Chapter 6

Pickup and delivery problems
with services on nodes or arcs of a network

6.1. Introduction

Pickup and delivery are activities that appear frequently in numerous companies and communities. It generally concerns transporting merchandise from one depot to branch offices, delivering orders to clients, maintaining road networks, distributing mail, picking up rubbish, etc. These operations are often very costly and substantial savings can be made by reducing the length of the trips that must be made.

According to the nature of the service furnished by the vehicles and in function of the geographical distribution of the clients, two large types of modelling emerge. The first consists in representing the clients by nodes of a network. The problem based on this model is called the node routing problem (NRP). Such a model is well adapted when the clients have precise distinct locations. The NRP has been and is still the object of many studies. Numerous variants of the NRP exist, going from the basic traveling salesman problem to problems with many more constraints, such as node routing problems with time windows.

Another type of modelling associates the clients to the arcs of a network. We then speak of an arc routing problem (ARP). An arc is a direct link between two nodes in a network. Less present in the literature than the NRP, the ARP appears when the clients are located along the streets, or when the streets themselves need a service. Just as for the NRP, there are different variants of the ARP.

Chapter written by Alain Hertz and Michel Mittaz
Most of the variants of the NRP and the ARP are \textit{NP-hard}, that is, there exists today no algorithm capable of finding an optimal solution in computing time that doesn’t explode when the size of the problem augments. To solve a NRP or an ARP of large size, we have to rely on heuristic methods that determine an approximation of the optimum of the problem. We present in this chapter a review of the principal heuristics that have been developed for the solution of the diverse variants of the NRP and the ARP.

6.2. Node routing problems

6.2.1. The traveling salesman problem

The traveling salesman problem was probably mentioned for the first time in a mathematical circle by A.W. Tucker in 1931. This problem has already been introduced in chapter 3 in the general context of solution methods. Here, we present specific heuristics adapted to this problem. Let’s remember that the formulation is particularly simple, which has captivated many researchers.

**PROBLEM.** *A salesman, departing from home (or a depot), must visit a set of clients and then return home; the order in which he visits the clients should be such that the total distance he travels is as small as possible.*

It isn’t necessary to be a particularly talented researcher to imagine algorithms producing good tours for the traveling salesman. However, the problem is NP-hard, and the best exact techniques (that is, guaranteeing optimality of the solution produced) can only treat instances with a few hundreds of clients. This explains why numerous researchers worked on developing efficient heuristics to solve the traveling salesman problem [LAW 85]. Among the most well known constructive heuristics, we will mention here only three. First of all, the \textit{Nearest Neighbour} algorithm is probably the simplest method to implement. It is described in algorithm 6.1.

\textbf{Nearest Neighbour algorithm}

1. Choose a client \(v\) and consider a partial tour \(T=\text{(depot, } v\text{, depot)}\).

2. Determine a client \(w\) who is not yet in \(T\) and who is the nearest to the last client \(v\) visited in \(T\).

3. Add client \(w\) to the end of tour \(T\). If all the clients have been visited, then STOP, if not return to 2.

\textbf{Algorithm 6.1. Nearest Neighbour algorithm}
Another algorithm just as simple as the preceding is the Cheapest Insertion algorithm. It is described in algorithm 6.2.

**Cheapest Insertion algorithm**

1. Choose two clients \( v \) and \( w \) and consider a partial tour \( T \) visiting only these clients, that is, \( T = (\text{depot}, v, w, \text{depot}) \).
2. For each client \( z \) who is not yet in \( T \), calculate the length \( L(z) \) of the smallest detour brought about by inserting \( z \) between two consecutive clients on \( T \).
3. Choose a client \( z \) who is not yet in \( T \) and who minimises \( L(z) \), and insert \( z \) in \( T \) (with the smallest detour). If all the clients have been visited, then STOP, if not return to 2.

**Algorithm 6.2 Cheapest Insertion algorithm**

These two heuristics for the traveling salesman problem generally produce solutions of reasonably good quality. They are illustrated in Figure 6.1. We can easily construct examples for which these heuristics are arbitrarily bad. More precisely, if we note \( H(I) \) the length of the tour produced by one of these two heuristics for an instance \( I \) of the problem, and if \( OPT(I) \) is the minimal length of a tour for instance \( I \), than the ratio \( H(I)/OPT(I) \) has no upper bound. This means that for each value \( r > 1 \) we can construct an instance \( I \) for which \( H(I)/OPT(I) \) is superior to \( r \).

