New Heuristic Algorithms for the Windy Rural Postman Problem

E. Benavent[1], A. Corberán[1], E. Piñana[1], I. Plana[1], J.M. Sanchis[2]

[1] Dept. d'Estadística i Investigació Operativa. Universitat de València. Spain

[2] Dept. de Matemática Aplicada. Universidad Politécnica de Valencia. Spain

May 21, 2003

Abstract

In this paper we deal with the Windy Rural Postman Problem. This problem generalizes several important Arc Routing Problems and has interesting real-life applications. Here, we present several heuristics whose study has lead to the design of a Scatter Search algorithm for the Windy Rural Postman Problem. Extensive computational experiments over different sets of instances, with sizes up to 988 nodes and 3952 edges, are also presented.

1 Introduction

In the well known (undirected) Chinese Postman Problem (CPP) the cost of traversing an edge (i, j) from i to j is the same as that of traversing it from j to i. If, as in some real world applications, this is not the case, the problem is called the Windy Postman Problem (WPP). The WPP was proposed by Minieka (1979), who named this problem illustrating the work that a postman should perform on a windy day.

It is easy to see that, by a properly definition of the edge costs, the undirected, directed and mixed Chinese Postman problems are special cases of the WPP. Therefore, since the Mixed Chinese Postman Problem (MCPP) is NP-hard, the WPP is also a NP-hard problem (Brucker, 1981 and Guan, 1984), although some particular cases can be solved in polynomial time (Guan, 1984 and Win, 1989). An integer linear programming formulation of the WPP was given in Grötschel & Win (1992). In that paper, the authors also proposed a cutting plane algorithm, based on a partial description of the Windy Postman polyhedron, producing good computational results. Concerning to the approximate resolution of the WPP, several heuristic algorithms have been proposed by Win (1987) and Pearn & Li (1994).

In this paper we deal with a generalization of the WPP, the Windy Rural Postman Problem (WRPP). This problem can be described as follows. Let $G = (V, E)$ be an undirected and connected graph with two nonnegative costs, c_{ij}, c_{ji}, associated to each edge $(i, j) \in E$, representing the cost of traversing the edge from i to j and from j to i, respectively. Let E_R be a subset of edges of E, that will be called the required edges

set. Then, the WRPP is the problem of finding a minimum cost closed walk traversing every edge in E_R at least once.

Obviously, if $E_R = E$, the WRPP reduces to the WPP, but note that the WRPP also generalizes the undirected, directed and mixed Rural Postman problems. Therefore, the WRPP is a difficult NP-hard problem that contains as special cases most of the known arc routing problems with a single vehicle. Moreover, besides its great theoretical interest, the WRPP has some very interesting practical applications (see Benavent et al., 2003).

Up to the authors knowledge, the only paper dealing with the WRPP is that of Benavent et al. (2003). In their paper, authors propose an integer linear programming formulation and different heuristics and lower bounds for the WRPP. Lower bounds are obtained by means of a cutting plane procedure that, besides using the inequalities in the WRPP formulation, it also identifies other violated valid inequalities coming from families of facet defining inequalities for the RPP polyhedron. The upper bounds are computed by means of three constructive heuristics and several improvement procedures. The computational results are very good. The cutting plane algorithm was able to solve 185 out of 288 tested instances. For the unsolved instances, the average deviation from the optimal value is always less than 1 %. On the other side, the best solution among those obtained with the 3 heuristic procedures proposed in Benavent et al. (2003) is 4% on average from the lower bound and it is computed in less than 1 second.

In what follows we will assume, for the sake of simplicity and without loss of generality, that all the nodes in V are incident with required edges. From the original graph $G = (V, E)$, let G_R be the graph induced by the required edges, i.e. $G_R = (V, E_R)$. Graph G_R is in general not connected. The sets of nodes of the connected components of G_R are usually called R-sets. A WRPP solution will be a strongly connected directed multigraph $G^* = (V, A)$ satisfying that: every node in G^* is symmetric, each arc $(i, j) \in A$ is a copy of an edge in E with a given orientation, and for each $(i, j) \in E_R$, either (i, j),(j, i), or both belong to A.

It is well known that there exists a closed walk (tour) traversing each arc of G^* exactly once. This tour is an alternative way of representing a WRPP solution.

In this paper, we present new heuristic algorithms which have a very good computational behavior on different sets of instances. In Section 2 we summarize two of the algorithms described in Benavent et al. (2003) and present two new constructive algorithms which are modified versions of them. Section 3 describes several improvement procedures, while Section 4 describes four Multi-Start procedures that have been obtained by randomizing certain steps of the constructive algorithms. Finally, an Scatter Search algorithm that combines the best previous procedures is presented in Section 5. Computational results for each class of heuristics are included in their respective sections. We have tried a large set of instances including those used by Benavent et al. (2003) and also some new large instances that have been randomly generated. Some conclusions are presented in Section 6.

2 Constructive Heuristics

As it has been mentioned earlier, the Mixed Chinese Postman Problem is a particular case of the WRPP. For this problem, Frederickson (1979) proposed a constructive procedure, the *Mixed Algorithm*, that has a worst case ratio of 5/3. A clever modification of this algorithm (Raghavachari & Veerasamy, 1998) led to a procedure with a better worst case ratio of 3/2. Not surprisingly, it has been shown very recently (see [5]) that the modified algorithm, besides its better theoretical behavior, also produces very good computational results. Then, we have introduced similar ideas into two of the heuristics described in Benavent et al. (2003) to obtain two new constructive heuristics for the WRPP, *H1* and *H2*, that are described in what follows. First, we begin this section by summarizing the original heuristic algorithms, *WRPP1* and *WRPP2*, proposed in [2]. The solutions generated by all the algorithms are improved by means of several simple procedures also described in [2].

2.1 Algorithm WRPP1

This algorithm consists of three phases. First, the components of graph G_R are joined to obtain a connected graph. To do this, an auxiliary graph G_{aux}, having a node for each connected component of G_R, is constructed. For any pair of nodes u and v of G_{aux} the shortest path on G from any node in component u to any node in v is calculated, and conversely. An edge between nodes u and v is created with cost equal to the minimum of the average costs of these two paths. The *average cost of a path* $i_1, i_2, \ldots, i_p$ is defined as $\frac{1}{2}(c_{i_1 i_2} + c_{i_2 i_1} + c_{i_2 i_3} + c_{i_3 i_2} \ldots)$. Then, a shortest spanning tree (SST) is computed on G_{aux}. The edges in the shortest path in G associated to each edge in the spanning tree are added to G_R to obtain a new graph G'_R. Now, a minimum cost matching problem on the odd degree nodes of graph G'_R is solved. The costs used in the matching problem are defined in the same way as before. Again, edges in the paths in G corresponding to the edges in the matching solution are added to G'_R to obtain an even and connected graph G''_R. Win's exact algorithm for the WPP defined on Eulerian graphs is then applied on G''_R to obtain a feasible solution to the WRPP on G. This algorithm (Win, 1989) consists of solving a minimum cost flow problem on an auxiliary graph.

