9

Recent Trends in Arc Routing

Alain Hertz

Ecole Polytechnique - GERAD
Département de Mathématiques et de génie industriel
CP 6079, succ. Centre-ville
Montréal (QC) H3C 3A7, Canada

Abstract

Arc routing problems (ARPs) arise naturally in several applications where streets require maintenance, or customers located along road must be serviced. The undirected rural postman problem (URPP) is to determine a least cost tour traversing at least once each edge that requires a service. When demands are put on the edges and this total demand must be covered by a fleet of identical vehicles of capacity Q based at a depot, one gets the undirected capacitated arc routing problem (UCARP). The URPP and UCARP are known to be NP-hard. This chapter reports on recent exact and heuristic algorithms for the URPP and UCARP.

1. Introduction

Arc routing problems (ARPs) arise naturally in several applications related to garbage collection, road gritting, mail delivery, network maintenance, snow clearing, etc. [19, 1, 14]. ARPs are defined over a graph $G = (V, E \cup A)$, where V is the vertex set, E is the edge set, and A is the arc set. A graph G is called *directed* if E is empty, *undirected* if A is empty, and *mixed* if both E and A are non-empty. In this chapter, we consider only undirected ARPs. The traversal *cost* (also called *length*) c_{ij} of an edge (v_i,v_j) in E is supposed to be non-negative. A *tour* T, or *cycle* in G is represented by a vector of the form $(v_1,v_2,\ldots,v_n)$ where (v_i,v_{i+1}) belongs to E for $i = 1,\ldots, n-1$ and $v_n = v_1$. All graph theoretical terms not defined here can be found in [2].

In the *Undirected Chinese Postman Problem*, one seeks a minimum cost tour that traverses all edges of E at least once. In many contexts, however, it is not necessary to *traverse* all edges of E, but to *service* or *cover* only a subset $R \subseteq E$ of *required* edges, traversing if necessary some edges of $E\backslash R$. A *covering tour* for R is a tour that traverses all edges of R 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 *Undirected Rural Postman*

Problem (URPP). Assume for example that a city's electric company periodically has to send electric meter readers to record the consumption of electricity by the different households for billing purposes. Suppose that the company has already decided who will read each household's meter. This means that each meter reader has to traverse a given subset of city streets. In order to plan the route of each meter reader, it is convenient and natural to represent the problem as a URPP in which the nodes of the graph are the street intersections while the edges of the graph are the street segments between intersections, some of them requiring meter readings.

Extensions of these classical problems are obtained by imposing capacity constraints. The *Undirected Capacitated Arc Routing Problem* (UCARP) is a generalization of the URPP in which m identical vehicles are available, each of capacity Q. One particular vertex is called the *depot* and each required edge has an integral non-negative *demand*. A vehicle route *is feasible* if it contains the depot and if the total demand on the edges covered by the vehicle does not exceed the capacity Q. The task is to find a set of m feasible vehicle routes of minimum cost such that each required edge is serviced by exactly one vehicle. The number m of vehicles may be given a priori or can be a decision variable. As an example, consider again the above problem of the city's electric company, but assume this time that the subset of streets that each meter reader has to visit is not fixed in advance. Moreover, assume that no meter reader can work more than a given number of hours. The problem to be solved is then a UCARP in which one has to build a route for each meter reader so that all household's meters are scanned while no meter reader is assigned a route which exceeds the specified number of work hours.

The URPP was introduced by Orloff [40] and shown to be NP-hard by Lenstra and Rinnooy Kan [35]. The UCARP is also NP-hard since the URPP reduces to it whenever Q is greater than or equal to the total demand on the required edges. Even finding a 0.5 approximation to the UCARP is NP-hard, as shown by Golden and Wong [27]. The purpose of this chapter is to survey some recent algorithmic developments for the URPP and UCARP. The algorithms described in this chapter should be considered as skeletons of more specialized algorithms to be designed for real life problems which typically have additional constraints. For example, it can be imposed that the edges must be serviced in an order that respects a given precedence relation [15]. Also, real life problems can have multiple depot locations [17] and time windows or time limits on the routes [18, 44]. Arc routing applications are described in details in chapters 10, 11 and 12 of the book edited by Dror [14].

In the next section, we give some additional notations, and we describe a reduction that will allow us to assume that all vertices are incident to at least one required edge. We also briefly describe some powerful general solution methods for integer programming problems. Section 3 contains recent exact methods for the URPP. Then in Section 4, we describe some recent basic procedures that can be used for the design of heuristic methods for the URPP and UCARP. Section 5 contains examples of recent effective algorithms that use the basic procedures of Section 4 as main ingredients.

2. Preliminaries

Let V_R denote the set of vertices incident to at least one edge in R. The *required subgraph* $G_R(V_R,R)$ is defined as the partial subgraph of G induced by R. It is obtained from G by removing all non-required edges as well as all vertices that are not incident to any required edge. Let C_i $(i=1,\ldots,p)$ be the i-th connected component of $G_R(V_R,R)$, and let $V_i \subseteq V_R$ be the set of vertices of C_i. [9] have designed a pre-processing procedure which converts any URPP instance into another instance for which $V = V_R$, (i.e. each vertex is incident with at least one required edge). This is done as follows. An edge (v_i,v_j) is first included in $G_R(V_R,R)$ for each v_i,v_j in V_R, with cost c_{ij} equal to the length of a shortest chain between v_i and v_j in G. This set of new edges added to $G_R(V_R,R)$ is then reduced by eliminating

(a) all new edges (v_i,v_j) for which $c_{ij} = c_{ik} + c_{kj}$ for some v_k in V_R, and
(b) one of two parallel edges if they have the same cost.

To illustrate, consider the graph G shown in Figure 1(a), where edges of R are shown in bold lines and numbers correspond to edge costs. The new instance with $V = V_R$ is represented in Figure 1(b).

From now on we will assume that the URPP is defined on a graph $G = (V,E)$ to which the pre-processing procedure has already been applied.

To understand the developments of Section 3, the reader has to be familiar with basic concepts in *Integer Programming*. If needed, a good introduction to this topic can be found in [45]. We briefly describe here below the *cutting plane algorithm* and the *Branch & Cut* methodology which are currently the most powerful exact solution approaches for arc routing problems.

Most arc routing problems can be formulated in the form

$$\begin{cases} \text{Minimize } \sum_{e\in E} c_e x_e \\ \text{subject to } \ x \in S \end{cases}$$

where S is a set of *feasible solutions*. The convex hull *conv(S)* of the vectors in S is a polyhedron with integral extreme points. Since any polyhedron can be described by a set of linear inequalities, one can theoretically solve the above problem by Linear

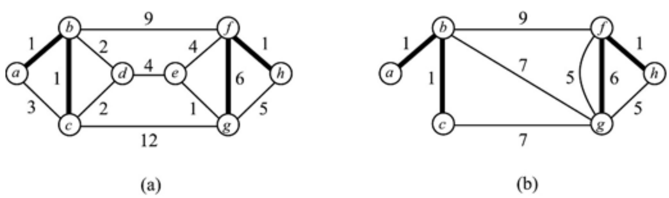

Figure 1. Illustration of the pre-processing procedure.