By noting \( D(x, y) \) the distance between two clients \( x \) and \( y \), we say that the distances satisfy the triangular inequality if \( D(x, y) \leq D(x, z) + D(z, y) \) for each triplet.
x, y, z of clients. In 1976, Christofides [CHR 76] proposed a heuristic guaranteeing that the length of a produced tour is in the worst case 50% superior to the optimum, assuming that the distances satisfy the triangular inequality. This heuristic is described in algorithm 6.3.

**Christofides’ algorithm**

1. Determine a maximal tree of minimum cost $A$ connecting the clients.
2. Let $W$ be the set of clients who are the extremity of an odd number of arcs in $A$. This set necessarily contains an even number of nodes. Determine a maximal matching $C$ of the elements of $W$ of minimum cost.
3. The union of $A$ and $C$ induces a tour $T$ that can eventually be shortened if we pass by the same client twice.

**Algorithm 6.3. Christofides’ algorithm**

Determining a maximal tree of minimum cost at step 1, and a maximal matching of minimum cost at step 2, can be realised in polynomial time (see Chapter 2). This algorithm is illustrated in Figure 6.2.

**Figure 6.2. Illustration of Christofides’ algorithm:** a) set of clients to visit; b) maximal tree of minimum cost. The nodes in grey are the extremity of an odd number of arcs; c) maximal matching of minimum cost on the grey nodes; d) union of the tree with the matching; e) shortened tour that avoids passing by the same client twice.

The algorithms described up to here were all proposed in the 1960’s and 1970’s. More efficient methods were proposed in the 1980’s and new algorithms are regularly published in the scientific literature. The most efficient current methods are based on *Local Search techniques* (descent algorithm, simulated annealing, tabu search, see chapter 3).
Numerous adaptations of Local Search techniques have been proposed for the solution to the traveling salesman problem. The set of solutions $S$ to explore is simply the set of tours visiting all clients, and the function $F$ to minimise is the length of these tours. The diverse adaptations proposed differ mainly in the definition of the notion of neighbourhood $N(s)$ of a solution $s$. A very common technique consists of putting in $N(s)$ all the tours that can be obtained by replacing two arcs of $s$ by two arcs that are not in $s$. Two neighbour solutions are represented in Figure 6.3. More refined neighbourhoods have been proposed, and we refer the reader to [JOH 97] for more details on Local Search techniques to solve the traveling salesman problem. It clearly appears from the numerous experiments carried out that the Local Search techniques generally give tours whose length is less than one percent superior to the optimal length of a tour.

![Figure 6.3. Two neighbour tours](image)

**Figure 6.3.** Two neighbour tours: The tour on the right is obtained from the tour on the left by replacing two arcs with two new arcs.

### 6.2.2. Vehicle tours with capacity constraints

If merchandise must be delivered to clients, it is often necessary to make deliveries with the help of several vehicles, each vehicle having only a limited capacity. Delivery planning then necessitates the solution of two sub-problems:

- we must determine the set of clients that each vehicle must serve;
- we must determine the order in which each vehicle serves its clients.

We will suppose here that all the vehicles have the same load capacity, that each vehicle starts and finishes its tour at the depot, and that the deliveries to a client are made with the help of a single vehicle (that is, we forbid all solutions where several vehicles share delivery to the same client). The goal is to minimise the total distance.

To solve this problem, Clarke and Wright [CLA 64] proposed in 1964 a constructive algorithm, which is still frequently used by companies today. To understand this algorithm, it is necessary to define the notion of saving.
DEFINITION.- Given two clients \( v \) and \( w \), the saving \( s(v, w) \) is defined as the gain in length obtained by delivering \( v \) and \( w \) in the same tour (depot, \( v \), \( w \), depot) instead of using two tours (depot, \( v \), depot) and (depot, \( w \), depot).

By noting \( D(x, y) \) the distance between two nodes \( x \) and \( y \) of the network, we thus have \( s(v, w) = D(v, \text{depot}) + D(\text{depot}, w) + D(v, w) \). This concept is illustrated in Figure 6.4.

Clarke et Wright’s algorithm constructs a first route by inserting step by step, at the start or at the end of the tour, clients that bring about the greatest savings. When no other client can be added to this first tour without inducing a capacity overload, the algorithm constructs a second route, according to the same principle, with the remaining clients. This process is repeated until all the clients have been served. This algorithm is more precisely described in algorithm 6.4.