2.2 Algorithm WRPP2

Basically, this second algorithm executes the same phases of algorithm *WRPP1* but in a different order. It begins by assigning to each edge in G_R the direction associated to its minimal traversing cost. Let us represent by G^d_R the resulting directed graph. The demand $d(i)$ of each node i is then computed as the difference between the arcs in G^d_R entering at and leaving from i. From graph G^d_R a new balanced mixed graph is obtained as follows. First, an auxiliary digraph $G_{aux} = (V, A)$ is constructed, where for each arc (i, j) in G^d_R three arcs are put in A: two arcs (i, j) and (j, i) with costs c_{ij} and c_{ji}, respectively, and capacity ∞, and another arc (j, i) with cost $\frac{c_{ji}-c_{ij}}{2}$ and capacity 2. Note that by the definition of G^d_R, $c_{ij} \leq c_{ji}$.

On the graph G_{aux}, a minimum cost flow problem with demands $d(i)$ is then solved. From its solution, a balanced mixed graph $G^1 = (V, E^1, A^1)$ is constructed in the following way. Let f_{ij} and f'_{ij} be the flow units in the solution through the arcs (i, j) of capacity ∞ and those of capacity 2, respectively. For each arc (i, j) in A of capacity 2:

- if $f'_{ij} = 0$, $f_{ji} + 1$ arcs (j, i) are added to A^1,
- if $f'_{ij} = 1$, an edge (i, j) is added to E^1, and
- if $f'_{ij} = 2$, $f_{ij} + 1$ arcs (i, j) are added to A^1.

If G^1 has odd degree nodes, a second phase consisting of the resolution of a minimum cost matching problem is executed. Edge costs for the matching problem are defined in the same way as in algorithm *WRPP1*. For each edge in the matching solution, the edges in its associated path in G are added to G^1 to obtain an even mixed graph G^2. Again, edges in G^2 are oriented in the direction of their minimal traversing cost, and a new network flow problem, similar to that described in the first phase, is solved to obtain a symmetric digraph G^3.

Finally, as graph G^3 may be disconnected, a third phase consisting of the resolution of a shortest spanning tree connecting its components is executed.

2.3 Algorithm H1

This new constructive heuristic is a modification of algorithm *WRPP1*. As in the *Modified Mixed Algorithm* for the MCPP [19], the idea of the new algorithm is to try to anticipate during the first steps what will happen in the last phases of the algorithm.

First, the algorithm connects the components of G_R by adding the edges of a shortest spanning tree computed as in the first phase of algorithm *WRPP1*. Again, let us represent by $G'_R = (V, E'_R)$ the resulting graph. Now, before solving a minimum cost matching problem to make G'_R even, we will change the edge costs in such a way the solution to the matching problem includes, at least partially, the solution to the flow problem of the last phase.

In order to do this, first, the *average cost* C_a of the edges in E'_R is computed as $C_a = \frac{1}{|E'_R|} \sum_{ij \in E'_R} \frac{c_{ij}+c_{ji}}{2}$. Now, the sets $E_1 = \{(i, j) \in E'_R : |c_{ji} - c_{ij}| > K * C_a\}$ and $E_2 = E \backslash E_1$ are defined, where K is a positive parameter that, after some computational testing, has been fixed to 0.2. Then, a directed graph G^d_R is constructed with set of nodes V and an arc (i, j) for each edge $(i, j) \in E_1$ (we assume $c_{ij} \leq c_{ji}$). Note that the set of arcs in G^d_R is associated to the edges in G'_R whose traversal is quite more expensive in one direction than in its opposite one. Our guess is that, in the final solution, these edges will be traversed in the direction of their minimal cost. The demand $d(u)$ of each node u is then computed in graph G^d_R. A minimum cost flow problem with demands defined by $d(u)$ is now solved on an auxiliary digraph $G_{aux} = (V, A_{aux})$ defined as follows. Arc set A_{aux} contains two arcs (i, j) and (j, i), with costs c_{ij} and c_{ji}, respectively, and infinite capacity, for each edge of the original graph G. Furthermore, for the edges $(i, j) \in E_1$, another arc (j, i) with cost $\frac{c_{ji}-c_{ij}}{2}$ and

capacity 2 is added to A_{aux}. This last arc would allow to change the direction formerly assigned to edge (i, j) in graph G_R^d.

Let f_{ij} be the flow through each arc $(i, j) \in A_{aux}$ with infinite capacity. A list of edges L is created, containing those edges satisfying one of the following conditions:

- $(i, j) \in E_1$ and $f_{ij} \geq 1$
- $(i, j) \in E_1$ and $f_{ji} \geq 1$
- $(i, j) \in E_2$ and $f_{ij} \geq 2$
- $(i, j) \in E_2$ and $f_{ji} \geq 2$

Note that list L contains those oriented edges in G_R^d for which a flow of at least 1 unit through them suggests that they will appear more than once in this direction or in the opposite one in the solution. Furthermore, non oriented edges in G_R^d but with a flow of at least two units through them seem to be good candidates to appear in the solution and, therefore, they have been added to L. Edges belonging to this list will be those whose costs will be modified in the next phase of the algorithm.

The heuristic now proceeds to make graph G'_R even. To do this, a minimum cost matching problem is solved exactly as in the second step of algorithm *WRPP1* but using costs $c_{ij} = c_{ji} = 0$ for the edges in L. The other edges have original costs.

Edges in the resulting graph are finally oriented by solving a minimum cost flow problem as at the end of algorithm *WRPP1*. The costs used in this phase will be the original ones for all the edges.

2.4 Algorithm H2

This other constructive heuristic is a modification of algorithm *WRPP2*. Again, we try to guess which arcs will be added to the solution in the last phase of the procedure in order to introduce them in the solution during the previous steps. We have then added to the original *WRPP2* heuristic an initial phase in which a shortest spanning tree is computed. From the SST solution, some edge costs are changed. These new costs will be used during the first two phases of *WRPP2*.

