2.3: Savingsheuristic

- Compute Savingsmatrix (contains all savings between each customer i and j)
- Construct back and forth routes.
- At each iteration combine 2 routes with the largest saving $s_{ij>0}$ if it feasible.
- Repeat as long as possible.

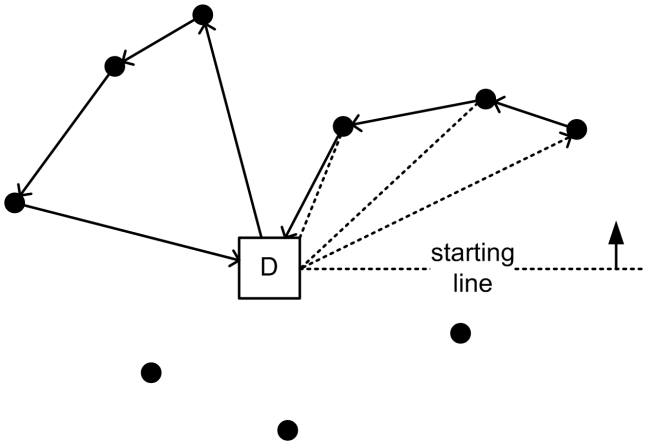


Figure 2.4: Sweep Algorithm

Sweep Algorithm (Gillett and Miller, 1974)

As you can see in figure 2.4 after determining a starting line, this line is rotated in a circle and whenever it touches a customer, this customer is added to the current vehicle route. If a constraint is hurt, a new route is started.
After the construction phase the solution is improved by improvement heuristics.
This algorithm does not work well in city problem with a plan view similar to Manhattan.

Two-phase Heuristic - Cluster first - Route second (Fisher and Jaikumar, 1980)

The idea in the **clustering phase** is to assign each customer to a seed in order to minimize assignment cost subject to capacity constraints (see figure 2.5).
d_{ik} = cost of assigning customer i to seed k, which leads to a GAP (Generalized Assignment Problem).

$$min \sum_{i,k} d_{ik} x_{ik}$$

$$\sum_{k} x_{ik} = 1 \qquad \forall i \quad \text{each customer i is assigned to one seed}$$

$$\sum_{i} q_i x_{ik} \leq Q \qquad \forall k \quad \text{capacity constraint}$$

$$x_{ik} \in \{0,1\}$$

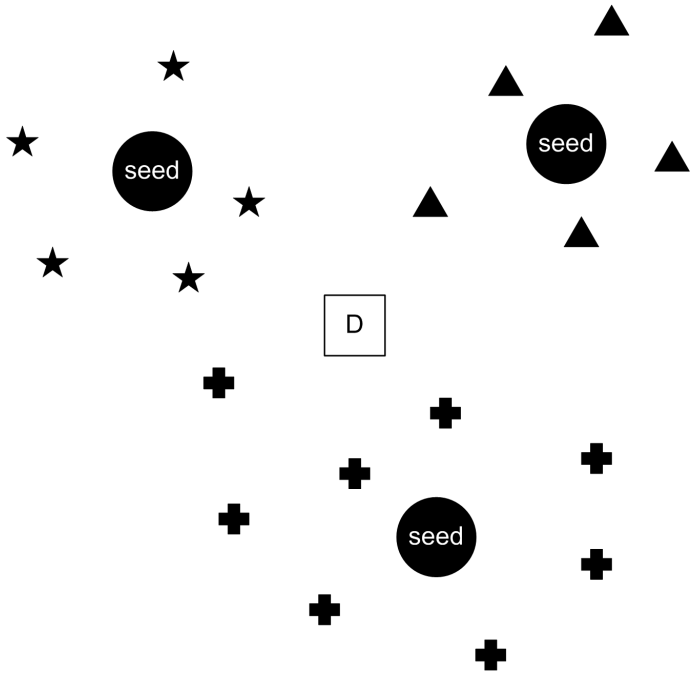


Figure 2.5: Cluster first - Route second

The GAP is a lot more difficult than the AP.
In the routing phase a TSP algorithm is applied to each cluster of customer given by the GAP solution.
One drawbak of this algorithm is that route length cannot easily be handled. On the other hand this algorithm is quite 'natural' because it follows two steps used by human planers.

Route first - Cluster second (Beasley, 1983)

In the **routing phase** a giant TSP is created (see figure 2.6).
In the clustering phase a shortest path problem like in figure 2.7

Petal Heuristics (Hjouring, Ryan, Glover, 1983)

This method has been also worked on by Renaud, Boctor and Laporte (1996).