**Clarke and Wright’s algorithm**

1. Determine the savings \( s(v, w) \) for all pairs of clients, and arrange these values in decreasing order.
2. Choose the first saving \( s(v, w) \) from the list so that \( v \) and \( w \) are not yet served, and so that the sum of requests of clients \( v \) and \( w \) does not exceed the vehicle capacity. Create a tour \( T = (\text{depot}, v, w, \text{depot}) \).
3. Choose the first saving \( s(v, w) \) of the list so that \( v \) is the first or the last client served in tour \( T \), \( w \) is still not served by a vehicle, and the merchandise to deliver to client \( w \) can be added to tour \( T \) without inducing a vehicle capacity overload.
4. Add \( w \) to the end of \( T \) if \( v \) was the last client of the tour, and at the beginning of \( T \) if \( v \) was the first client of the tour.
5. If all the clients are served, STOP. If not, if no client can be added to \( T \) without causing overload of the vehicle capacity, then memorise this tour and return to step 2. If not return to step 3.

*Algorithm 6.4 Clarke and Wright’s algorithm*
This algorithm has now been improved, mainly with the help of more adequate definitions of the notion of saving. More details on this type of technique can be obtained by consulting [GOL 85].

The two-phase methods described below generally give better results than a constructive algorithm. These methods are principally of two types.

- The cluster first – route second strategy first determines a partition of the clients into clusters, each having a total demand not exceeding the vehicle capacity. In a second phase, a traveling salesman problem is solved on each cluster.

- The route first – cluster second strategy first solves a traveling salesman problem to construct a giant tour visiting all clients (without taking vehicle capacity into account). Then, in a second phase, the tour is partitioned into smaller tours satisfying all the vehicle capacity constraints.

These two strategies are illustrated in Figure 6.5. Examples of using these strategies are described in [CHR 85].

Figure 6.5. The two strategies of the two-phase methods: a) a set of clients to visit; b) partition of the clients into clusters; c) construction of a tour on each cluster; d) solution of the traveling salesman problem; e) partition of the tour into smaller tours satisfying all the vehicle capacity constraints.
Local Search techniques can also be adapted to the routing problems with capacity constraints. In this case, a solution $s$ is defined as being a set of tours satisfying all the client requests and respecting capacity constraints. The value $F(s)$ of a solution is the total distance covered by the set of vehicles, and the neighbourhood $N(s)$ of a solution $s$ contains for example the set of solutions that can be obtained from $s$ by moving the service of a client from one vehicle to another. Two neighbour solutions are represented in Figure 6.6.

![Figure 6.6. Two neighbour solution: transfer of a client from one tour to another.](image)

The adaptation of Local Search techniques described above can sometimes pose a problem because of limited vehicle capacities. Let’s consider for example a solution constituted of two tours $T_1$ and $T_2$, the first comprising 4 clients, each having a request for one unit of merchandise, and the second having only two clients with requests for two units of merchandise. If vehicle capacities are limited to 4 units of merchandise, it isn’t possible to move a client from one tour to another without causing a capacity overload. Indeed, if a client is transferred from $T_1$ toward $T_2$, the tour $T_2$ will have an overload of one unit, and if a client is transferred from $T_2$ toward $T_1$, it is an overload of two units that we will have in $T_1$. Even a permutation between a client of $T_1$ and a client of $T_2$ does not solve the problem since that would induce an overload of one unit in $T_1$. It is thus not possible to regroup the clients differently in the tours while the optimal solution can eventually consist of two tours, each having a client with a request of 2 units and two clients with a request for one unit.

To get around this problem, we can decide to accept violations of the capacity constraint. In other words, a solution $s$ is a set of tours satisfying all the client requests but not necessarily respecting capacity constraints. When a solution $s$ violates capacity constraints, we define a penalty $P(s)$ proportional to the capacity
overloads. The value $F(s)$ of solution $s$ is then obtained by adding the total distance covered by vehicles with the penalty $P(s)$. Such an approach has been used with success in [GEN 94].

6.3. Arc routing problems

In this section we will describe some algorithms to solve vehicle routing problems in which clients are situated on the arcs of a network. If all the arcs in the network are oriented, we can easily transform an ARP into a NRP in the following manner. We create a complete network $G$ where each node corresponds to an arc $a \rightarrow b$ to be serviced. The distance between two clients $a \rightarrow b$ and $c \rightarrow d$ is the length of the shortest path connecting the terminal extremity $b$ of $a \rightarrow b$ to the initial extremity $c$ of $c \rightarrow d$. By solving a NRP in $G$, we obtain a tour $T$ that can then easily be transformed into a solution $S$ of the ARP. The difference in length between $T$ and $S$ corresponds to the total length of the arcs to be serviced. This transformation is illustrated in Figure 6.7.