Algorithm *H2* starts computing a SST connecting the components of graph G_R in the same way as, for instance, algorithm *WRPP1* does. However, the edges in the shortest path in G associated to each edge in the spanning tree are not added here to G_R, but listed in L and their average cost is computed ($C_a = \frac{1}{|L|} \sum_{ij \in L} \frac{c_{ij}+c_{ji}}{2}$). For each edge $(i, j) \in E$ (we will assume that $c_{ij} \leq c_{ji}$) new costs c'_{ij} and c'_{ji} are now defined as follows:

- $c'_{ij} = c_{ij} - \alpha C_a$ and $c'_{ji} = c_{ji} - \alpha C_a$, if i and j are in different connected components of G_R and $c_{ij} \geq \alpha C_a$.
- $c'_{ij} = 0$ and $c'_{ji} = c_{ji} - c_{ij}$, if i and j are in different connected components of G_R and $c_{ij} < \alpha C_a$.
- $c'_{ij} = c_{ij}$ and $c'_{ji} = c_{ji}$, otherwise,

where α is a parameter that after some testing has been fixed to 0.8.

Algorithm *WRPP2* is then executed with these new costs. Only the computation of the shortest spanning tree during the last phase of the algorithm, if needed (when G^3 is not a connected graph), is performed with the original costs c_{ij}.

2.5 Improvement Procedures

Several improvement procedures proposed in [2] have been also applied to the solutions generated by heuristics *H1* and *H2*. The two first methods are based on eliminating redundant cycles from the solution. By redundant cycle we mean a cycle defined by a subset of arcs associated to non required edges or to copies of required edges. Note that a redundant cycle can be removed from the solution only if the resulting graph keeps being strongly connected and contains at least one arc corresponding to each original required edge. The last improvement procedure is widely used in the context of Arc Routing Problems. It consists of first constructing an Eulerian tour traversing the solution graph and then substituting any path of non-required arcs connecting two consecutive required edges by a shortest path.

2.6 Computational comparison

Several computational experiments have been done in order to compare the performance of the constructive algorithms *WRPP1* and *WRPP2* and their new versions *H1* and *H2*. All the heuristics were run over 144 instances randomly generated by Benavent et al. (2003) from the Albaida (Valencia, Spain) and the Madrigueras (Albacete, Spain) real street networks. The characteristics of these instances are shown in table 7. Table 1 shows the results obtained with the four heuristic algorithms without applying the above improvement procedures (row 1) and after applying them (row 2) over the full set of instances. All the entries in the table are average percentage deviations from the lower bound obtained with the cutting plane algorithm described in [2]. In all these instances lower bounds are, in fact, optimal.

Average Deviation (%)	WRPP1	WRPP2	H1	H2
Without improvements	4.22	3.92	4.17	3.38
With improvements	3.44	3.06	3.13	2.12

Table 1: Constructive heuristics comparison

From the results in Table 1 it can be noticed that modificated algorithms *H1* and *H2* present a better behavior than their respective original algorithms *WRPP1* and *WRPP2*, both before and after the improvement procedures are applied.

3 New improvement procedures

In this section we present some new methods we have applied to improve the quality of the solutions obtained by the heuristic algorithms described in this work. First, we

generate a compact representation of the solution, which may decrease the solution cost as it substitutes the paths among required edges by shortest paths. Furthermore, for a given order of traversal of the required edges, another method finds the optimal direction in which each required edge has to be traversed. Finally, several local search procedures are described.

3.1 Representing the WRPP solutions

A WRPP solution is a tour, that is, a closed walk containing each required edge at least once in one of the two possible directions of traversal. Let us denote by $m = |E_R|$. We assign to each required edge two different numbers: the first one, say k, $1 \leq k \leq m$, is associated to the traversal of the edge in one direction, and the second one, $k + m$, to the opposite direction of traversal. Let us consider a sequence of m numbers between 1 and $2m$, such that, for each pair of numbers k and $k+m$, with $1 \leq k \leq m$, exactly one of them belongs to the sequence. This sequence determines a WRPP solution if we assume that the tour follows the shortest path to travel from the end node of a required edge to the initial node of the next required edge in the sequence, as well as, in order to close the tour, from the end node of the last edge to the initial node of the first one. Note that the optimal WRPP solution may indeed be represented in that way, so we may restrict ourselves to work only with such solutions.

Therefore, we may represent a solution with a sequence of m numbers, called *compact representation*, which is very convenient to be used in evolutionary algorithms because it needs less memory and its length is constant. A similar representation was introduced by Lacomme et al. (2001) for the Capacitated Arc Routing Problem (CARP).

The constructive algorithms for the WRPP presented in the previous section produce solutions that can be converted into WRPP tours. In order to obtain their compact representation, we identify the first traversal of every required edge in the tour thus determining the sequence in which the required edges are covered by it. Note that the path followed in the original tour from a given required edge to the next one not yet covered, may not be a shortest path, so the compact representation may in fact have a lesser cost than the original tour. Furthermore, some required edges may be traversed several times by the tour, so the compact representation may not be unique. The following two sections present improvement procedures for the WRPP that can be used with the compact representation.

3.2 Reversing the direction of traversal

Given the compact representation of a solution, $a_1, a_2, \ldots, a_m$, each element of the sequence represents the traversal of a required edge in a given direction. If one edge is traversed in the opposite direction, we obtain a different solution with possibly a different cost but where the required edges are covered in the same order. We may address the problem of determining the best direction of traversal of each required edge by keeping unchanged the order in which they are covered. This can be solved in polynomial time with a procedure proposed by Lacomme, Prins & Ramdane-Chérif

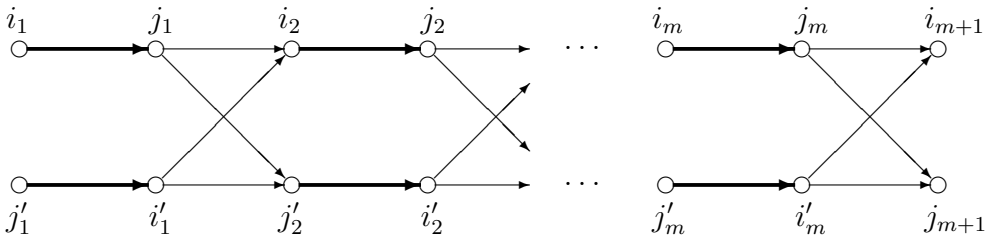

Figure 1: Reversing the direction of traversal

(see [13] or [20]), that is described as follows.