1 Generate many feasible routes (petals) using for example sweep or another construction heristic

2 Make an optimal selection - Set Partitioning Problem

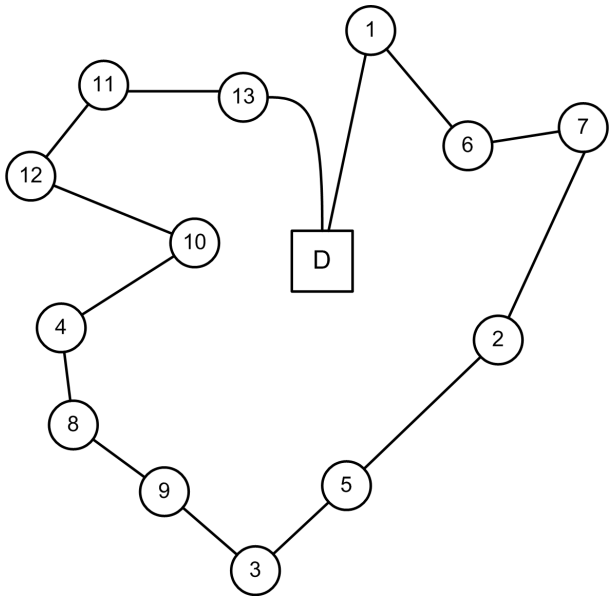


Figure 2.6: Routing Phase - Create a Giant TSP Tour

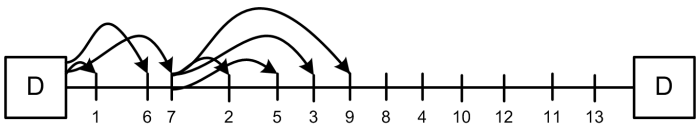


Figure 2.7: Clustering Phase - Shortest Path Problem

Heuristic	Speed	Flexibility	Accuracy	Simplicity
Savings	high	low	medium	high
Sweep	high	low	medium	high
Cluster first	high	low	good	medium
Route first	high	very low	very low	high
Petal	medium	high	good	medium

Table 2.1: Assessment for Different Construction Heuristics for the VRP

petals					
ik	1	2	3	...	
1	1	0			=1
2	0	0			=1
3	0	1			=1
4	1	0			=1
	1	1			=1
	0	1			=1
n	1	0			
c_k					

c_k = optimal cost of visiting all customers of petal k obtainable by solving a small TSP

$$x_k = \begin{cases} 1 & \text{if petal k is chosen} \\ 0 & \text{otherwise} \end{cases}$$

$$a_{ik} = \begin{cases} 1 & \text{if customer i is in petal k} \\ 0 & \text{otherwise} \end{cases}$$

$$min \sum_k c_k x_k$$

$$\sum_k a_{ik} x_k = 1 \qquad \forall i$$

$$x_k \in \{0, 1\}$$

2.2.2 Classical Improvement Heuristices

Intra-Route Improvements (see TSP Methods)

Inter-Route Improvements

- **Transfer**: Move customer i from route I (customer i is no longer part of route I) and insert it in all possible positions in route II.
- **Swap**: Exchange customer i from route I with customer j from route II
- **Lambda-Interchanges**: Osman 1991
  e.g.: lambda = 2
  $(0, 2)$ 0 customers are taken out of route I, but 2 customers are taken out of route II and inserted in route I
  $(0, 1)$
  $(1, 0)$
  $(1, 1)$
  $(1, 2)$
  $(2, 0)$
  $(2, 1)$
  $(2, 2)$

- **Ejection Chains**:
  Rego, Roucard (1996) and Thompson, Baraftis (1989)

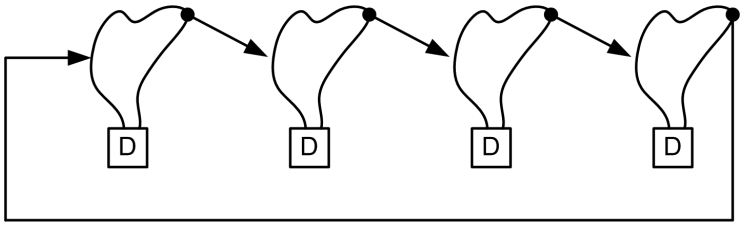


Figure 2.8: Ejection Chains

Classical improvement heuristics have two properties:

- The solution never deteriorates:

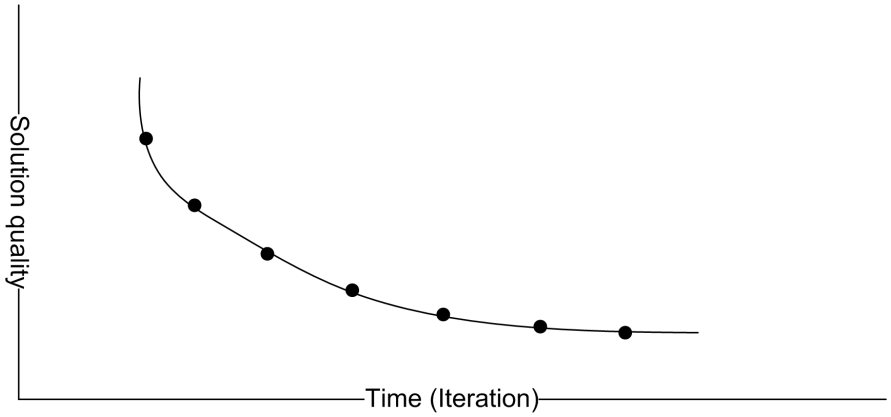


Figure 2.9: Descend Algorithm

- Solution always remains feasible.

2.3 Metaheuristics

So-called metaheuristics were first applied in 1989 and they have the following three main characteristics:

- *Local Search*
  Classical intra and inter route improvements are applied BUT deteriorating and infeasible solutions are allowed (see figure 2.10).
  Using classical heuristics would give the local optimum LO1 and the better solution in LO2 would not be possible to determine. Therefore the idea is to allow worse solutions to probably move in the search space into the area of a better local ore even the global optimum.
- *Solution Recombination*
  Many solutions are created and recombine them.
- *Learning*
  Repeat what worked well before.