![Figure 6.7. Transformation of an oriented ARP into a NRP: a) network in which one must solve an ARP. Clients are represented in heavy lines; b) associated graph in which one must solve a NRP; c) optimal tour $T$ for the NRP. Its length is 3; d) optimal tour $S$ of the ARP, deduced from tour $T$. Its length is 7=3+4.]

6.3.1. The Chinese postman problem

The Chinese postman problem was introduced for the first time by the mathematician Meigu Guan in 1962 [GUA 62]. Given a graph, it is the problem of determining a circuit of minimum total length traversing each arc of the graph at least once. The optimal solution to this problem can be obtained in polynomial time when the graph is non oriented (each arc can be traversed in any direction) or totally oriented (one direction is prohibited for each arc in the graph). On the other hand, if certain arcs in the graph have an orientation imposed while others don’t, we then speak of a mixed graph and the problem to solve becomes NP-hard.
If each node of the considered network $G$ has an even number of neighbours, and if the arcs are non oriented, the Chinese postman problem consists of determining a cycle passing exactly once through each arc in the graph. This problem can easily be solved with algorithm 6.5. It is illustrated in Figure 6.8.

**Algorithm for the non oriented Chinese postman problem, where each node has an even number of neighbours**

1. Determine a cycle $C$ in the graph. If $C$ covers all arcs, STOP.
2. Choose a node $v$ belonging to $C$ and incident to an arc that is not in $C$. Construct a second cycle $C'$ containing $v$ so that $C$ and $C'$ have no arc in common.
3. Join $C$ and $C'$ to form a cycle $C''$. This merging is done by departing from node $v$, covering the arcs of cycle $C$, then being back in node $v$ by visiting the arcs of cycle $C'$ to finally terminate in $v$.
4. Set $C$ equal to $C''$. If $C$ covers all the arcs, STOP; if not, return to 2.

Algorithm 6.5. Algorithm for the non oriented Chinese postman problem, where each node has an even number of neighbours

Figure 6.8. Solution of the Chinese postman problem in a network where all nodes have an even number of neighbours: a) network in which we want to construct a tour. All the nodes have an even number of neighbours; b) detection of a first cycle $(a, b, c, d, e, f, g, h, a)$; c) detection of a second cycle $(f, k, i, c, j, f)$ joined with the first cycle by node $f$. We thus obtain a new cycle $(a, b, c, d, e, f, k, i, c, j, f, g, h, a)$; d) detection of a third cycle $(k, i, j, k)$ joined with the preceding cycle by node $k$. We thus obtain a new cycle $(a, b, c, d, e, f, k, i, j, k, h, i, c, j, f, g, h, a)$.

If the nodes of the graph have an odd number of neighbours, we can duplicate a certain number of arcs so that the augmented graph has only nodes with an even number of neighbours. The choice of arcs to duplicate so that the Chinese postman tour is of minimal length can be done with the help for the search of a maximal
Pickup and delivery problems

matching of minimum cost in a complete auxiliary graph (that is, where all nodes are connected two by two). More precisely, the algorithm is described in algorithm 6.6.

**Algorithm for the non oriented Chinese postman problem**

1. If all the nodes in the graph $G$ have an even number of neighbours, then determine a tour with the help of the preceding algorithm, STOP.
2. Determine the set $W$ of nodes having an odd number of neighbours. This set necessarily contains an even number of nodes. Construct a complete auxiliary graph $G'$ where the set of nodes is $W$ and whose cost associated to each arc $(v, w)$ is equal to the length of the shortest path between $v$ and $w$ in the original graph.
3. Determine a maximal matching of minimum cost in $G'$. For each arc $(v, w)$ of this matching, add in the original graph a copy of all the arcs belonging to a shortest path from $v$ to $w$. The nodes in the augmented graph now have an even number of neighbours.
4. Determine a tour in the augmented graph with the help of the algorithm 6.5.

**Algorithm 6.6. Algorithm for the non oriented Chinese postman problem**

Steps 2 and 3 of this algorithm are illustrated in Figure 6.9. The search for a maximal matching of minimum cost can easily be done with the help of polynomial algorithms (see Chapter 2).