Let H be a directed graph which contains, for each required edge $a_r = (i_r, j_r)$, $r = 1, \ldots, m$, two arcs representing the two possible traversals of a_r with their corresponding costs. These arcs will be denoted (i_r, j_r) and (j'_r, i'_r), with costs $c_{i_r j_r}$ and $c_{j_r i_r}$ respectively, where i'_r and j'_r are copies of nodes i_r and j_r. For each two consecutive required edges in the solution, (i_r, j_r) and (i_{r+1}, j_{r+1}), $r = 1, \ldots, m-1$, four more arcs are added to H: (j_r, i_{r+1}), (j_r, j'_{r+1}), (i'_r, i_{r+1}) and (i'_r, j'_{r+1}), with costs equal to the shortest path costs in G between their corresponding original nodes (see figure 1).

In order to take into account the cost of travelling from the end node of the last edge $a_m = (i_m, j_m)$, to the initial node of the first edge $a_1 = (i_1, j_1)$, we add two new nodes, denoted by i_{m+1} and j_{m+1}, which are additional copies of nodes i_1 and j_1 respectively, and four arcs: (j_m, i_{m+1}), (j_m, j_{m+1}), (i'_m, i_{m+1}) and (i'_m, j_{m+1}) with the costs of the corresponding shortest paths (see figure 1).

Then, two shortest paths are computed, one from i_1 to i_{m+1} and another from j'_1 to j_{m+1}, and the one with the lesser cost is selected. Note that, for every required edge (i_r, j_r), $r = 1, \ldots, m$, this path uses exactly one arc from the two ones representing it, (i_r, j_r) and (j'_r, i'_r), thus determining the direction in which it must be traversed to minimize the total cost.

3.3 Local search phase

Here we propose two improvement procedures based on local search. The first one is a steepest-descent algorithm in which a simple move is performed at each iteration. This move consists of interchanging the positions of two required edges in the sequence (see Figure 2) and it is called 2-interchange. The second procedure is also a steepest-descent algorithm in which each move is an Or-interchange (see [16]). In this move, a section consisting of at most L consecutive required edges in the sequence is inserted elsewhere between two consecutive required edges, the first one of which must be among the K edges closest to the first edge of the section. This insertion is performed in such a way that the direction of traversal remains unchanged (see figure 3). If the section consists of only one edge, we also try to insert it changing its the direction of traversal (see figure 4).

Some computational experiments have been done in order to find good values for parameters L and K. Algorithm *WRPP2* was used to generate a set of 100 different solutions for each of a subset of 10 representative instances. The details about how these solutions have been generated will be given in the next section. The two improvement

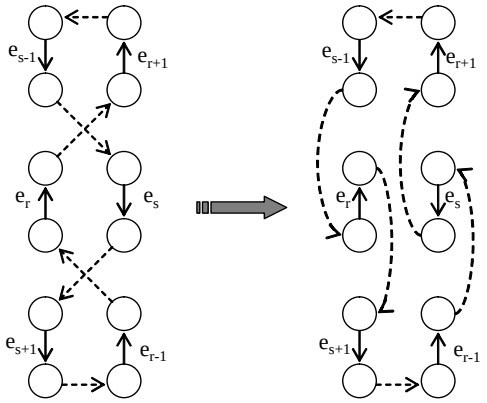

Figure 2: 2-interchange

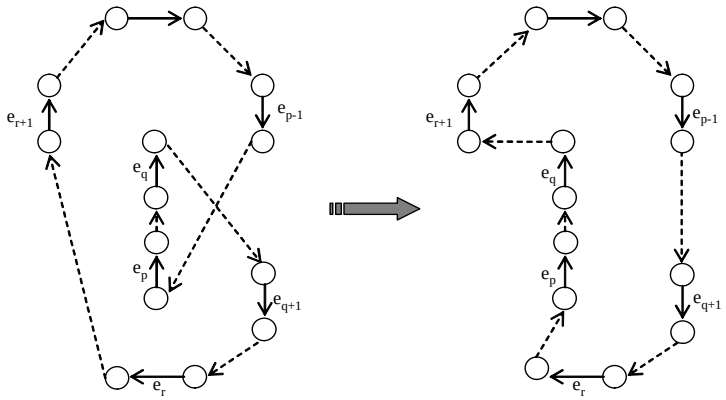

Figure 3: Or-interchange

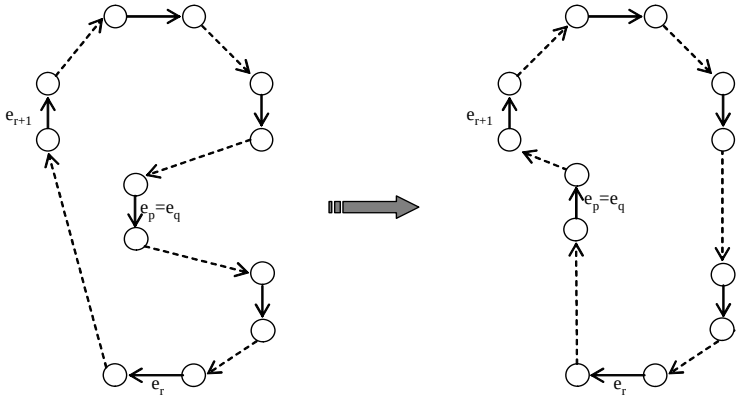

Figure 4: Or-interchange

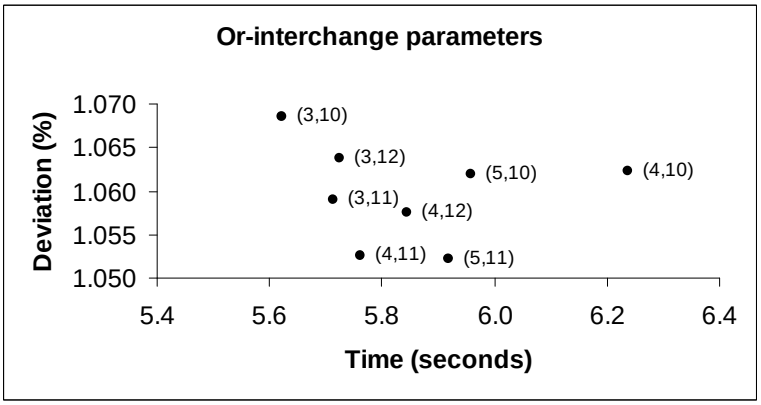

Figure 5: Results with different values for L and K

methods proposed in [2] were applied to each generated solution to eliminate redundant cycles. The solutions were then transformed into their compact representation. The Or-interchange algorithm mentioned above was finally applied with values for L between 3 and 5 and for K between 10 and 12. Figure 5 shows the average deviation of the best solution found and the time used (in seconds) for each test. The numbers in brackets show the values used for L and K. From these results, we decided to use $L = 4$ and $K = 11$.