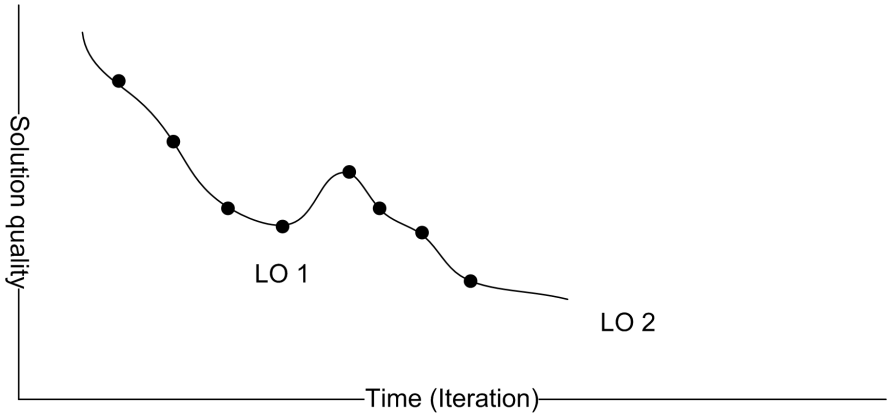


Figure 2.10: Traversing Local Optima

2.3.1 Local Search Metaheuristics

Simulated Annealing

x^* is the best solution so far
x_t is the solution at iteration t
$N(x_t)$ is the neighborhood of x_t
The neighborhood is the set of all solutions reachable from x_t by performing a special operation.
Procedure
At iteration t select randomly one solution x in the neighborhood. The function f gives the value of the solution:

- if $f(x) \leq f(x_t)$ then $x_{t+1} := x$
  if $f(x_{t+1}) \leq f(x^*)$ then $x^* := x_{t+1}$
- if $f(x) > f(x_t)$ then set $x_{t+1} := \begin{cases} x & \text{with probability} \quad p_t \\ x_t & \text{with probability} \quad 1 - p_t \end{cases}$

Concerning assessment Simulated Annealing is reasonably simple, flexible, a bit slow but more accurate then a descend algorithm.

Deterministic Annealing (Dueck ,1993)

Select randomly x in the neighborhood $N(x_t)$

- if $f(x) \leq f(x_t)$ then $x_{t+1} = x$

- if $f(x) > f(x_t)$ then
  - Version I: $x_{t+1} = \begin{cases} x & \text{if} \quad f(x) \leq \vartheta_i f(x^*) \\ x_t & \text{otherwise} \end{cases}$ where ϑ_i is a multiplicative component (e.g.: 1.01)
  - Version II: $x_{t+1} = \begin{cases} x & \text{if} \quad f(x) \leq \vartheta_k + f(x^*) \\ x_t & \text{otherwise} \end{cases}$ where ϑ_k is an additive component.

Tabu Search, Glover (1986)

At iteration t go to the best neighbor x of x_t in $N(x_t)$ even if $f(x) > f(x_t)$. With this simple rule cycling can occur because if the best neighbor in iteration t is worse than the current solution, in the next iteration the current solution is automatically a better neighbor solution. Therefore some moves have to be set tabu:
There are 60 seats in a classroom and you have to give each student a seat:

It	1	2	3	4	...	60	solution
1	S5	S1	S16				A
2	S5	S16	S1				B
3	S5	S1	S16				C

Solution C is equal to solution A due to cycling therefore all neighbors are allowed except those for which S1 (Student 1) sits in position 2. Due to this adjustment solution C becomes tabu, but also the following Configuration is tabu: S10 - S1 - S16
The tabu status is temporary and is lifted after certain iterations.
Revised Rule to Avoid Cycling:
At iteration t go to the best non tabu neighbor x of x_t in $N(x_t)$. Declare any solution with some characteristic of x_t tabu for ϑ iterations.
BUT: If the best neighbor x of x_t is tabu but $f(x) < f(x^*)$ then x_{t+1}:=x because cycling will not occur.
Tabu Search has been applied by many (more than 20) researchers to the VRP:

- **Willard, 1989**
- **Taillard, 1993**
  Defines $N(x_t)$ by means of 1-interchanges (which means move from route r to s

or move from route s to r or swap between route r and s)
If a vertex is moved from route r to route s at iteration t then it cannot go back to route r until iteration $t + \vartheta$, where ϑ is randomly selected in some integer interval (e.g.: $[5, 10]$). Periodically Taillard tries to improve the routes with TSP algorithm (intra route improvement).

- Taillard uses parallelism. He clusters all customers into 4 compartments and after 1o iterations the boundaries of which computer does what are changed.
- Taillard uses continuous diversification:
  if $f(x) \leq f(x_t)$ and x not tabu then $x_{t+1} := x$
  if $f(x) > f(x_t)$ then $f(x) := f(x) + \Delta$ where Δ is a term proportional to the number of times vertex v has been moved in the past.

Concerning assessment Taillard obtained 12 best solutions out of 14 on standard test problems. So far only one has been beaten. Therefore this algorithm is highly accurate, slow, not very simple but reasonable flexible.

- **Taburoute (Gendreau, Hertz, Laporte, 1994)**
  In some ways similar to Taillard but as neighborhood only simple moves are used (which means no interchanges) and for periodic intra route improvements different TSP algorithms are used.
  The tabu mechanism is equal and continuous diversification is also applied.
  Taburoute allows *intermediate infeasible solutions* for instance: The capacity constraint on a HGV $Q = 100$. The current solution obtains 3 routes which need the following capacities on their HGV: $q_1 = 97, q_2 = 98$ and $q_3 = 96$. Moving customer i from route 1 to route 2 with $q_v = 5$ violates the capacity constraint in route 2 ($q_2 = 103$), but moving in the next step customer j from route 2 to route 3 with $q_v = 4$ makes the whole solution again fesible, which would not have been obtained without intermediate infeasibility.
  The researches implemented this by a penalized function $f'(x) = f(x) + \alpha Q(x) + \beta L(x)$, which means that there are added the cumulative (of all routes) capacity overweight and the cumulative overduration of solution x to the objective function. The parameters α and β are dynamically adjusted. Initially they are both 1 and every 10 iterations they are adjusted in the following way:
  If the solutions in the last 10 iterations where feasible due to capacity constraints set $\alpha = \frac{\alpha}{2}$, else $\alpha = 2\alpha$. The same is done with respect of L.