![Figure 6.9](https://example.com/fig6_9.png)

*Figure 6.9. Transformation of a network so that all nodes are the extremity of an even number of arcs: a) network $G$ in which we want to construct a tour. Nodes $c$, $h$, $i$, and $j$ are the extremity of an odd number of arcs. The arcs here all have a unitary length; b) auxiliary graph $G'$ in which are indicated all the lengths of the shortest paths between $c$, $h$, $i$, and $j$. The optimal matching is constituted by arcs $(c, i)$ and $(h, j)$; c) by adding the shortest path $(c, i)$ between $c$ and $i$, and $(h, k, j)$ between $h$ and $j$, we obtain a new network in which each node is the extremity of an even number of arcs.*
When all the arcs in the considered network $G$ are oriented, we can solve the Chinese postman problem by using an approach similar to the one described above. More precisely, if every node $v$ of the graph has as many arcs entering in $v$ as arcs leaving from $v$, we can then easily determine an oriented circuit in the graph, passing exactly once (in the right direction) through each arc in the graph. If not, we must duplicate certain arcs so that each node $v$ has as many arcs entering $v$ as arcs leaving $v$. The choice of arcs to duplicate is made with the help of the solution to a minimum cost flow problem (see chapter 2) in an auxiliary graph. More details on this algorithm are given for example in [EDM 73].

The algorithms presented up to here produce an optimal tour of the Chinese postman when no arc or all of them are oriented. In the mixed case, when certain arcs in the network are oriented while others are not, the Chinese postman problem becomes NP-hard, which means that there exists no algorithm today to solve the problem in polynomial time. Solutions of reasonably good quality can be obtained, for example, by adequately orienting the non oriented arcs and then duplicating certain arcs to obtain a totally oriented augmented graph, where each node $v$ has as many arcs entering $v$ as arcs leaving $v$. We are thus returning to the preceding problem that we know how to solve. More details on the manner of orienting the arcs as well as the choice of arcs to duplicate can be obtained by consulting for example [MIN 79].

### 6.3.2. The rural postman problem

Let $A$ denote the set of arcs of the considered network $G$. In the Chinese postman problem, one seeks a minimum cost tour that traverses all arcs of $A$ at least once. In many contexts, however, it is not necessary to traverse all arcs of the network, but to service only a subset $R \subseteq A$ of required arcs (also called clients), traversing if necessary some arcs of $A \setminus R$. When a required arc is traversed on a tour $T$, we say that $T$ covers this arc. A covering tour for $R$ is a tour that covers all arcs of $R$, which means that all arcs of $R$ are traversed at least once. When $R$ is a proper subset of $E$, the problem of finding a minimum cost covering tour for $R$ is known as the rural postman problem. It was introduced by Orloff in 1974 [ORL 74].

If the sub-network of $G$, composed uniquely of required arcs, is connected (that is, one can go from one client to another by using only service arcs), then the rural postman problem can be solved in polynomial time in cases where $G$ is non oriented or totally oriented. The ingredients to use are identical to those used for the Chinese postman problem. As an example, we give below the algorithm 6.7 for the non oriented case. This algorithm is illustrated in Figure 6.10.
**Algorithm for the non oriented rural postman problem where the clients form a connected sub-network**

1. Let $G_R$ be the partial network $G$ composed uniquely of required arcs. Determine the set $W$ of nodes having an odd number of neighbours in $G_R$. This set $W$ necessarily contains an even number of nodes. Construct a complete auxiliary graph $G'$ whose set of nodes is $W$ and whose cost associated to each arc $(v, w)$ is equal to the length of the shortest path between $v$ and $w$ in the original graph $G$.

2. Determine a maximal matching of minimum cost in $G'$. For each arc $(v, w)$ of the optimal matching, add in graph $G_R$ a copy of all the arcs belonging to a shortest path from $v$ to $w$. The nodes in the augmented graph now have an even number of neighbours.

3. Determine a tour by solving the Chinese postman problem in the augmented graph.

**Algorithm 6.7. Algorithm for the non oriented rural postman problem where the clients form a connected sub-network**

---

**Figure 6.10. Illustration of the algorithm for the non oriented rural postman problem in the case where the clients form a connected sub-network:**

a) network in which we want to construct a tour. All arcs correspond to clients except (k, i), (i, j), and (j, c). All arcs have unit length; b) partial network only containing required arcs. The nodes c, h, i, and j are the extremity of an odd number of arcs. The best matching among these 4 nodes consists in connecting h to i and j to c; c) augmented graph containing the required arcs and the matching. A rural postman tour is thus for example (a, b, c, i, k, h, j, f, i, j, c, d, e, f, g, h, a).