Programming (LP). Unfortunately, a complete linear description of *conv(S)* typically contains a number of inequalities which is exponential in the size of the original problem. To circumvent this problem, one can start the optimization process with a small subset of known inequalities and compute the optimal LP solution subject to these constraints. One can then try to identify an inequality that is valid for *conv(S)* but violated by the current LP solution. Such an inequality is called a *cutting plane*, because, geometrically speaking, it "cuts off" the current LP solution. If such a cutting plane is found, then it is added to the current LP and the process is repeated. Otherwise, the current LP solution is the optimal solution of the original problem. This kind of procedure is called the *cutting plane algorithm*. It originated in the pioneering work of Dantzig, Fulkerson and Johnson [12] on the Symmetric Traveling Salesman Problem.

The problem consisting in either finding a cutting plane, or proving that no such inequality exists is known as the *separation problem*. An algorithm that solves it is called an *exact separation algorithm*. The separation problem can however be NP-hard for some classes of inequalities. In such a case, one has to resort to a *heuristic separation algorithm* that may fail to find a violated inequality in the considered class, even if one exists.

The cutting plane algorithm stops when no more valid inequality can be found. This however does not mean that no such inequality exists. It may be that a violated inequality belongs to an unknown class of inequalities, or that it belongs to a known class for which we have used, without success, a heuristic separation problem. When the cutting plane algorithm fails to solve a given instance, one can choose among several options. One option is to feed the current LP solution into a classical Branch & Bound algorithm for integer programs. A more powerful option is to use the so-called *Branch & Cut* method (see for example [42]). A Branch & Cut is much like a Branch & Bound method except for the fact that valid inequalities may be added at any node of the branching tree. This leads to stronger linear relaxations at any node, which normally leads in turn to a considerable reduction in the number of nodes, in comparison with standard Branch & Bound.

3. Exact Algorithms for the URPP

A connected graph is said to be *Eulerian* if each vertex has an even degree. It is well known that finding a tour in an Eulerian graph that traverses each edge exactly once is an easy problem that can be solved, for example, by means of the $O(|E|)$ algorithm described by Edmonds and Johnson [16]. Hence, the URPP is equivalent to determining a least cost set of additional edges that, along with the required edges, makes up an Eulerian subgraph. Let x_e denote the number of copies of edge e that must be added to R in order to obtain an Eulerian graph, and let $G(x)$ denote the resulting Eulerian graph.

For a subset $W \subseteq V$ of vertices, we denote $\delta(W)$ the set of edges of E with one endpoint in W and the other in $V \setminus W$. If W contains only one vertex v, we simply write

$\delta(v)$ instead of $\delta(\{v\})$. [9] have proposed the following integer programming formulation for the URPP:

$$\text{Minimize} \sum_{e \in E} c_e x_e$$

subject to

$$|R \cap \delta(v_i)| + \sum_{e \in \delta(v_i)} x_e = 2z_i \qquad (v_i \in V) \tag{1}$$

$$\sum_{e \in \delta(W)} x_e \geq 2 \qquad (W = \bigcup_{k \in P} V_k,\ P \subset \{1, \ldots, p\},\ P \neq \emptyset) \tag{2}$$

$$x_e \geq 0 \text{ and integer} \qquad (e \in E) \tag{3}$$

$$z_i \geq 0 \text{ and integer} \qquad (v_i \in V) \tag{4}$$

Constraints (1) stipulate that each vertex in $G(x)$ must have an even degree. Indeed, the left-hand side of the equality is the total number of edges incident to v_i in $G(x)$, while the right-hand side is an even integer.

Constraints (2) enforce the solution to be connected. To understand this point, remember first that $G_R(V, R)$ contains p connected components with vertex sets $V_1, \ldots, V_p$. Now, let P be a non-empty proper subset of $\{1, \ldots, p\}$ and consider the vertex set $W = \bigcup_{k \in P} V_k$. Notice that no required edge has one endpoint in W and the other one outside W. In order to service not only the required edges with both endpoints in W, but also those with both endpoints in $V \setminus W$, a tour must traverse the frontier between W and $V \setminus W$ at least twice. Hence $\sum_{e \in \delta(W)} x_e$ must be at least equal to 2.

The associated polyhedron was not examined in detail by [9]. This was done in [10] who proposed the following formulation that avoids variables z_i and where $\delta_R(W) = R \cap \delta(W)$.

$$\text{Minimize} \sum_{e \in E} c_e x_e$$

subject to

$$\sum_{e \in \delta(v)} x_e = |\delta_R(v)| \pmod 2 \qquad (v \in V) \tag{5}$$

$$\sum_{e \in \delta(W)} x_e \geq 2 \qquad (W = \bigcup_{k \in P} V_k,\ P \subset \{1, \ldots, p\},\ P \neq \emptyset) \tag{2}$$

$$x_e \geq 0 \text{ and integer} \qquad (e \in E) \tag{3}$$

The convex hull of feasible solutions to (2), (3), (5) is an unbounded polyhedron. The main difficulty with this formulation lies with the non-linear degree constraints (5).

Another approach has recently been proposed by Ghiani and Laporte [24]. They use the same formulation as Corberán and Sanchis, but they noted that only a small set of variables may be greater than 1 in an optimal solution of the RPP and, furthermore, these variables can take at most a value of 2. Then, by duplicating these latter variables, Ghiani and Laporte formulate the URPP using only 0/1 variables. More precisely, they base their developments on *dominance relations* which are equalities or inequalities that reduce the set of feasible solutions to a smaller set which surely contains an optimal solution. Hence, a dominance relation is satisfied by at least one optimal solution of the problem but not necessarily by all feasible solutions. While some of these domination relations are difficult to prove, they are easy to formulate. For example, [9] have proved the following domination relation.

Domination relation 1
Every optimal solution of the URPP satisfies the following relations:

$$\begin{aligned} x_e &\leq 1 \quad \text{if } e \in R \\ x_e &\leq 2 \quad \text{if } e \in E \setminus R \end{aligned}$$

This domination relation indicates that given any optimal solution x^* of the URPP, all edges appear at most twice in $G(x^*)$. This means that one can restrict our attention to those feasible solutions obtained by adding at most one copy of each required edge, and at most two copies of each non-required one. The following second domination relation was proved by Corberán and Sanchis [10].

Domination relation 2
Every optimal solution of the URPP satisfies the following relation:

$x_e \leq 1$ if e is an edge linking two vertices in the same connected component of G_R

The above domination relation not only states (as the first one) that it is not necessary to add more that one copy of each required edge (i.e., $x_e \leq 1$ if $e \in R$), but also that it is not necessary to add more than one copy of each non-required edge linking two vertices in the same connected component of G_R. Another domination relation is given in [24].