To determine the best order in which the improvements methods should be applied, more experiments were done. Again, the same set of 100 solutions were used in these experiments. The redundant cycles of the solutions were eliminated and the transformation into the compact representation applied. Then the two local search procedures (labeled *2-int* and *Or*) and the method that finds the optimal traversal direction of the edges described in Section 3.2 (labeled *rev*) were applied in different orders and different combinations (each one on its own, two of them and all of them). Table 2 shows the average deviation from the lower bound of the best solution found and the average computing times.

	2-int	*Or*	*rev*	*2-int/Or*	*Or/2-int*	*Or/rev*	*Or/2-int/rev*
Deviation (%)	1.31	1.05	1.25	1.07	1.01	0.98	0.93
Time (scs.)	6.36	5.76	5.96	6.48	6.43	6.06	6.58

Table 2: Local Search. Ordering of improvement methods

From the results obtained, we decided to apply first the Or-interchange, then the 2-interchange and finally the reversing method.

4 Multi-Start Algorithms

From the constructive algorithms described in Section 2, it is possible to design several Multi-Start heuristics. Basically, Multi-Start algorithms consist of the execution of

a number of global iterations until some stopping criterium is satisfied. Each global iteration generates a solution with a constructive algorithm and then improves it with a local search method. A description of both phases follows.

Each one of the heuristics described in section 2 produces only one solution. In order to obtain a set of different feasible solutions with each algorithm, several random elements have been introduced in some of their steps. To get solutions as different from each other as possible, the random elements have been introduced in the first phases of the algorithms.

In the *WRPP1* and *H1* algorithms, the construction of the spanning tree has been modified. Now, at each step of the tree computation, the chosen edge will not be necessarily that of minimum cost since it will be randomly chosen from a set of edges with the lower costs. The size of this set has been fixed to $\lceil \frac{|V|}{p} \rceil$, where p is the number of R-sets. The spanning tree thus obtained will be in general different for each execution of the corresponding algorithm.

If $p = 1$, i.e. G_R consists of only one connected component and the WRPP reduces to a WPP, the above procedure would not work, as no spanning tree is needed. In such a case, edge costs are randomly modified previously to each execution of the algorithm. The cost modification should be large enough to get different solutions, but not as large as to produce bad solutions. New costs are randomly generated in the interval $[0.5c_{ij}, 1.5c_{ij}]$, where c_{ij} denotes the original costs.

Since the spanning tree computation is done in the last phase of algorithms *WRPP2* and *H2*, its randomization is also not appropriate in this case. Here, we have tried two different strategies. The first one consists of modifying the cost of the edges in the same way as in the case $p = 1$ above. The second one does not modify the cost of the edges, but the cost of the shortest paths. The cost $\hat{c}_{ij}$ of the shortest path between every pair of nodes i, j is substituted by a new cost randomly chosen in the interval $[0.6\hat{c}_{ij}, 1.4\hat{c}_{ij}]$, i.e. with a maximum deviation of 40% from the original cost. After some computational testing, the second strategy resulted to be more efficient, so we decided to use this one.

The improvement phase consists of applying first the Or-interchange, then the 2-interchange and finally the reversing procedure. The Multi-Start procedures thus obtained will be denoted by MS1, MS2, MSH1 and MSH2, respectively.

4.1 Computational results

Table 3 presents the average percentage deviations of the best solutions obtained on the Albaida and Madrigueras instances after performing 250 iterations of the Multi-Start algorithms, while table 4 shows the average deviations after running the algorithms for 2 minutes, without limit of iterations. Figures 6 and 7 show the evolution of the average deviations on all the instances with both stopping criteria.

From the results, notice that algorithm MS2 is the best of all four procedures, achieving deviations of 0.47% and 0.64% for the Albaida and Madrigueras instances respectively, if we let it run for 2 minutes, although most of the best solutions are found within the first 40 seconds. After that time it becomes more difficult for MS2 to find better solutions.

Deviation (%)	MS1	MS2	MSH1	MSH2
Albaida	1.22	0.68	1.05	0.71
Madrigueras	2.26	0.82	2.03	1.12

Table 3: MS with 250 iterations

Deviation (%)	MS1	MS2	MSH1	MSH2
Albaida	0.80	0.47	0.75	0.60
Madrigueras	1.98	0.64	1.91	1.21

Table 4: MS with 120 seconds

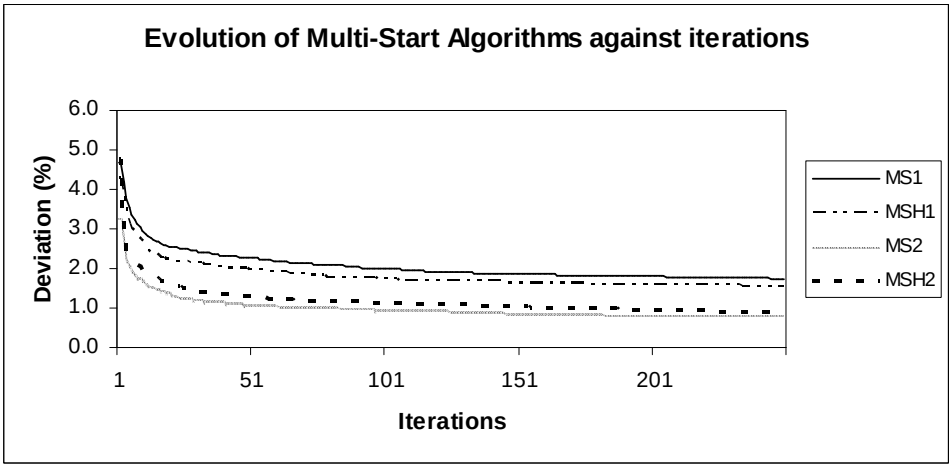

Figure 6: Multi-start evolution (iterations)

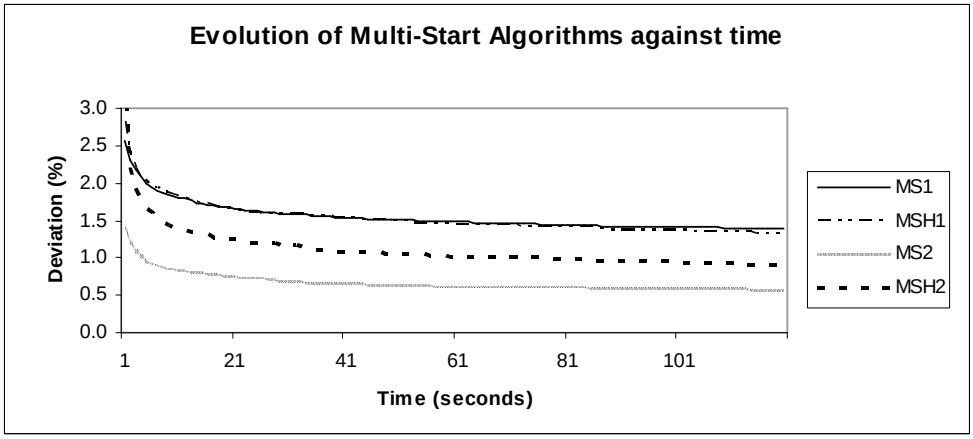

Figure 7: Multi-start evolution (time)