Concerning assessment Taburoute is not so simple, quite accurate, faster than Taillard and very flexible due to the penalized objective function.

- **Unified Tabu Search Algorithm (Cordeau, Laporte, Mercier, 2001 and 2004)**
  UTSA is an improvement of Taburoute and there the following differences:

  1 fixed tabu duration

  2 UTSA can solve a large variety of VRPs including:
    - capacity constraints
    - route length constraints
    - time windows
    - multiple depots
    - multiple periods
    - etc.

  3 UTSA penalizes time windows like capacity and duration.
  $f'(x) = f(x) + \alpha Q(x) + \beta L(x) + \gamma W(x)$

  4 Adjustment of penalizing parameters at each iteration:
  if solution is feasible then divide parameter by 1.5
  if solution is infeasible then multiply parameter by 1.5

  Concerning assessment UTSA is more accurate, faster, simpler and more flexible than Taburoute.

- **Granularity Principle (Toth, Vigo, 2003)**
  The Granularity Principle is an approach to remove unnecessary data from a problem to accelerate the computations. To illustrate what is meant by removeing consider the following example in single depot VRP without time windows and with Euclidean distances:
  the depot has the coordinates (40,40), customer i (0,0) and customer j (80,80). It is very unlikely that customer i and j are connected (are within one route) in the optimal solution.
  Toth and Vigo use Savingsalgorithm by Clark and Wright as construction heuristic and compute the average edge length $\bar{c}$. Then they only keep the following edges:

  - all edges incident to the depot

- all edges $< \beta\bar{c} \quad \beta \in [1,2]$
  With this choice of β 80%- 90% of all edges are eliminated and the now leaner problem can be solved by any algorithm (they used Taburoute).

Concerning assessment they achieved slightly better solutions than Taburoute, reduced computing time by 75% and it is highly flexible.

2.3.2 Genetic Algorithms

Adaptive Memory Procedure (Rochat, Taillard, 1995)

Generate a pool of good solutions. New solutions are obtained by extracting high quality vehicle routes, where routes belonging to better solutions are given a higher probability of being selected. While extracting vehicle routes care is taken not to include those that contain already covered customers. Eventually this process will stop with a set of selected routes and unrouted customers, which is compensated by TS. If the gained solutions (baby) is better than a solution in the pool, kick out the worst pool solution and insert the baby.

Population Mechanism (Prins, 2005)

Idea???

1 Generate a giant TSP Tour

2 Population is a set of solutions

3 Genetic algorithm
Parent 1: 1-3-6-4-5-8-2-7-9-1
Parent 2: 1-5-9-2-6-4-7-8-3-1
Baby 1: Take sequence 1-3-6 from P1 to get 1-3-6-4-7-8-5-9-2-1
Baby 2: Take sequence 2-6-4-7 from P2 to get 2-6-4-7-9-1-3-5-8-2
Improve by 2-opt!

3 Location Problems

3.1 Applications

1 Decision Level

- Strategic Level
  - new town
  - airport
  - metro system
  - major factory
- Tactical Level
  - warehouse
  - McDonalds
  - busline
- Operational Level
  - where to put merchandize in a warehouse
  - post boxes
  - recycling base

2 What to locate?

- area
  - airport
  - nuclear bomb testing site
- network (metro)
- line (electricity lines)
- point

3 Which criteria to use?

- location availability
- cost
- accessibility
- coverage
- market share
- flow interception (where to locate a gas station)
- anti-accessibility (garbage dumps, bomb testing sites)

3.2 Problems

3.2.1 Median Problems

A median is a point that minimizes the total distance between it and the users. Given the planar problem in figure 3.1 one can calculate total distances for each location.

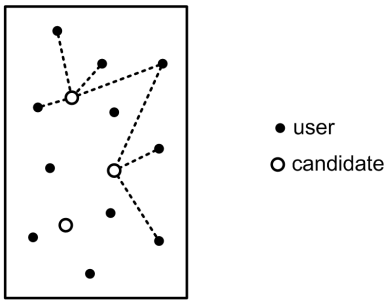


Figure 3.1: Median Problem

3.2.2 p-Median Problems

Locate p facilities to minimize the total cost (distances) between users and their closest facility. Given the network in figure 3.2 two schools should be located in the network (The numbers in the vertices represent the number of pupils. Hakimi (1964) proposed that it is never suboptimal to locate the facilities at the vertices of the network.
Suppose that L people enter from the left (see figure 3.3) and R people go from the right. Therefore on this edge the total distance travel is $Lx + R(1 - x)$. Minimizing this with respect to $0 \leq x \leq 1$ gives $x = 1$ or $x = 0$.