Domination relation 3
Let G^* be an auxiliary graph having a vertex w_i for each connected component C_i of G_R and, for each pair of components C_i and C_j, an edge (w_i,w_j) corresponding to a least cost edge between C_i and C_j. Every optimal solution of the URPP satisfies the following relation:

$x_e \leq 1$ if e does not belong to a minimum spanning tree on G^*.

Let E_2 denote the set of edges belonging to a minimum spanning tree on G^*, and let $E_1 = E \setminus E_2$. The above relation, combined with domination relation 1 proves that given any optimal solution x^* of the URPP, the graph $G(x^*)$ is obtained from G_R by adding at most one copy of each edge in E_1, and at most two copies of each edge in E_2. In summary, every optimal solution of the URPP satisfies the following relations:

$$x_e \leq 1 \quad \text{if} \quad e \in E_1$$
$$x_e \leq 2 \quad \text{if} \quad e \in E_2$$

Ghiani and Laporte propose to replace each edge $e \in E_2$ by two parallel edges e' and e''. Doing this, they replace variable x_e that can take values 0, 1 and 2 by two binary variables $x_{e'}$ and $x_{e''}$. Let E' and E'' be the set of edges e' and e'' and let $E^* = E_1 \cup E' \cup E''$. The URPP can now be formulated as a binary integer program:

$$\text{Minimize} \sum_{e \in E^*} c_e x_e$$

subject to

$$\sum_{e \in \delta(v)} x_e = |\delta_R(v)| \pmod 2 \qquad (v \in V) \tag{5}$$

$$\sum_{e \in \delta(W)} x_e \geq 2 \qquad (W = \bigcup_{k \in P} V_k, P \subset \{1, \ldots, p\}, P \neq \emptyset) \tag{2}$$

$$x_e = 0 \; or \; 1 \qquad (e \in E^*) \tag{6}$$

The convex hull of feasible solutions to (2), (5), (6) is a polytope (i.e., a bounded polyhedron). The *cocircuits inequalities*, defined by Barahona and Grötschel [3] and described here below, are valid inequalities for this new formulation, while they are not valid for the unbounded polyhedron induced by the previous formulations. These inequalities can be written as follows:

$$\sum_{e \in \delta(v) \setminus F} x_e \geq \sum_{e \in F} x_e - |F| + 1 \qquad (v \in V,\ F \subseteq \delta(v), |\delta_R(v)| + |F| \text{ is odd}) \tag{7}$$

To understand these inequalities, consider any vertex v and any subset $F \subseteq \delta(v)$ of edges incident to v, and assume first that there is at least one edge $e \in F$ with $x_e = 0$ (i.e., no copy of e is added to $G_R(V,R)$ to obtain $G(x)$). Then $\sum_{e \in F} x_e - |F| + 1 \leq 0$ and constraints (7) are useless in that case since we already know from constraints (6) that $\sum_{e \in \delta(v) \setminus F} x_e$ must be greater than or equal to zero. So suppose now that $G(x)$ contains a copy of each edge $e \in F$. Then vertex v is incident in $G(x)$ to $|\delta_R(v)|$ required edges and to $|F|$ copies of edges added to $G_R(V,R)$. If $|\delta_R(v)| + |F|$ is odd then at least one additional edge in $\delta(v) \setminus F$ must be added to $G_R(V,R)$ in order to get the desired Eulerian graph $G(x)$. This is exactly what is required by constraints (7) since, in that case, $\sum_{e \in F} x_e - |F| + 1 = 1$. Ghiani and Laporte have shown that the non-linear constraints (5) can be replaced by the linear constraints (7), and they therefore propose the following binary linear formulation to the URPP:

$$\text{Minimize} \sum_{e \in E^*} c_e x_e$$

subject to

$$\sum_{e \in \delta(v) \setminus F} x_e \geq \sum_{e \in F} x_e - |F| + 1 \qquad (v \in V,\ F \subseteq \delta\ (v), |\delta_R(v)| + |F| \text{ is odd}) \quad (7)$$

$$\sum_{e \in \delta(W)} x_e \geq 2 \qquad (W = \bigcup_{k \in P} V_k,\ P \subset \{1, \ldots, p\},\ P \neq \emptyset) \quad (2)$$

$$x_e = 0 \text{ or } 1 \qquad (e \in E^*) \quad (6)$$

All constraints in the above formulation are linear, and this makes the use of Branch & Cut algorithms easier (see Section 2). Cocircuit inequalities (7) can be generalized to any non-empty subset W of V:

$$\sum_{e \in \delta(W) \setminus F} x_e \geq \sum_{e \in F} x_e - |F| + 1 \qquad (F \subseteq \delta(W), |\delta_R(W)| + |F| \text{ is odd}) \quad (8)$$

If $\delta_R(W)$ is odd and F is empty, constraints (8) reduce to the following *R-odd inequalities* used by Corberán and Sanchis [10]:

$$\sum_{e \in \delta(W)} x_e \geq 1 \qquad (W \subset V, |\delta_R(W)| \text{ is odd}) \quad (9)$$

If $\delta_R(W)$ is even and F contains one edge, constraints (8) reduce to the following *R-even inequalities* defined by Ghiani and Laporte [24]:

$$\sum_{e \in \delta(W) \setminus \{e^*\}} x_e \geq x_{e^*} \qquad (W \neq \emptyset,\ W \subset V, |\delta_R(W)| \text{ is even},\ e^* \in \delta(W)) \quad (10)$$

These *R-even inequalities* (10) can be explained as follows. Notice first that they are useless when $x_{e^*} = 0$ since we already know from constraints (6) that $\sum_{e \in \delta(W) \setminus \{e^*\}} x_e \geq 0$. So let W be any non-empty proper subset of V such that $|\delta_R(W)|$ is even, and let e^* be any edge in $\delta(W)$ with $x_{e^*} = 1$ (if any). Since $G(x)$ is required to be Eulerian, the number of edges in $G(x)$ having one endpoint in W and the other outside W must be even. These edges that traverse the frontier between W and $V \setminus W$ in $G(x)$ are those in $\delta_R(W)$ as well as the edges $e \in \delta(W)$ with value $x_e = 1$. Since $|\delta_R(W)|$ is supposed to be even and $x_{e^*} = 1$, we can impose $\sum_{e \in \delta(W)} x_e = \sum_{e \in \delta(W) \setminus \{e^*\}} x_e + 1$ to also be even, which means that $\sum_{e \in \delta(W) \setminus \{e^*\}} x_e$ must be greater than or equal to $1 = x_{e^*}$.

Several researchers have implemented cutting plane and Branch & Cut algorithms for the URPP, based on the above formulations. It turns out that cutting planes of type (9) and (10) are easier to generate than the more general ones of type (7) or (8). Ghiani and Laporte have implemented a Branch & Cut algorithm for the URPP, based on connectivity inequalities (2), on R-odd inequalities (9) and on R-even inequalities (10). The separation problem (see Section 2) for connectivity inequalities is solved by means of a heuristic proposed by Fischetti, Salazar and Toth [21]. To separate R-odd inequalities, they use a heuristic inspired by a procedure developed by Grötschel and Win [28]. The exact separation algorithm of [41] could be used to identify violated R-even inequalities, but [24] have developed a faster heuristic procedure that detects several violations at a time. They report very good computational results on a set of 200 instances, corresponding to three classes of random graphs generated as in [31]. Except for 6 instances, the other 194 instances involving up to 350 vertices were solved to optimality in a reasonable amount of time. These results outperform those reported by Christofides, et al. [9], [10] and [36] who solved much smaller randomly generated instances ($|V| \leq 84$).