When the clients do not constitute a connected sub-network of the original graph, the problem is NP-hard (that is, no polynomial algorithm is known today to solve this problem). Frederickson [FRE 79] proposed an algorithm for the non oriented case. This algorithm guarantees that the length of the produced tour is in the worst case 50% superior to the optimum, assuming that the distances satisfy the triangular inequality. Frederickson’s algorithm is described in algorithm 6.8 and illustrated in Figure 6.11. Algorithms for the case where graph $G$ is totally oriented or mixed are described for example in [BAL 88], [CHR 86] and [HER 00b].
Algorithm for the non oriented rural postman problem

1. Let \( G_R \) be the partial network \( G \) composed uniquely of required arcs. Let \( C_1, C_2, \ldots, C_r \) be the connected components of network \( G_R \).
2. Construct a complete auxiliary graph \( G' \) whose set of nodes is \( \{1, 2, \ldots, r\} \) (a node for each connected component) and whose cost associated to each arc \((i, j)\) is equal to the length of the shortest path connecting a node of \( C_i \) to a node of \( C_j \) in \( G \).
3. Determine a maximal tree of minimum cost in \( G' \). For each arc \((i, j)\) of the optimal tree, add in graph \( G_R \) a copy of all the arcs belonging to a shortest path connecting a node of \( C_i \) to a node of \( C_j \) in \( G \).
4. Solve the rural postman problem in this connected augmented network with the help of algorithm 6.7.

Algorithm 6.8. Algorithm for the non oriented rural postman problem

A completely different approach is proposed in [HER 99] to solve the rural postman problem. We only describe here the non oriented case. All the ingredients that we will present can however easily be adapted to oriented and mixed cases.

Let's first of all consider a tour \( T \) covering a subset \( R' \) of required arcs. The procedure described in algorithm 6.9 tries to reduce the length of \( T \) without modifying the set of covered arcs. It is illustrated in Figure 6.12.
Procedure **Shorten**

1. Let $T$ be a tour covering a subset $R'$ of required arcs. Choose a direction for the tour $T$ and choose a departure node $v$ on $T$.

2. Determine a node $w$ on $T$ such that the length of the path linking $v$ to $w$ is as long as possible, while the path linking $w$ to $v$ covers all required arcs in $R'$. Let $P$ denote the path going from $v$ to $w$ in $T$.

3. Let $Q$ be the path connecting $w$ to $v$ in $T$, obtained by eliminating $P$ from $T$. If there exists on $Q$ a non-serviced arc $(z, w)$ entering $w$, then the first arcs of $Q$ to this arc $(z, w)$ induce a circuit $(w, ..., z, w)$; in such a case, inverse the orientation of all arcs in this circuit and return to 2. If not go to 4.

4. If the length of the shortest path from $v$ to $w$ in the original graph $G$ is inferior to the length of $P$, then replace $P$ by this shortest path.

5. Repeat steps 2 to 4 by considering the two possible orientations of $T$ and each possible departure node $v$ on $T$ until no further improvement can be obtained.

**Algorithm 6.9.** Procedure **Shorten** that permits to shorten a tour without modifying the set of serviced clients

---

**Figure 6.12.** Illustration of the procedure **Shorten**: a) network with a set of four clients to serve in heavy lines. All arcs have unit length; b) an oriented tour $T$ covering all required arcs. By choosing $v=c$ as departure node and by applying Step 2 of procedure **Shorten**, one can reach $w=e$ with $P=(c, d, e)$. The path from $e$ to $c$ obtained by removing $P$ from $T$ is $Q=(e, f, b, a, c, g, e, d, c)$. The arc $(a, e)$ of $Q$ is a non serviced arc entering $w=e$; c) new tour obtained by inverting the orientation of the circuit $(e, f, b, a, e)$ on $Q$. One can now reach $w=a$ from $v=c$ with $P=(c, d, e, a)$; d) shorter covering tour obtained by replacing the path $P=(c, d, e, a)$ of length 3 with the arc $(c, a)$ of length 1.
The second procedure described in algorithm 6.10 permits to add a client to an existing tour. This algorithm is illustrated in Figure 6.13.

**Procedure Add**

Let \( T \) be a tour covering a subset \( R' \) of clients. Let \((v, w) \notin R'\) be a client to add in \( T \).

1. If neither \( v \) nor \( w \) appears in \( T \), then determine the node \( z \) in \( T \) for which the sum of the distances to \( v \) and \( w \) is minimal, and add to \( T \) a cycle constituted of a shortest path from \( z \) to \( v \), the required arc \( (v, w) \), and a shortest path from \( w \) to \( z \).
   