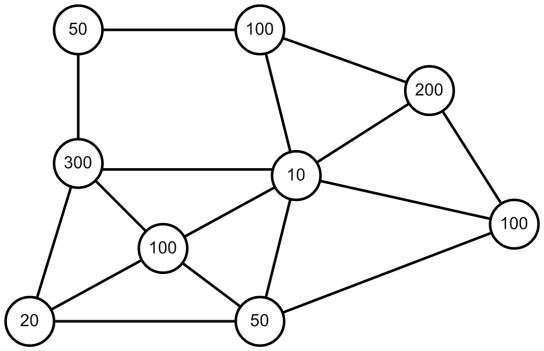


Figure 3.2: p-Median Problem

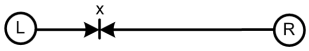


Figure 3.3: Location at vertices is never suboptimal

How to Solve a p-Median Problem

n number of candidate locations on the network (possibly all vertices)
p number of facilities to be located

- Method 1: Enumeration
  If $\binom{n}{p} = \frac{n!}{p!(n-p)!}$ is small enumerate all possible solutions and choose the best.
- Method 2: Integer Linear Programming
  p number of facilities
  d_i population of vertex i
  c_{ij} distance between vertices i and j

  $$y_j = \begin{cases} 1 & \text{if facility is located at vertex j} \\ 0 & \text{otherwise} \end{cases}$$

  $$x_{ij} = \begin{cases} 1 & \text{if user i goes to facility located at j} \\ 0 & \text{otherwise} \end{cases}$$

$$min \sum_i \sum_j d_i c_{ij} x_{ij}$$

$$\sum_j y_j = p$$

$$\sum_j x_{ij} = 1 \qquad \forall \quad i$$

$$x_{ij} \leq y_j \qquad \forall \quad j$$

$$x_{ij} \in \{0,1\} \qquad \forall \quad i,j$$

$$y_j \in \{0,1\} \qquad \forall \quad j$$

- Method 3: Heuristic
  - Locate p facilities randomly.
  - Attempt to improve by relocating one facility at a time. Repeat as long as improvements are possible.

  This method works very well and can be improved by repeating with different starting solutions in the first step.

3.2.3 Covering Problems

Total Coverage Problem

Cover *all* users using the smallest number of facilities.

$$a_{ij} = \begin{cases} 1 & \text{if facilitiy j covers user i} \\ 0 & \text{otherwise} \end{cases}$$

$$y_j = \begin{cases} 1 & \text{if a facility is located at j} \\ 0 & \text{otherwise} \end{cases}$$

$$min \sum_j y_j$$

$$\sum_j a_{ij} y_i \geq 1$$

$$y_j \in \{0,1\} \qquad \forall \quad j$$

$$a_{ij} \in \{0,1\} \qquad \forall \quad i,j$$

Maximum Coverage Problems

d_i population at vertex i

p number of ambulances

$$z_i = \begin{cases} 1 & \text{if i is covered} \\ 0 & \text{otherwise} \end{cases}$$

$$a_{ij} = \begin{cases} 1 & \text{if facilitiy j covers user i} \\ 0 & \text{otherwise} \end{cases}$$

$$y_j = \begin{cases} 1 & \text{if a facility is located at j} \\ 0 & \text{otherwise} \end{cases}$$

$$max \quad z(p) = \sum_i d_i z_i$$

$$\sum_j y_j = p$$

$$z_i \leq \sum_j a_{ij} y_j \qquad \forall i$$

$$z_i \in \{0,1\} \qquad \forall i$$

$$y_j \in \{0,1\} \qquad \forall j$$

In practice one can observe the curve depicted in figure 3.4.

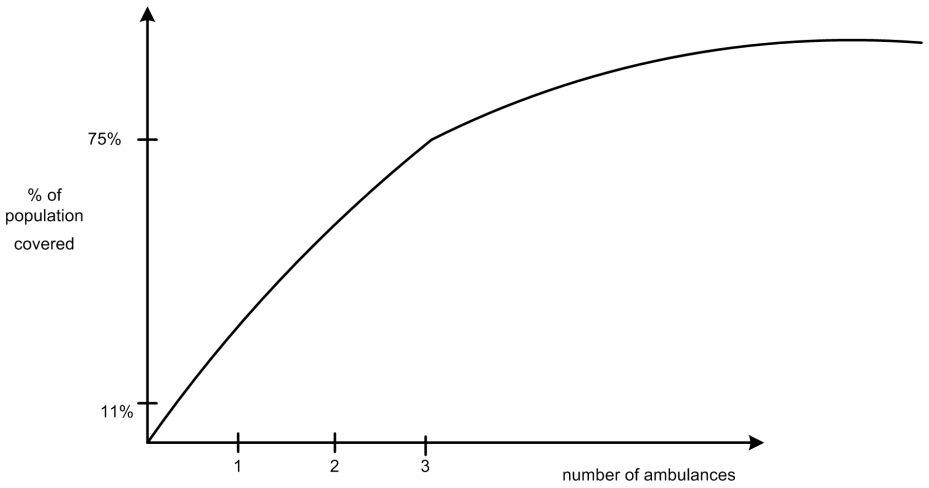


Figure 3.4: p-Median Problem

4 Arc Routing Problems

4.1 Applications and Graphical Representation

- Mail Delivery
- Garbage Collection
- Street Cleaning
- Snow Removing
- Meter reading
- ...

ARC routing problems are defined on mixed graphs and (un)directed graphs. The graphical representation is depicted in figure **??**.