4. Basic Procedures for the URPP and the UCARP

Up to recently, the best known constructive heuristic for the URPP was due to [22]. This method works along the lines of Christofides's algorithm [8] for the undirected traveling salesman problem, and can be described as follows.

Frederickson's Algorithm

Step 1. Construct a minimum spanning tree S over G^* (see domination relation 3 in Section 3 for the definition of G^*).
Step 2. Determine a minimum cost matching M (with respect to shortest chain costs) on the odd-degree vertices of the graph induced by $R \cup S$.
Step 3. Determine an Eulerian tour in the graph induced by $R \cup S \cup M$.

As Christofides's algorithm for the undirected traveling salesman, the above algorithm has a worst case ratio of 3/2. Indeed, let C^* be the optimal value of the URPP and let C_R, C_S and C_M denote the total cost of the edges in R, S and M, respectively. It is not difficult to show that $C_R + C_S \leq C^*$ and $C_M \leq C^*/2$, and this implies that $C_R + C_S + C_M \leq 3C^*/2$.

Two recent articles [31, 30] contain a description of some basic algorithmic procedures for the design of heuristic methods in an arc routing context. All these procedures are briefly described and illustrated in this section. In what follows, SC_{vw} denotes the shortest chain linking vertex v to vertex w while L_{vw} is the length of this chain. The first procedures, called POSTPONE and REVERSE, modify the order in which edges are serviced or traversed without being serviced.

Procedure Postpone

INPUT : a covering tour T with a given orientation and a starting point v on T.
OUTPUT : another covering tour.

Whenever a required edge e appears several times on T, delay service of e until its last occurrence on T, considering v as starting vertex on T.

Procedure Reverse

INPUT : a covering tour T with a given orientation and a starting point v on T.
OUTPUT : another covering tour.

Step 1. Determine a vertex w on T such that the path linking v to w on T is as long as possible, while the path linking w to v contains all edges in R. Let P denote the path on T from v to w and P' the path from w to v.
Step 2. If P' contains an edge (x,w) entering w which is traversed but not serviced, then the first edges on P' up to (x,w) induce a circuit C. Reverse the orientation of C and go to Step 1.

The next procedure, called Shorten, is based on the simple observation that if a tour Tcontains a chain P of traversed (but not serviced) edges, then T can eventually be shortened by replacing P by a shortest chain linking the endpoints of P.

Procedure Shorten

INPUT : a covering tour T
OUTPUT : a possibly shorter covering tour.

Step 1. Choose an orientation for T and let v be any vertex on T.
Step 2. Apply Postpone and Reverse
Step 3. Let w be the first vertex on T preceeding a required edge. If L_{vw} is shorter than the length of P, then replace P by SC_{vw}.
Step 4. Repeatedly apply steps 2 and 3, considering the two possible orientations of T, and each possible starting vertex v on T, until no improvement can be obtained.

As an illustration, consider the graph depicted in Figure 2(a) containing 4 required edges shown in bold lines. An oriented tour $T = (c,d,e,f,b,a,e,g,d,c)$ is represented in Figure 2(b) with $v = c$ as starting vertex. Since the required edge (c,d) appears twice on T, Postpone makes it first traversed and then serviced, as shown on Figure 2(c). Then, Reverse determines $P = (c,d,e)$ and $P' = (e,f,b,a,e,g,d,c)$, and since P' contains a non-serviced edge (a,e) entering e, the orientation of the circuit (e,f,b,a,e) is reversed, yielding the new tour represented in Figure 2(d). The first part $P = (c,d,e,a)$ of this tour is shortened into (c,a), yielding the tour depicted in Figure 2(e).

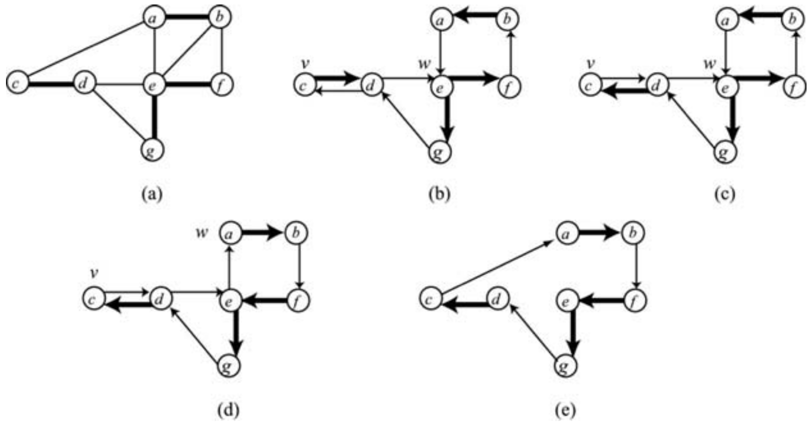

Figure 2. Illustration of procedures POSTPONE, REVERSE and SHORTEN.

The next procedure, called SWITCH, also modifies the order in which required edges are visited on a given tour. It is illustrated in Figure 3.

Procedure SWITCH

INPUT : a covering tour T
OUTPUT : another covering tour.

Step 1. Select a vertex v appearing several times on T.
Step 2. Reverse all minimal cycles starting and ending at v on T.

Given a covering tour T and given a non-required edge (v,w), procedure ADD builds a new tour covering $R \cup (v,w)$. On the contrary, given a required edge (v,w) in R, procedure DROP builds a new tour covering $R \backslash (v,w)$.

Procedure ADD

INPUT : a covering tour T and an edge $(v,w) \notin R$
OUTPUT : a covering tour for $R \cup (v,w)$

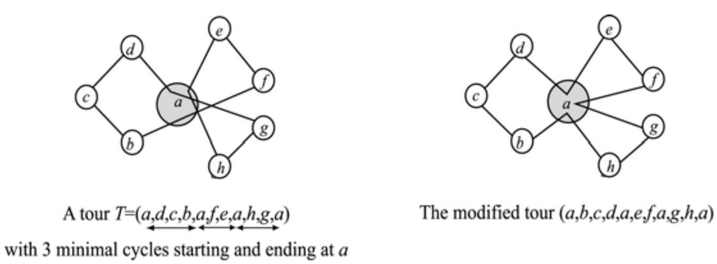

Figure 3. Illustration of procedure SWITCH.

Step 1. If neither v nor w appear on T, then determine a vertex z on T minimizing $L_{zv} + L_{wz}$, and add the circuit $SC_{zv} \cup (v,w) \cup SC_{wz}$ on T. Otherwise, if one of v and w (say v), or both of them appear on T, but not consecutively, then add the circuit (v,w,v) on T.
Step 2. Set $R := R \cup (v,w)$ and attempt to shorten T by means of SHORTEN.