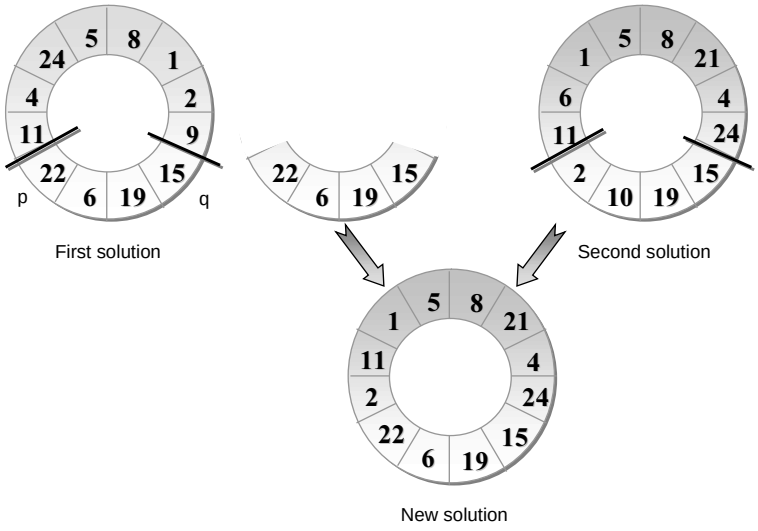

Figure 8: Order crossover operator

5 A Scatter search algorithm

Scatter search (see the recent book by Laguna & Martí, 2003) is a population-based method that has been shown to yield promising outcomes for solving hard combinatorial optimization problems. The Scatter Search process tries to capture information not contained separately in the original solutions, takes advantage of improving methods, and makes use of strategy instead of randomization to carry out component steps. Our implementation is based on the description given in [8].

The procedure starts with the construction of an initial reference set of solutions (RefSet) from a large set P of diverse solutions. The solutions in RefSet are ordered according to quality, where the best solution is the first one in the list. The search is then initiated by combining solutions in RefSet. Although the method can use combinations of any number of solutions, in this implementation we restrict our attention to combinations of 2 solutions. RefSet is updated with the newly created solutions using different strategies. The procedure stops when, after combining all the solutions in RefSet, it remains unchanged.

Our scatter search algorithm starts with an initial population P with 200 different solutions generated with the construction phase of the MS2 heuristic, to which only the Or-interchange and the reversing direction procedures are applied. From this set, the best k solutions are included in RefSet. Other l solutions are added in such a way that the diversity of RefSet is as large as possible. In order to do this we define the distance between any pair of solutions as m minus the number of shortest paths between required edges that both solutions have in common. Then, we iteratively select the solution from P to be added to RefSet as the one whose minimum distance to those solutions already in RefSet is maximal.

To combine the solutions in the reference set, we apply a version of the classical order crossover similar to that used in Lacomme et al. (2001) (see figure 8). Given two

WRPP solutions in their compact representation (as sequences of the required edges), $\{i_1, i_2, \ldots, i_m\}, \{j_1, j_2, \ldots, j_m\}$, two elements of the first solution, i_p, i_q, are randomly selected. The new solution will contain the segment $\{i_p, i_{p+1}, \ldots, i_q\}$ as it appears in the first solution. Then, starting from the element j_{q+1} in the second solution, all the elements in this solution not yet included in the new one are sequentially added to the first segment in the new solution. Note that the solutions are really cycles. Therefore, if the new solution has not been completed when we arrive to element j_m, we continue the inspection of the second solution starting now from j_1.

Similarly, starting with the segment $\{i_{q+1}, \ldots, i_m, i_1, \ldots, i_{p-1}\}$, another solution is obtained. The best of the two new solutions is selected, and a local search phase is applied to it. This phase consists of the Or-interchange, the 2-interchange and the reversing direction procedures described earlier.

If the new solution is good enough, it can immediately replace the worst solution in RefSet (dynamic strategy) or be stored in a secondary set that will be used to update RefSet when all possible combinations of solutions have been performed (static strategy). In the last case, the new Refset will contain the best solutions among those in the old Refset and in the secondary set. The dynamic strategy changes RefSet very quickly and usually produces good solutions in short times, while the static strategy can be slower but usually produces better solutions.

Table 5 shows the results obtained with both strategies on the Albaida and Madrigueras instances for different values of k (number of solutions in RefSet selected for quality) and l (number of solutions in RefSet selected for diversity). As it can be seen from the tables, the static strategy always outperforms the dynamic one, while running times are quite similar. Therefore, we decided to implement in our algorithm the static strategy.

As an alternative, we have tried a different method of combining solutions. The two parent solutions are transformed into a directed graph each, where the shortest path from a required edge to the next one is represented by an arc. Now a new graph is constructed containing all the arcs of these two graphs. Note that this new graph can contain duplicated arcs and arcs traversed in both directions. An eulerian circuit, which is also a solution of the WRPP, is now found in this graph and then transformed into the compact representation described in section 3.1. As with the method for combining solutions explained above, the local phase is applied to the new solution and, if appropriated, RefSet is updated. The results obtained with the Scatter Search algorithm, using the static strategy and this new combination method, on the same sets of instances are shown in Table 6. Since this alternative produced worst results both in terms of quality and computing time, it was not further explored.

5.1 Computational results

In order to analyze the behavior of the Scatter Search algorithm proposed in the previous section, we have performed an extensive computational study on several sets of instances of different sizes and characteristics, summarized in table 7.