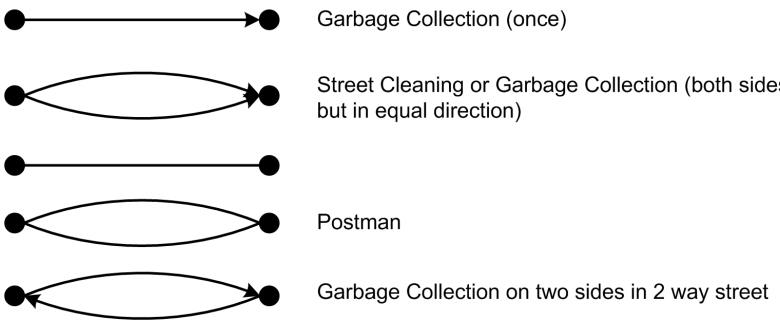


Figure 4.1: Graphical Representation

4.2 Problems

4.2.1 Undirected Graphs

In 1736 Leonhard Euler was confronted with the following problem: The inhabitants of Knigsberg debated wether there existed a closed walk using each of the seven bridges

of the Pregel River exactly once (see figure 4.2).
If some vertex has an odd degree, then there is no closed walk using each edge exactly once, otherwise it is possible and the graph is called Eularian or unicursal.
A necessary condition for an undirected graph to be Eularian is

- The Graph must be connected.
- All vertex degrees must be even.

In a graph the number of odd degree vertices is even because the total degree is even (two times the number of edges) which is equal to the total degree of even degree vertices (even) plus the total degree of odd degree vertices (even). Hierholzer (1873) pro-

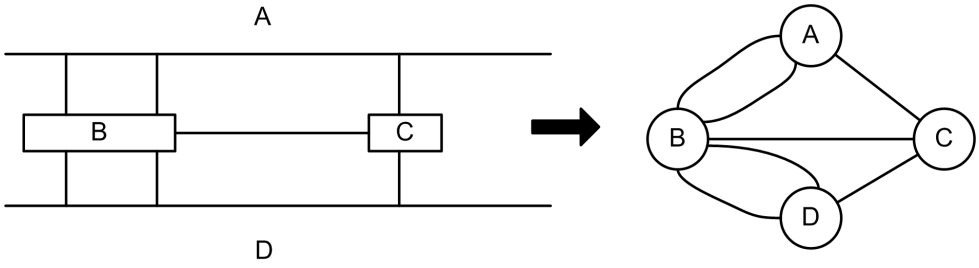


Figure 4.2: Graphical Representation

vided an algorithm to traverse a Eularian graph (End-Pairing Algorithm):

Step I Draw a first cycle by visiting unvisited edges until this is no longer possible.

Step II Have all edges been visited?
If yes → Stop!
If no → Continue with next step!

Step III Draw a second cycle in the same way starting from an unvisited edge incident to the first cycle.

Step IV Merge the two cycles and go to step II.

Suppose that in the example in figure 4.3 the post office is located in F. A closed walk can be generated as follows:
Cycle I: F-H-G-D-F-E-F
Cycle II: E-A-B-C-E
Merge cycle I and II which gives:

Cycle III: F-H-G-D-F-E-A-B-C-E-F
Cycle IV: D-C-A-I-J-K-G-J-I-D
Merge cycle III and IV: F-H-G-D-C-A-I-J-K-G-J-I-D-F-E-A-B-C-E-F
In practice however most graphs are not Eularian (every T-junction in a city has an

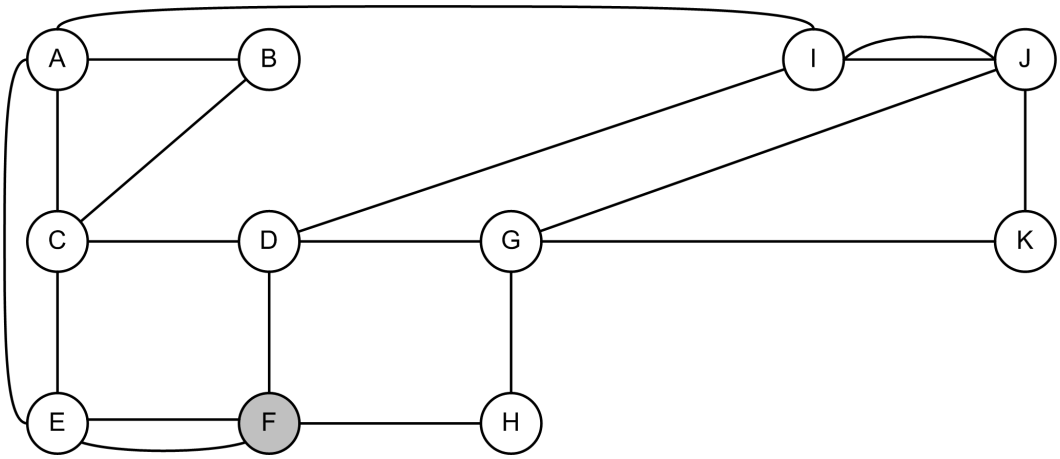


Figure 4.3: Eularian Graph

odd degree). Supermarkets tend to be non-Eularian too. Looking at figure 4.4 at the left side is the outline of an ordinary supermarket. Customers have two choices: Either they traverse several edges twice or the miss some. On the right hand side there is an alternative outline which is Eularian and therefore each edge has to be traversed only once.