Procedure DROP

INPUT : a covering tour T and an edge (v,w) in R
OUTPUT : a covering tour for $R\backslash(v,w)$.

Step 1. Set $R := \mathrm{R}\backslash(v,w)$.
Step 2. Attempt to shorten T by means of SHORTEN.

The last two procedures, called PASTE and CUT can be used in a UCARP context. PASTE merges two routes into a single tour, possibly infeasible for the UCARP.

Procedure PASTE

INPUT : two routes $T_1 =$ (depot, $v_1, v_2, \ldots, v_r$, depot) and $T_2 =$ (depot, $w_1, w_2, \ldots, w_s$, depot).
OUTPUT : a single route T containing all required edges of R_1 and R_2.

If (v_r, depot) and (depot, w_1) are non-serviced edges on T_1 and T_2, respectively, then set $T =$ (depot, $v_1, \ldots, v_r, w_1, \ldots, w_s$, depot), else set $T =$ (depot, $v_1, \ldots, v_r$, depot, $w_1, \ldots, w_s$, depot).

CUT decomposes a non-feasible route into a set of feasible routes (i.e., the total demand on each route does not exceed the capacity Q of each vehicle).

Procedure CUT

INPUT : A route T starting and ending at the depot, and covering R.
OUTPUT : a set of feasible vehicle routes covering R.

Step 0. Label the vertices on T so that $T =$ (depot, $v_1, v_2, \ldots, v_t$, depot).
Step 1. Let D denote the total demand on T. If $D \leq Q$ then STOP : T is a feasible vehicle route.
Step 2. Determine the largest index s such that (v_{s-1}, v_s) is a serviced edge, and the total demand on the path (depot, $v_1, \ldots, v_s$) from the depot to v_s does not exceed Q. Determine the smallest index r such that (v_{r-1}, v_r) is a serviced edge and the total demand on the path ($v_r, \ldots, v_t$, depot) from v_r to the depot does not exceed $Q(\lceil D/Q \rceil - 1)$. If $r > s$ then set $r = s$.

For each index q such that $r \leq q \leq s$, let $v_q{*}$ denote the first endpoint of a required edge after v_q on T, and let δ_q denote the length of the chain linking v_q to $v_q{*}$ on T. Select the vertex v_q minimizing $L(v_q) = L_{v_q,\,\text{depot}} + L_{\text{depot},\, v_q*} - \delta_q$.

Step 3. Let P_{v_q} (P_{v_q*}) denote the paths on T from the depot to v_q (v_q*). Construct the feasible vehicle route made of $P_{v_q} \cup SC_{v_q,\text{depot}}$, replace P_{v_q*} by SC_{depot,v_q*} on T and return to Step 1.

The main idea of the above algorithm is to try to decompose the non-feasible route T into $\lceil D/Q \rceil$ feasible vehicle routes, where $\lceil D/Q \rceil$ is a trivial lower bound on the number of vehicles needed to satisfy the demand on T. If such a decomposition exists, then the demand covered by the first vehicle must be large enough so that the residual demand for the $\lceil D/Q \rceil - 1$ other vehicles does not exceed $Q(\lceil D/Q \rceil - 1)$ units: this constraint defines the above vertex v_r. The first vehicle can however not service more than Q units, and this defines the above vertex v_s. If $r > s$, this means that it is not possible to satisfy the demand with $\lceil D/Q \rceil$ vehicle routes, and the strategy described above is to cover as many required edges as possible with the first vehicle. Otherwise, the first vehicle satisfies the demand up to a vertex v_q on the path linking v_r to v_s, and the process is then repeated on the tour T'obtained from T by replacing the path (depot, $v_{1,\ldots,}v_q$*) by a shortest path from the depot to v_q*. The choice for v_q is made so that the length of T' plus the length of the first vehicle route is minimized.

Procedure Cut is illustrated in Figure 4. The numbers in square boxes are demands on required edges. The numbers on the dashed lines or on the edges are shortest chain lengths. In this example, $Q = 11$ and $D = 24$. The procedure first computes $Q(\lceil D/Q \rceil - 1) = 22$, which implies that the first vehicle route must include at least the first required edge (i.e., $r = 2$). Since the first vehicle cannot include more than the three first required edges without having a weight exceeding Q, we have $s = 5$. Now, $v_2* = v_3$, $v_3* = v_3$, $v_4* = v_4$ and $v_5* = v_6$, and since $L(v_2) = 10$, $L(v_3) = 12$, $L(v_4) = 8$ and $L(v_5) = 11$, vertex v_4 is selected. The first vehicle route is therefore equal to (depot,v_1,v_2,v_3,v_4,depot) and the procedure is reapplied on the tour (depot, $v_4, v_5, \ldots, v_{10}$, depot) with a total demand $D = 17$. We now have $s = 7$ and $r = 9$, which means that the four remaining required edges cannot be serviced by two vehicles. We therefore set $s = r = 7$, which means that the second vehicle route is equal to (depot, v_4, v_5, v_6, v_7, depot) and the procedure is repeated on $T =$ (depot, v_8 ,v_9 ,v_{10}, depot) with $D = 12$. Since $s = r = 9$, the third vehicle route is equal to (depot, v_8, v_9, depot) and the residual tour $T =$ (depot, v_9, v_{10}, depot) is now feasible and corresponds to the fourth vehicle route.

Notice that procedure Cut does not necessarily produce a solution with a minimum number of vehicle routes. Indeed, in the above example, the initial route

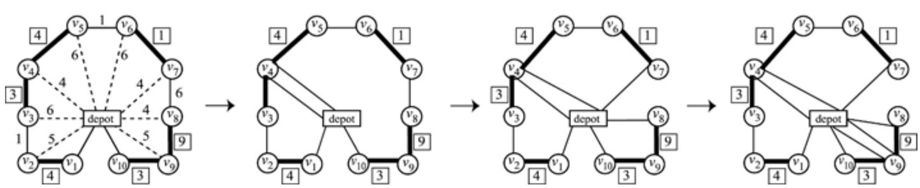

Figure 4. Illustration of procedure Cut.

T has been decomposed into four vehicle routes while there exists a solution with three vehicle routes (depot, $v_1, \ldots, v_5$, depot), (depot, $v_6, \ldots, v_9$, depot) and (depot, v_9, v_{10}, depot).

5. Recent Heuristic Algorithms for the URPP and the UCARP

The procedures described in the previous section can be used as basic tools for the design of constructive algorithms for the URPP. As an example, a solution to the URPP can easily be obtained by means of the following very simple algorithm designed by [31].

Algorithm Construct-URPP

Step 1. Choose a required edge (v_i,v_j) and set $T = (v_i,v_j,v_i)$.
Step 2. If T contains all required edges then stop; else chose a required edge which is not yet in T and add it to $T$by means of procedure ADD.