   If not, if exactly one node among \( v \) and \( w \) appears in \( T \) (let’s say \( v \)), or if the two nodes \( v \) and \( w \) appear in \( T \) but not consecutively, add to \( T \) the cycle \((v, w, v)\).

2. Add \((v, w)\) to \( R' \) and try to diminish the length of the tour \( T \) with the help of the procedure *Shorten* described in algorithm 6.9 (by imposing a service on all arcs in the new set \( R' \)).

*Algorithm 6.10. Procedure Add that permits to add a client to a tour*

![Figure 6.13](image.png)

**Figure 6.13. Illustration of the first step of the procedure Add:** a) network with a set of clients to serve in heavy lines. All arcs have unit length; b) a tour \( T = (a, b, f, e, g, e, a) \) that covers the subset \( R' = \{(a, b), (e, f), (e, g)\} \) of clients; c) addition of client \( (c, d) \). Neither \( c \) nor \( d \) appears in \( T \). The node \( z \) in \( T \) for which the sum of the distances to \( c \) and \( d \) is minimal is node \( e \). The new tour is \( (a, b, f, e, g, e, d, c, d, e, a) \); d) addition of client \( (d, e) \), with node \( e \) on \( T \). The new tour is \( (a, b, f, e, g, e, d, e, a) \); e) addition of client \( (b, e) \) with nodes \( b \) and \( e \) on \( T \). The new tour is \( (a, b, c, b, f, e, g, e, a) \).
The third procedure is the inverse of the preceding one, and consists of removing
the service of a client on an existing tour. This procedure is described in algorithm
6.11 and is illustrated in Figure 6.14.

**Procedure Drop**

Let \( T \) be a tour covering a subset \( R' \) of clients. Let \((v, w) \in R'\) be a client to remove.

Remove \((v, w)\) from \( R' \) and try to diminish the length of tour \( T \) with the help of the
procedure Shorten described in algorithm 6.9 (by imposing a service only on the arcs in
the new set \( R' \)).

**Algorithm 6.11. Algorithm Drop that permits to remove a service on an arc of a tour**

A solution to the rural postman problem (not necessarily optimal) can easily be
obtained on the basis of the procedures described above. We can for example use the
constructive algorithm 6.12 proposed in [HER 99].

**Algorithm for the non oriented rural postman problem**

1. Choose a required arc \((v, w)\) in \( R \), and set \( T=(v, w, v) \) and \( R'=(v, w)\).
2. If \( R'=R \), then STOP. If not, choose a client \((v, w)\) belonging to \( R \) but not to \( R' \), add
   the service on this client \((v, w)\) with the help of the procedure Add, add \((v, w)\) to the
   set \( R' \) of clients already serviced, and repeat step 2.

**Algorithm 6.12. Algorithm for the non oriented rural postman problem**

### 6.3.3. Arc routing problems with capacity constraints

We again place ourselves in the context where the deliveries to make necessitate
the use of several vehicles, each vehicle having only a limited capacity (see Section
6.2.2). We then speak about an arc routing problem with capacity constraints. Most of the algorithms that have been developed to solve the ARP with capacity constraints have initially been described for the non oriented Chinese postman (see Section 6.3.1). In other words, these algorithms are applied to non oriented graphs where each arc corresponds to a client. These algorithms can however easily be adapted to other ARPs, for example to the oriented case where certain arcs in the network require no service.

Among the constructive methods proposed for the solution of the Chinese postman problem with capacity constraints, we only describe the Augment-Merge algorithm proposed in 1981 by Golden and Wong [GOL 81]. This algorithm is inspired by the one developed by Clarke and Wright [CLA 64] for the NRP (see Section 6.2.2). Starting with a set of tours, each one servicing only one client, the number of vehicles used is progressively reduced in two different ways. First of all, the service on certain arcs is transferred from one tour toward another. Then, new tours are obtained by merging existing tours.