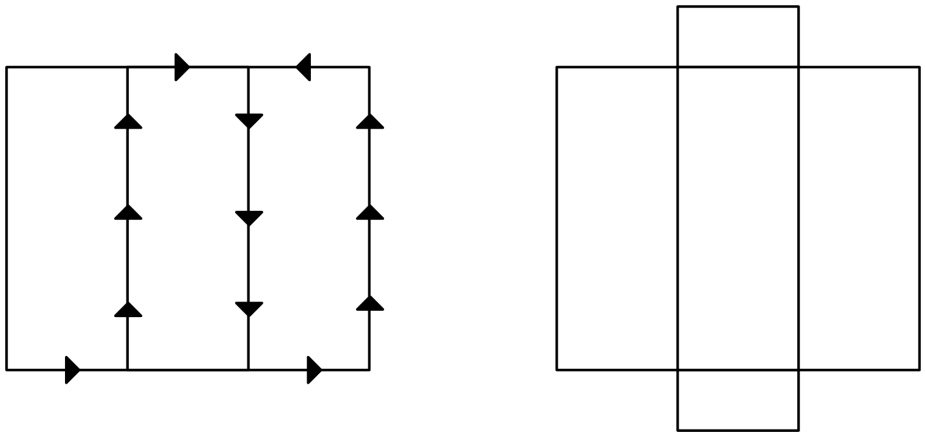


Figure 4.4: Outline of a Supermarket

4.2.2 The Chinese Postman Problem (CPP)

Given the graph in figure 4.5 all edges have to be traversed in the most economical way (guan 1962). The idea is to add paths between pairs of odd degree vertices:
In this case there are three possibilities:

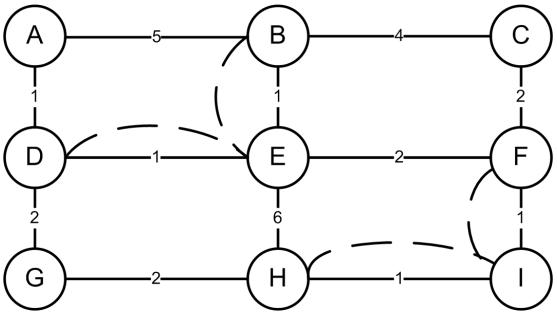


Figure 4.5: Undirected CPP Example

$$\overline{BD} + \overline{FH} = 2 + 2 = 4$$
$$\overline{BF} + \overline{DH} = 3 + 4 = 7$$
$$\overline{BH} + \overline{DF} = 5 + 3 = 8$$

Algorithm for the CPP on an undirected graph:

Step I If the graph is Eularian, the end-pairing algorithm has to be applied, otherwise the graph has to be augmented.

Step II Compute a shortest path for every vertex pair (i,j) off odd degree.

Step III Find the best combination of shortest paths using a matching algorithm (Edmonds an Johnson, 1973).

$$c_{ij} = \text{matching cost}$$

$$x_{ij} = \begin{cases} 1 & \text{if vertices i and j are matched} \\ 0 & \text{otherwise} \end{cases}$$

$$min \sum_{i<j} c_{ij} x_{ij} \qquad \text{(minimum cost augmentation problem)}$$

$$\sum_{i<k} x_{ik} + \sum_{j>k} x_{kj} = 1 \qquad \forall k$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j$$

This minimum cost augmentation problem is a relatively easy problem.
The edges added to create the augmented graph are called *dead headed edges* (they are traversed, but not serviced). In an undirected graph an edge is dead headed at most once.

4.2.3 Directed Graphs

Necessary conditions for a directed graph to be Eularian:

- Graph must be strongly connected (it must be possible to go from anywhere to anywhere else).
- For each vertex the in-degree must be equal to the out-degree.

If a graph is Eularian it is possible o traverse it by using end-pairing algorithm by taking directions into account.
Algorithm for the CPP on a directed graph:

Step I If the graph is Eularian apply end-pairing algorithm otherwise go to step 2.

Step II Identify the deficit and surplus vertices.

Step III Solve the following transportation problem:

$$S = \text{set of surplus vertices}$$
$$D = \text{set of deficit vertices}$$
$$c_{ij} = \text{length of shortest path from i to j}$$
$$x_{ij} = \text{number of paths added from i to j}$$
$$s_i = \text{surplus of i}$$
$$d_j = \text{deficit of j}$$
$$min \sum_{i \in S} \sum_{j \in D} c_{ij} x_{ij}$$
$$\sum_{j \in D} x_{ij} = s_i \qquad \forall i \in S$$
$$\sum_{i \in S} x_{ij} = d_j \qquad \forall j \in D$$
$$x_{ij} \geq 0 \quad \text{and integer} \qquad \forall i \in S, j \in D$$

Step IV Go to step I.

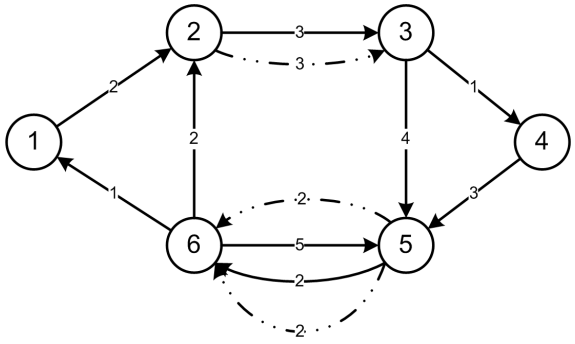


Figure 4.6: Directed CPP Example

Concerning the example presented in figure 4.6 three types of vertices can be distinguished:

- *balanced vertices*, where in-degree is equal to the out-degree (vertices 1 and 4)
- *deficit vertices*, where the in-degree is smaller than the out-degree (vertices 3 and 6)
- *surplus vertices*, where the in-degree is greater than the out-degree (vertices 2 and 5)

To economically augment the graph the following problem has to be solved:

S/D	3	6	
2	1 (3)	(9)	1
5	(7)	2 (2)	2
	1	2	

As depicted in dashed lines 1 edge between vertex 2 and vertex 3 has to be added and 2 edges between vertex 5 and vertex 6.