Post-optimization procedures can be designed on the basis of procedures DROP, ADD and SHORTEN. As an example, an algorithm similar to the 2-opt procedure [11] for the undirected traveling salesman problem can be designed for the URPP as shown below.

Algorithm 2-opt-URPP

Step 1. Choose an orientation of the given tour T and select two arcs (v_i,v_j) and (v_r,v_s) on T. Build a new tour T' by replacing these two arcs by the shortest chains SP_{ir} and SP_{js}, and by reversing the orientation of the path linking v_j to v_r on T.
Step 2. Let R' be the set of required edges appearing on T'. Apply SHORTEN to determine a possibly shorter tour T'' that also covers R'. If $R \neq R'$ then add the missing required edges on T'' by means of procedure ADD.
Step 3. If the resulting tour T'' has a lower cost than T, then set T equal to T''.
Step 4. Repeat steps 1, 2 and 3 with the two possible orientations of T and with all possible choices for (v_i,v_j) and (v_r,v_s), until no additional improvement can be obtained.

[31] propose to use a post-optimization procedure, called DROP-ADD, similar to the Unstringing-Stringing (US) algorithm for the undirected traveling salesman problem [23]. DROP-ADD tries to improve a given tour by removing a required edge and reinserting it by means of DROP and ADD, respectively.

Algorithm DROP-ADD

Step 1. Choose a required edge e, and build a tour T' covering $R\backslash\{e\}$ by means of DROP.

Step 2. If edge e is not traversed on T', then add e to T' by means of ADD.
Step 3. If the resulting tour T' has a lower cost than T, then set T equal to T'.
Step 4. Repeat steps 1, 2 and 3 with all possible choices for e, until no additional improvement can be obtained.

[31] have generated 92 URPP instances to test the performance of these two post-optimization procedures. These 92 instances correspond to three classes of randomly generated graphs. First class graphs are obtained by randomly generating points in the plane; class 2 graphs are grid graphs generated to represent the topography of cities, while class 3 contains grid graphs with vertex degrees equal to 4. Computational experiments show that Frederickson's algorithm is always very quick but rarely optimal. Percentage gaps with respect to best known solutions can be as large as 10%, particularly in the case of larger instances or when the number of connected components in G_R is large. Applying DROP-ADD after Frederickson's algorithm typically generates a significant improvement within a very short computing time. However, much better results are obtained if 2-opt-URPP is used instead of DROP-ADD, but computing times are then more significant. The combination of Frederickson's algorithm with 2-opt-URPP has produced 92 solutions which are now proved to be optimal using the Branch & Cut algorithm of [24].

Local search techniques are iterative procedures that aim to find a solution s minimizing an objective function f over a set S of feasible solutions. The iterative process starts from an initial solution in S, and given any solution s, the next solution is chosen in the *neighbourhood* $N(s) \subseteq S$. Typically, a neighbour s' in $N(s)$ is obtained from s by performing a local change on it. *Simulated Annealing* [34] and *Tabu Search* [25] are famous local search techniques that appear to be quite successful when applied to a broad range of practical problem.

[30] have designed an adaptation of Tabu Search, called CARPET, for the solution of the UCARP. Tests on benchmark problems have shown that CARPET is a highly efficient heuristic. The algorithm works with two objective functions: $f(s)$, the total travel cost, and a penalized objective function $f'(s) = f(s) + \alpha E(s)$, where α is a positive parameter and $E(s)$ is the total excess weight of all routes in a possibly infeasible solution s. CARPET performs a search over neighbor solutions, by moving at each iteration from the current solution to its best non-tabu neighbor, even if this causes a deterioration in the objective function. A neighbor solution is obtained by moving a required edge from its current route to another one, using procedures DROP and ADD.

Recently, [39] have designed a new local search technique called *Variable Neighborhood Search* (VNS). The basic idea of VNS is to consider several neighborhoods for exploring the solution space, thus reducing the risk of becoming trapped in a local optimum. Several variants of VNS are described in [29]. We describe here the simplest one which performs several descents with different neighborhoods until a local optimum for all considered neighborhoods is reached. This particular variant of VNS is called *Variable neighborhood descent* (VND). Let $N_1, N_2, \ldots, N_K$ denote a set of K neighborhood structures (i.e., $N_i(s)$ contains the solution that can be obtained by performing a local change on s according to the i-th type). VND works as follows.

Variable Neighbourhood Descent

Step 1. Choose an initial solution s in S.
Step 2. Set $i := 1$ and $s_{best} := s$.
Step 3. Perform a descent from s, using neighborhood N_i, and let s' be the resulting solution. If $f(s') < f(s)$ then set $s := s'$. Set $i := i + 1$. If $i \leq K$ then repeat Step 3.
Step 4. If $f(s) < f(s_{best})$ then go to Step 2; else stop.

[32] have designed an adaptation of VND to the undirected CARP, called VND-CARP. The search space S contains all solutions made of a set of vehicle routes covering all required edges and satisfying the vehicle capacity constraints. The objective function to be minimized on S is the total travel cost. The first neighborhood $N_1(s)$ contains solutions obtained from s by moving a required edge (v,w) from its current route T_1 to another one T_2. Route T_2 either contains only the depot (i.e., a new route is created), or a required edge with an endpoint distant from v or w by at most α, where α is the average length of an edge in the network. The addition of (v,w) into T_2 is performed only if there is sufficient residual capacity on T_2 to integrate (v,w). The insertion of (v,w) into T_2 and the removal of (v,w) from T_1 are performed using procedures ADD and DROP described in the previous section.

A neighbor in $N_i(s)$ $(i > 1)$ is obtained by modifying i routes in s as follows. First, a set of i routes in s are merged into a single tour using procedure PASTE, and procedure SWITCH is applied on it to modify the order in which the required edges are visited. Then, procedure CUT divides this tour into feasible routes which are possibly shortened by means of SHORTEN.

As an illustration, consider the solution depicted in Figure 5(a) with three routes T_1 = (depot,a,b,c,d,depot), T_2 = (depot,b,e,f,b,depot) and T_3 = (depot,g,h,*depot)*. The capacity Q of the vehicles is equal to 2, and each required edge has a unit demand. Routes T_1 and T_2 are first merged into a tour T = (depot,a,b,c,d, depot,b,e,f,b,depot) shown in Figure 5(b). Then, SWITCH modifies T into T' = (depot,d,c,b,f,e,b,a,depot,b,depot) represented in Figure 5(c). Procedure CUT divides T' into two feasible routes T'_1 = (depot,d,c,b,f,e,b,depot) and T'_2 = (depot,b,a,depot,b,depot) depicted in Figure 5(d). Finally, these two routes are shortened into T''_1 = (depot,d,c,f,e,b,depot) and T''_2 = (depot,b,a,depot) using SHORTEN. Routes T''_1 and T''_2 together with the third non-modified route T_3 in s constitute a neighbor of s in $N_2(s)$ shown in Figure 5(e).