Let’s consider two tours \( T_1 \) and \( T_2 \) so that the sum of the requests of their clients doesn’t exceed the capacity of a vehicle. Let’s note \( v_1 \) and \( v_2 \) the first extremities of arcs to service encountered on \( T_1 \) and \( T_2 \), respectively. Similarly, let \( w_1 \) and \( w_2 \) be the last extremities of arcs to service encountered on \( T_1 \) and \( T_2 \), respectively. Merging \( T_1 \) with \( T_2 \) can be done in four different ways. We can for example traverse \( T_1 \) up to node \( w_1 \), then go to \( v_2 \) by a shortest path, and finally traverse \( T_2 \) from \( v_2 \) to the depot. A second possibility consists for example to traverse \( T_1 \) up to \( w_1 \), then to go to \( w_2 \) with the help of a shortest path, and finally traverse \( T_2 \) in an inverse direction from \( w_2 \) to the depot. The two other possibilities are obtained by permuting the roles of \( v_1 \) and \( w_1 \), or of \( v_2 \) and \( w_2 \). The merging of two tours is illustrated in Figure 6.15, and the Augment-Merge algorithm is described in algorithm 6.13.

![Figure 6.15](image)

**Figure 6.15. Illustration of the merging of two tours:** a) two tours that we want to merge; b) the four possible merging.
Augment-Merge algorithm

1. For each required arc \((v, w)\), construct a tour \(T_{v,w}\) made of a shortest path of non-serviced arcs between the depot and \(v\), the serviced arc \((v, w)\), and a shortest path of non-serviced arcs between \(w\) and the depot.

2. Starting with the longest tour \(T_{v,w}\) and as long as vehicle capacity permits, change the status of traversed arcs, from non-serviced to serviced, if these arcs are covered by shorter vehicle routes. Remove the shorter vehicle routes whose unique serviced arc is now covered by a longer route.

3. Subject to capacity constraints, evaluate all possible mergers of two tours. If the largest saving obtained by one of these mergers is positive, do the merger which yields the largest positive saving, and repeat step 3, if not STOP.

Algorithm 6.13. Augment-Merge algorithm

Other constructive algorithms have been proposed for the solution of the ARP with capacity constraints. A complete review of these algorithms is given in [HER 00b] and [MIT 00].

Currently, the most efficient algorithms for the solution of ARPs with capacity constraints are adaptations of Local Search techniques (see chapter 3). Similar to what has been proposed for the NRP (see Section 6.2.2), a solution \(s\) is defined as a set of tours satisfying all the client requests, but not necessarily respecting the capacity constraints. If a solution \(s\) violates the capacity constraints, a penalty \(P(s)\) proportional to the excess demand with respect to vehicle capacity is calculated. The value \(F(s)\) of a solution \(s\) is then obtained by adding the total distance covered by the vehicles with the penalty \(P(s)\). The set \(N(s)\) of neighbour solutions to \(s\) contains all the solutions that can be obtained by transferring the service of a required arc from one tour toward another. To make such a transfer, it suffices to eliminate a service in a tour and add it in another. The elimination and addition of a service are done with the help of the procedures \(Add\) and \(Drop\) described in Section 6.3.2. Such an approach has been used with success in [HER 00a].

6.4. Conclusion

To transport raw materials, people, or information, to assure services at the clients, do maintenance work on the road networks or on the electric lines, etc., all these activities are the daily work of numerous companies. They can’t be competitive without good management of the resources allotted to these types of problems. Operational research has the techniques to treat certain aspects of these problems. The objective of this chapter was to describe the principal heuristic methods used to solve vehicle routing problems where the clients are located either on the nodes (problem NRP), or on the arcs (problem ARP) of a network. The reader interested in knowing more about the models and solution techniques of vehicle
routing problems is invited to consult the reference works [LAW 85] and [TOT 02] for the NRP and [DRO 00] for the ARP.

The variants of the NRP and the ARP that we have analysed in this chapter are basic vehicle routing problems. When we must treat real routing problems, additional constraints must be taken into account (limit on length or duration of tours, different categories of routes, multiple depots, etc.), and the algorithms that we have described must then be adapted or extended since they cannot be directly applied to such problems. The algorithms described in this chapter must be considered as skeletons of more specialised algorithms to be designed for each particular real life problem.

6.5. Bibliography


arc routing problem, 1
Augment-Merge algorithm, 19
capacity constraints, 5, 18
Cheapest Insertion algorithm, 3
Chinese postman problem, 10
Christofides’ algorithm, 4
Clarke et Wright’s algorithm, 6
covering tour, 13
Frederickson’s algorithm, 14
Local Search techniques, 5, 8, 20
maximal matching, 4, 11
maximal tree, 4, 14
minimum cost flow, 12
mixed graph, 10
Nearest Neighbour algorithm, 2
node routing problem, 1
NP-hard, 2, 10, 13, 14
required arcs, 13
rural postman problem, 13
saving, 6
traveling salesman problem, 2
triangular inequality, 4, 14
two-phase methods, 7