4.2.4 The Undirected Rural Postman Problem (RPP), Ortoff (1976)

In a graph $g = (V, E)$ $R \subseteq E$ is a set of *required edges*, that must be serviced. The edges of $E \setminus R$ may be used but do not require service. In figure 4.7 there are $p = 3$ connected components. This problem is difficult to solve exactly. Therefore in practice one uses heuristics:

Frederickson's Constructive Heuristic (1979)

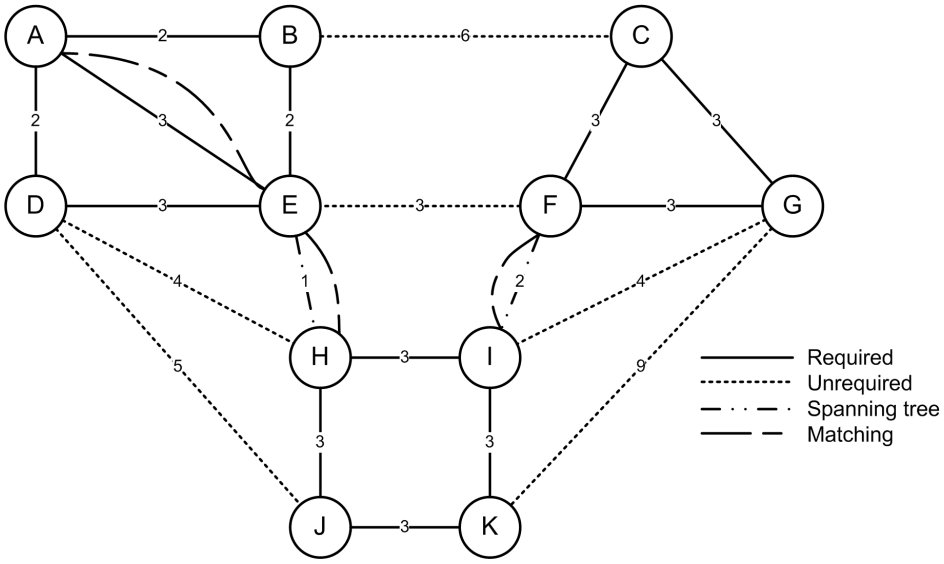


Figure 4.7: RPP

Preliminary Problem: Shortest Spanning Tree problem: Choose the next cheapest edge that does not cycle as long as possible.

Step I Construct a shortest spanning tree to link the connected components.

Step II Identify the odd degree vertices in the graph induced by

- the connected components and
- the spanning tree

and match them optimally as in the undirected CPP.

Step III Apply the end -pairing algorithm over the graph induced by

- the connected components
- the spanning tree
- the matching edges

In practice this heuristic works quite well (about 3% above the optimum) but can improved by the procedure SHORTEN due to Hertz, Laporte and Nanchan-Hugo (1999).

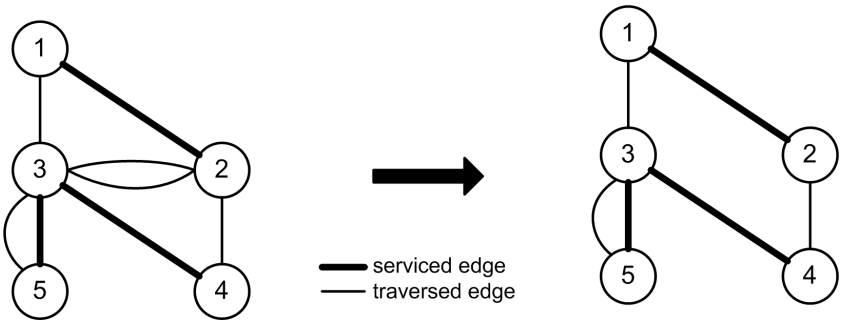


Figure 4.8a: Initial solution and final solution of RPP

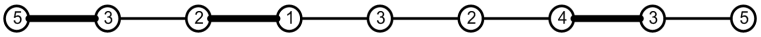


Figure 4.8b: Initial Solution presented as circular vector

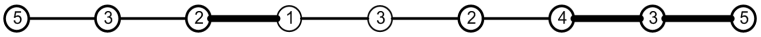


Figure 4.8c: Applying POSTPONE to edge (5,3)

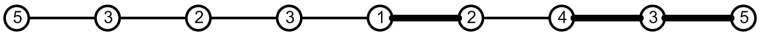


Figure 4.8d: Applying REVERSE to chain (2,1,3,2)

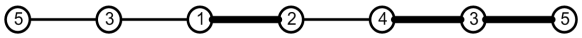


Figure 4.8e: Shortening chain (5,3,2,3,1) into (5,3,1)

4.2.5 The Capacitated Arc Routing Problem (CARP) Golden, Wong (1981)

The CARP is like an arc-VRP.
Design a set of vehicle routes that start and end at the depot, respect the capacity constraints at minimum total cost. (In practice there are often route length constraints, too)
Algorithm Route first - Cluster second by Hertz, Laporte and Miller (2000)

Step I Solve a RPP over all edges by Frederickson's algorithm.

Step II CUT the mega-tour into feasible vehicle routes.

Step III SHORTEN each route.

Step IV PASTE: Merge all routes into a single RPP tour.

Step V SWITCH: Reverse some chains that start and end at the same vertex to avoid cycling.

Step VI Stop or start with Step 1.

To obtain better results one could apply a tabu search similar to Taburoute (move edges to different routes instead of vertices)