Hertz and Mittaz have performed a comparison between CARPET, VND-CARP and the following well known heuristics for the UCARP : CONSTRUCT-STRIKE

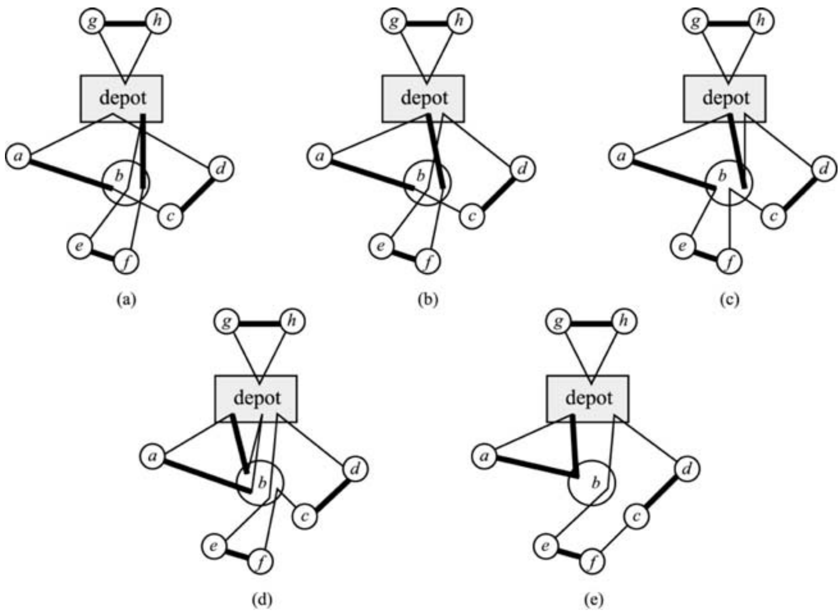

Figure 5. Illustration of neighbourhood N_2.

[7], PATH-SCANNING [26], AUGMENT-MERGE [27], MODIFIED-CONSTRUCT-STRIKE [43] and MODIFIED-PATH-SCANNING [43]. Three sets of test problems have been considered. The first set contains 23 problems described in [13] with $7 \leq |V| \leq 27$ and $11 \leq |E| \leq 55$, all edges requiring a service (i.e. $R = E$). The second set contains 34 instances supplied by Benavent [5] with $24 \leq |V| \leq 50$, $34 \leq |E| \leq 97$ and $R = E$. The third set of instances was generated by Hertz, Laporte and Mittaz [30] in order to evaluate the performance of CARPET. It contains 270 larger instances having 20, 40 or 60 vertices with edge densities in [0.1,0.3], [0.4,0.6] or [0.7,0.9] and $|R|/|E|$ in [0.1,0.3], [0.4,0.6] or [0.8,1.0]. The largest instance contains 1562 required edges.

A lower bound on the optimal value was computed for each instance. This lower bound is the maximum of the three lower bounds CPA, LB2′ and NDLB′ described in the literature. The first, CPA, was proposed by Belenguer and Benavent [4] and is based on a cutting plane procedure. The second and third, LB2′ and NDLB′, are modified versions of LB2 [6] and NDLB [33] respectively. In LB2′ and NDLB′, a lower bound on the number of vehicles required to serve a subset R of edges is computed by means of the lower bounding procedure LR proposed by Martello and Toth [37] for the bin packing problem, instead of $\lceil D/Q \rceil$ (where D is the total demand on R).

Average results are reported in tables 1 and 2 with the following information:

- *Average deviation*: average ratio (in %) of the heuristic solution value over the best known solution value.

Table 1. Computational results on DeArmon instances

	PS	AM	CS	MCS	MPS	CARPET	VND
Average deviation	7.26	5.71	14.03	4.02	4.45	0.17	0.17
Worst deviation	22.27	25.11	43.01	40.83	23.58	2.59	1.94
Number of proven optima	2	3	2	11	5	18	18

- *Worst deviation*: largest ratio (in %) of the heuristic solution value over the best known solution value;
- *Number of proven optima*: number of times the heuristic has produced a solution value equal to the lower bound.

PS, AM, CS, MCS, MPS and VND are abbreviations for PATH-SCANNING, AUGMENT-MERGE, CONSTRUCT-STRIKE, MODIFIED-CONSTRUCT-STRIKE, MODIFIED-PATH-SCANNING and VND-CARP.

It clearly appears in Table 1 that the tested heuristics can be divided into three groups. CONSTRUCT-STRIKE, PATH-SCANNING and AUGMENT-MERGE are constructive algorithms that are not very robust: their average deviation from the best known solution value is larger than 5%, and their worst deviation is larger than 20%. The second group contains MODIFIED-CONSTRUCT-STRIKE and MODIFIED-PATH-SCANNING; while better average deviations can be observed, the worst deviation from the best known solution value is still larger than 20%. The third group contains algorithms CARPET and VND-CARP that are able to generate proven optima for 18 out of 23 instances.

It can be observed in Table 2 that VND-CARP is slightly better than CARPET both in quality and in computing time. Notice that VND-CARP has found 220 proven optima out of 324 instances. As a conclusion to these experiments, it can be observed that the most powerful heuristic methods for the solution of the UCARP all employ on the basic tools described in Section 4.

6. Conclusion and Future Developments

In the field of exact methods, Branch & Cut has known a formidable growth and considerable success on many combinatorial problems. Recent advances made by

Table 2. Computational results on Benavent and Hertz-Laporte-Mittaz instances

	Benavent instances		Hertz-Laporte-Mittaz instances	
	CARPET	VND	CARPET	VND
Average deviation	0.93	0.54	0.71	0.54
Worst deviation	5.14	2.89	8.89	9.16
Number of proven optima	17	17	158	185
Computing times in seconds	34	21	350	42

Corberán and Sanchis [10], Letchford [36] and Ghiani and Laporte [24] indicate that this method also holds much potential for arc routing problems.

In the area of heuristics, basic simple procedures such as POSTPONE, REVERSE, SHORTEN, DROP, ADD, SWITCH, PASTE and CUT have been designed for the URPP and UCARP [31]. These tools can easily be adapted to the directed case [38]. Powerful local search methods have been developed for the UCARP, one being a Tabu Search [30], and the other one a Variable Neighborhood Descent [32].

Future developments will consist in designing similar heuristic and Branch & Cut algorithms for the solution of more realistic arc routing problems, including those defined on directed and mixed graphs, as well as problems incorporating a wider variety of practical constraints.

References

[1] A.A. Assad and B.L. Golden, Arc Routing Methods and Applications, in M.O. Ball, T.L. Magnanti, C.L. Monma and G.L. Nemhauser (eds.), *Network Routing,* Handbooks in Operations Research and Management Science, North-Holland, Amsterdam, (1995).

[2] C. Berge, *Graphs and Hypergraphs*, North-Holland, Amsterdam, (1973).

[3] F. Barahona and M. Grötschel, On the Cycle Polytope of a Binary Matroid, *Journal of Combinatorial Theory* 40: 40–62 (1986).