First, we have tested the algorithm on the three sets of instances proposed in [2]: Albaida, Madrigueras and Christofides et al. instances. The next set of WRPP

	RefSet size (k-l)	5-5	10-0	10-5	15-0
Albaida	Deviation (%)	0.68	0.67	0.56	0.59
	# of optima	9	11	9	9
	Time (scs.)	3.33	3.21	4.32	3.47
Madrigueras	Deviation (%)	0.76	0.76	0.74	0.69
	# of optima	0	1	0	1
	Time (scs.)	9.36	9.18	10.59	10.73

(a) Dynamic list

	RefSet size (k-l)	5-5	10-0	10-5	15-0
Albaida	Deviation (%)	0.50	0.53	0.41	0.40
	# of optima	11	11	16	15
	Time (scs.)	3.32	3.23	3.60	3.49
Madrigueras	Deviation (%)	0.66	0.61	0.51	0.49
	# of optima	1	2	3	3
	Time (scs.)	9.60	9.34	10.58	10.22

(b) Static list

Table 5: Scatter Search. Dynamic versus static list

	RefSet size (k-l)	5-5	10-0	10-5	15-0
Albaida	Deviation (%)	0.76	0.86	0.52	0.57
	# of optima	3	2	6	5
	Time (scs.)	14.45	13.26	18.91	17.33
Madrigueras	Deviation(%)	1.95	1.90	1.79	1.77
	# of optima	0	0	0	0
	Time (scs.)	50.65	45.14	55.07	50.95

Table 6: Scatter Search. Alternative combination

instances has been generated from some RPP instances described in Hertz, Laporte & Nanchen (1999). We have used the largest instances proposed in that paper. They all have 100 nodes and correspond to 9 instances in which the set E of edges, all of them with unit cost, forms a grid graph (class G) and to other 9 instances defined on graphs in which almost all nodes have degree 4 (class D). From each of these 18 undirected rural postman instances, 6 WRPP instances were obtained by using the two procedures described in [22] and in [2]. Note that although the original RPP instances have 100 nodes, this number varies in the WRPP generated instances. The reason is that, before generating the WRPP instances, a preprocessing algorithm is applied to the original RPP instances in order to simplify them by removing all those nodes not incident with required edges. This is a well known procedure for arc routing problems (see [4] or [6]).

Finally, two sets of randomly generated instances of larger sizes have also been tried. As with the previous set of instances, different RPP instances are initially built, from which the final WRPP instances are generated. In order to do this, $|V|$ points in a square of size 1000×1000 are first selected. Then $|E|$ edges are randomly generated as pairs of nodes i and j with costs defined by $c_{ij} = \lfloor b_{ij} + \frac{1}{2} \rfloor$, where b_{ij} are the Euclidean distances. This is the same cost function as the one proposed in the TSPLIB (see [21]). If the resulting graph is not connected, edges in 5 different trees spanning the connected components of the graph are also added. Finally, each edge is defined as required with probability P. An RPP instance has been generated for each possible combination of the following parameters: $|V| = 500, 1000$, $|E| = 1.5|V|, 2|V|, 3|V|$ and $P = 0.25, 0.50, 0.75$, producing a total of 18 instances. As in the Hertz et al. set of instances, all the nodes not incident with required edges have been removed by applying the simplification procedure previously mentioned. Then, the number of nodes in the new RPP instances will vary depending on the different sets of required edges randomly generated. From each simplified RPP instance, 3 WRPP instances are created using the first strategy of Win's procedure [22] as follows: For each edge $e = (i, j)$, let k be an integer randomly selected in $[-a, a]$. Set $c'_{ij} = c_{ij} + k$. If $c'_{ij} \leq 0$, set $c'_{ij} = 1$. Select another random integer k and set c'_{ji} in the same way. While in [22] values 5, 8 and 10 for a were used, since in our RPP instances costs c_{ij} are larger, we have used values 50, 80 and 100 for a.

The algorithms have been coded in C and run on a Personal Computer with a 1700 MHz Pentium IV processor. Table 8 shows the computational results obtained by the Scatter Search algorithm with different values for the size of RefSet.

For each set of instances, the table shows the deviation percentage from a lower bound obtained with the cutting plane procedure described in Benavent et al. (2003), the number of optimal solutions found, the average computing time and the average percentage of this time used to generate the initial population P.

Table 8 shows the results of the Scatter Search algorithm with two sizes of RefSet: 10 and 15. Further computational testing was done with RefSet containing a larger number of solutions, but the improvement in the results was too small to justify the big increase in computing times. For each size, two different compositions of the initial RefSet were tried: all the solutions selected from P for quality, or some of them selected to contribute to its diversity.

		$\|V\|$	$\|E\|$	$\|E_R\|$	R-sets
Christofides et al.	Average	25.1	58.9	28.1	4.0
	Min	7	10	4	1
	Max	50	184	78	8
Albaida	Average	116	174	96.3	23
	Min	116	174	83	7
	Max	116	174	129	33
Madrigueras	Average	196	316	172.9	31.4
	Min	196	316	152	5
	Max	196	316	230	47
Hertz et al. (G)	Average	83.4	149.3	77.2	13.3
	Min	60	105	41	4
	Max	100	180	113	20
Hertz et al. (D)	Average	84.6	172.9	85.2	14.1
	Min	68	141	50	9
	Max	100	193	121	22
Random 500	Average	400.8	1268.2	481.3	33.9
	Min	265	842	190	1
	Max	488	1719	770	76
Random 1000	Average	848.3	2521.8	1148.6	49.3
	Min	599	1656	450	1
	Max	988	3952	2229	150

Table 7: Characteristic of the instances

As expected, computational results improve with larger sizes of RefSet, although computing times also increase. Note also that the use of the solutions selected for diversity usually produces better results. More precisely, the use of values (10-5) for the RefSet size seems to be the best option in most cases.

The solutions obtained by our Scatter Search algorithm on the instances of moderate size (Christofides et al.) are very good in terms of deviation percentage, number of optimal solutions obtained and computing time. Concerning the instances of medium size (Albaida, Madrigueras and Hertz et al.), the results show deviation percentages lesser than 1% and low computing times. Finally, the deviation observed on the instances of large size (random instances) are still good (less than 2% from the lower bound) although computing times are quite large. Nevertheless, note that a big percentage of this time is consumed to generate the initial population P. We think that the use of faster constructive algorithms would result in the improvement of the performance of the Scatter Search algorithm.

6 Conclusions

In this paper we have described and tested several heuristic algorithms for the Windy Rural Postman Problem. The WRPP is an interesting problem that generalizes other well known Routing Problems and has interesting real world applications. We have first proposed two new constructive algorithms for the WRPP and discussed several improvement procedures. Some random elements have been introduced in the constructive algorithms to obtain different Multi-Start procedures that have been able to find solutions which are less than 1% far from the lower bounds in few seconds on medium size instances. This deviation percentage has been improved up to 0.5% with a Scatter Search algorithm that, basically, combines pairs of solutions from an initial population generated by one of the Multi-Start procedures. The performance of the Scatter Search algorithm has been extensively tested on a wider set of instances, some of them of large size. It has been able to find solutions with deviations less than 2% from the lower bounds on instances with hundreds of nodes and thousands of edges.