[4] J.M. Belenguer and E. Benavent, A Cutting Plane Algorithm for the Capacitated Arc Routing Problem, Unpublished manuscript, University of Valencia, Spain, (1997).

[5] E. Benavent, ftp://indurain.estadi.uv.ed/pub/CARP, (1997).

[6] E. Benavent, V. Campos, A. Corberán and E. Mota, The Capacitated Arc Routing Problem: Lower Bounds, *Networks* 22: 669–690 (1992).

[7] N. Christofides, The Optimum Traversal of a Graph, *Omega* 1: 719–732 (1973).

[8] N. Christofides, Worst Case Analysis of a New Heuristic for the Traveling Salesman Problem, Report No 388, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, (1976).

[9] N. Christofides, V. Campos, A. Corberán and E. Mota, An Algorithm for the Rural Postman Problem, Imperial College Report I C.O.R. 81.5, London, (1981).

[10] A. Corberán and J.M. Sanchis, A Polyhedral Approach to the Rural Postman Problem, *European Journal of Operational Research* 79: 95–114 (1994).

[11] G.A. Croes, A Method for Solving Traveling-Salesman Problems, *Operations Research* 6: 791–812 (1958).

[12] G.B. Dantzig, D.R. Fulkerson and S.M. Johnson, Solution of a Large Scale Traveling Salesman Problem, *Operations Research* 2: 393–410 (1954).

[13] J.S. DeArmon, A Comparison of Heuristics for the Capacitated Chinese Postman Problem, Master's Thesis, University of Maryland at College Park, (1981).

[14] M. Dror, *ARC Routing : Theory, Solutions and Applications*, Kluwer Academic Publishers, Boston, (2000).

[15] M. Dror, H. Stern and P. Trudeau Postman Tour on a Graph with Precedence Relation on Arcs, *Networks* 17: 283–294 (1987).

[16] J. Edmonds and E.L. Johnson, Matching, Euler Tours and the Chinese Postman Problem, *Mathematical Programming* 5: 88–124 (1973).

[17] R.W. Eglese, Routing Winter Gritting Vehicles, *Discrete Applied Mathematics* 48: 231–244 (1994).

[18] R.W. Eglese and L.Y.O. Li, A Tabu Search Based Heuristic for Arc Routing with a Capacity Constraint and Time Deadline, in I.H. Osman and J.P. Kelly (eds.), *Meta-Heuristics: Theory & Applications,* Kluwer Academic Publishers, Boston, (1996).

[19] H.A. Eiselt, M. Gendreau and G. Laporte, Arc Routing Problems, Part 1: The Chinese Postman Problem, *Operations Research* 43: 231–242 (1995a).

[20] H.A. Eiselt, M. Gendreau and G. Laporte, Arc Routing Problems, Part 2: The Rural Postman Problem, *Operations Research* 43: 399–414 (1995b).

[21] M. Fischetti, J.J. Salazar and P. Toth, A Branch-and Cut Algorithm for the Symmetric Generalized Traveling Salesman Problem, *Operations Research* 45: 378–394 (1997).

[22] G.N. Frederickson, Approximation Algorithms for Some Postman Problems, *Journal of the A.C.M.* 26: 538–554 (1979).

[23] M. Gendreau, A. Hertz and G. Laporte, A Tabu Search Heuristic for the Vehicle Routing Problem, *Management Science* 40: 1276–1290 (1994).

[24] G. Ghiani and G. Laporte, A Branch-and-cut Algorithm for the Undirected Rural Postman Problem, *Mathematical Programming* 87: 467–481 (2000).

[25] F. Glover, Tabu Search – Part I, *ORSA Journal on Computing* 1: 190–206 (1989).

[26] B.L. Golden, J.S. DeArmon and E.K. Baker, Computational Experiments with Algorithms for a Class of Routing Problems, *Computers & Operations Research* 10: 47–59 (1983).

[27] B.L. Golden and R.T. Wong, Capacitated Arc Routing Problems, *Networks* 11: 305–315 (1981).

[28] M. Grötschel and Z. Win, A Cutting Plane Algorithm for the Windy Postman Problem, *Mathematical Programming* 55: 339–358 (1992).

[29] P. Hansen and N. Mladenović, An Introduction to VNS. In S. Voss, S. Martello, I.H. Osman and C. Roucairol (eds.), *Meta-Heuristics : Advances and Trends in Local Search Paradigms for Optimization*, Kluwer Academic Publishers, Boston, (1998).

[30] A. Hertz, G. Laporte and M. Mittaz, A Tabu Search Heuristic for the Capacitated Arc Routing Problem, *Operations Research* 48: 129–135 (2000).

[31] A. Hertz, G. Laporte and P. Nanchen-Hugo, Improvement Procedure for the Undirected Rural Postman Problem, *INFORMS Journal on Computing* 11: 53–62 (1999).

[32] A. Hertz and M. Mittaz, A Variable Neighborhood Descent Algorithm for the Undirected Capacitated Arc Routing Problem, *Transportation Science* 35: 425–434 (2001).

[33] R. Hirabayashi, Y. Saruwatari and N. Nishida, Tour Construction Algorithm for the Capacitated Arc Routing Problem, *Asia-Pacific Journal of Operational Research* 9: 155–175 (1992).

[34] S. Kirkpatrick, C.D. Gelatt and M. Vecchi, M. Optimization by Simulated Annealing, *Science* 220: 671–680 (1983).

[35] J.K. Lenstra and A.H.G. Rinnooy Kan, On General Routing Problems, *Networks* 6: 273–280 (1976).

[36] A.N. Letchford, Polyhedral Results for some Constrained Arc Routing Problems, PhD Dissertation, Dept. of Management Science, Lancaster University, (1997).

[37] S. Martello and P. Toth, Lower Bounds and Reduction Procedures for the Bin Packing Problem, *Discrete Applied Mathematics* 28: 59–70 (1990).

[38] M. Mittaz, Problèmes de Cheminements Optimaux dans des Réseaux avec Contraintes Associées aux Arcs, PhD Dissertation, Department of Mathematics, Ecole Polytechnique Fédérale de Lausanne, (1999).

[39] N. Mladenović and P. Hansen, Variable Neigbhourhood Search, *Computers & Operations Research* 34: 1097–1100 (1997).

[40] C.S. Orloff, A Fundamental Problem in Vehicle Rouring, *Networks* 4: 35–64 (1974).

[41] M.W. Padberg and M.R. Rao, Odd Minimum Cut-Setss and *b*-Matchings, *Mathematics for Operations Research* 7: 67–80 (1982).

[42] M.W. Padberg and G. Rinaldi, A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems, *SIAM Review* 33: 60–100 (1991).

[43] W.-L. Pearn, Approximate Solutions for the Capacitated Arc Routing Problem, *Computers & Operations Research* 16: 589–600 (1989).

[44] S. Roy and J.M. Rousseau, The Capacitated Canadian Postman Problem, *INFOR* 27: 58–73 (1989).

[45] L.A. Wolsey, *Integer Programming*, John Wiley & Sons, (1998).