Acknowledgments: The contribution by E. Benavent, A. Corberán, I. Plana and J.M. Sanchis has been partially supported by the Ministerio de Ciencia y Tecnología of Spain through projects TIC2000-C06-01 and HF2001-0071.

References

[1] C. Balaguer, A. Giménez, J.M. Pastor, V.M. Padrón & M. Abderrahim, A climbing autonomous robot for inspection applications in 3D complex environments, *Robotica* 18, 287-297, 2000.

[2] E. Benavent, A. Carrotta, A. Corberán, J.M. Sanchis & D. Vigo, Heuristics and Lower Bounds for the Windy Rural Postman Problem. DEIO Technical Report TR03-2003, March 2003.

	RefSet size (k-l)	5-5	10-0	10-5	15-0
Christofides et al. (144 instances)	Deviation (%)	0.32	0.43	0.27	0.32
	# of optima	106	115	111	110
	Time (scs.)	0.66	0.65	0.71	0.71
	Pop. time (%)	79.90	81.04	75.16	76.57
Albaida (77 instances)	Deviation (%)	0.50	0.53	0.41	0.40
	# of optima	11	11	116	15
	Time (scs.)	3.32	3.23	3.60	3.49
	Pop. time (%)	76.97	78.76	70.68	73.19
Madrigueras (77 instances)	Deviation (%)	0.66	0.60	0.51	0.48
	# of optima	1	2	3	3
	Time (scs.)	9.60	9.34	10.58	10.22
	Pop. time (%)	74.86	77.00	68.10	70.72
Hertz et al. (D) (54 instances)	Deviation (%)	1.16	1.30	0.92	0.91
	# of optima	4	3	5	10
	Time (scs.)	2.25	2.17	2.53	2.45
	Pop. time (%)	70.82	72.79	62.39	63.88
Hertz et al. (G) (54 instances)	Deviation (%)	1.06	1.11	0.69	0.78
	# of optima	12	10	22	18
	Time (scs.)	2.12	1.85	2.20	2.17
	Pop. time (%)	72.96	76.08	64.14	66.84
Random 500 nodes (27 instances)	Deviation (%)	1.79	1.77	1.67	1.69
	# of optima	0	0	0	0
	Time (scs.)	104.86	96.57	153.55	159.35
	Pop. time (%)	72.31	76.68	59.74	59.64
Random 1000 nodes (27 instances)	Deviation (%)	1.70	1.71	1.69	1.78
	# of optima	0	0	0	0
	Time (scs.)	1322.57	1322.73	1811.09	1597.10
	Pop. time (%)	84.73	84.91	72.94	75.76

Table 8: Scatter Search. Static list

[3] P. Brucker, The Chinese Postman Problem for Mixed Graphs. In *Proc. Int. Workshop, Bad Honnef, 1980*. Lecture Notes in Computer Science, 100, 354-366, 1981.

[4] N. Christofides, V. Campos, A. Corberán & E. Mota, An Algorithm for the Rural Postman Problem on a directed graph, *Mathematical Programming Study*, 26, 155-166, 1986.

[5] A. Corberán, R. Martí & J.M. Sanchis, A GRASP heuristic for the Mixed Chinese Postman Problem, *European Journal of Operational Research* 142, 70-80, 2002.

[6] H.A. Eiselt, M. Gendreau & G. Laporte, Arc-Routing Problems, Part 2: the Rural Postman Problem, *Operations Research*, 43, 399-414, 1995.

[7] G.N. Frederickson, Approximation algorithms for some postman problems, *Journal of the ACM*, 26, 538-554, 1979.

[8] F. Glover, M. Laguna & R. Martí, Fundamentals of Scatter Search and Path Relinking, *Control and Cybernetics*, 29, 653-684, 2000.

[9] M. Grötschel & Z. Win, A cutting plane algorithm for the Windy Postman Problem, *Mathematical Programming*, 55, 339-358, 1992.

[10] M. Guan, On the Windy Postman Problem, *Discrete Applied Mathematics*, 9, 41-46, 1984.

[11] A. Hertz, G. Laporte & P. Nanchen, Improvement procedures for the undirected Rural Postman Problem, *INFORMS J. Comput.*, 11, 53-62, 1999.

[12] P. Lacomme, C. Prins & W. Ramdane-Chérif, A Genetic Algorithm for the Capacitated Arc Routing Problem and its extensions. In E.J.W. Boers et al. (Eds.) *Applications of Evolutionary Programming*. Lecture Notes in Computer Science, 2037, 473-483. Springer-Verlag, Berlin, 2001.

[13] P. Lacomme, C. Prins & W. Ramdane-Chérif, Fast Algorithms for General Arc Routing Problems. IFORS 2002, 08-12/07/2002, Edimbourg, UK.

[14] M. Laguna & R. Martí, *Scatter Search: Methodology and Implementations in C*. Kluwer Academic Publishers, 2003.

[15] E. Minieka, The Chinese Postman Problem for Mixed Networks, *Management Science*, 25, 643-648, 1979.

[16] I. Or, Traveling Salesman-Type Combinatorial Problems and their relation to the Logistics of Regional Blood Banking. *PhD Dissertation*, Northwestern University, Evanston, USA, 1976.

[17] V.M. Padrón, Herramientas de planificación de movimientos con restricciones para robots escaladores en estructuras tridimensionales complejas. *PhD Dissertation* (in spanish), Dept. de Ingeniería Eléctrica, Electrónica y Automática, University Carlos III of Madrid, Spain, 2000.

[18] W.L. Pearn & M.L. Li, Algorithms for the Windy Postman Problem, *Computers and Ops. Res.* 21, 641-651, 1994.

[19] B. Raghavachari & J. Veerasamy, Approximation Algorithms for the Mixed Postman Problem. In: Bixby RE, Boyd EA and Ríos-Mercado RZ (Eds.) *Proceedings of 6th Integer Programming and Combinatorial Optimization*, Vol. 1412 of LNCS, Springer-Verlag, 169-179, 1998.

[20] W. Ramdane-Chérif, Problèmes de Tournées sur Arcs. PhD Thesis, University of Troyes, France, 2002 (in French).

[21] G. Reinelt, TSPLIB: a Traveling Salesman Problem Library. *ORSA Journal of Computing* 3, 376-384, 1991.

[22] Z. Win, Contributions to routing problems. *PhD Dissertation*, University of Augsburg, Germany, 1987.

[23] Z. Win, On the Windy Postman Problem on Eulerian Graphs, *Mathematical Programming*, 44, 97-112, 1989.