Volume 21 (1), pp. 33–51
http://www.orssa.org.za

ORiON
ISSN 0529-191-X

Efficient heuristics for the Rural Postman Problem

GW Groves* JH van Vuuren†

Received: 29 July 2004; Revised: 16 December 2004; Accepted: 9 February 2005

Abstract

A local search framework for the (undirected) *Rural Postman Problem* (RPP) is presented in this paper. The framework allows local search approaches that have been applied successfully to the well–known *Travelling Salesman Problem* also to be applied to the RPP. New heuristics for the RPP, based on this framework, are introduced and these are capable of solving significantly larger instances of the RPP than have been reported in the literature. Test results are presented for a number of benchmark RPP instances in a bid to compare efficiency and solution quality against known methods.

Key words: Chinese Postman Problem, Rural Postman Problem, Travelling Salesman Problem.

1 Introduction

Consider a weighted graph $\mathcal{G} = (V, E)$, with vertex set $V = \{v_1, \ldots, v_p\}$, edge set E, and edge weights denoted by $c(i, j)$ for all $v_i v_j \in E$. The well–known *Chinese Postman Problem* (**CPP**) is the problem of determining a minimum–weight closed route traversing each edge $v_i v_j \in E$ at least once (Guan, 1962). The **CPP** is tractable and may be solved in $O(|V|^3)$ time (Edmonds and Johnson, 1973). The *Rural Postman Problem* (**RPP**) is a generalisation of the **CPP** in which a subset of the edges $E_r \subseteq E$ (called *required* edges) have to be traversed. It is the problem of determining a minimum–weight closed route traversing each edge in E_r at least once. The **RPP** is NP–Hard (Lenstra and Rinnooy Kan, 1976), except when $E_r = E$, in which case the problem reduces to the **CPP**. The above **CPP** and **RPP** definitions for undirected graphs have been generalised in many ways, and algorithms catering for directed and mixed graphs, for example, have been introduced — see Ball *et al.* (1995), Dror (2000) and Eiselt *et al.* (1995a, 1995b) for an overview. These versions of the **RPP** have many applications — see, for example, Angel *et al.* (1972), Beltrami and Bodin (1974), Bennet and Gazis (1972), Bodin and Berman (1979), Bodin *et al.* (1989), Bodin and Kursh (1979), Braca *et al.* (1993), Desrosiers *et al.* (1986), Eglese (1994), Eglese and Murdock (1991), Gelders and Cattrysse (1991), Ghiani

*Department of Industrial Engineering, University of Stellenbosch, Private Bag X1, Matieland, 7602, South Africa

†Corresponding author: Department of Applied Mathematics, University of Stellenbosch, Private Bag X1, Matieland, 7602, South Africa, email: vuuren@sun.ac.za

and Improta (2001), Grötschel *et al.* (1991), Levy and Bodin (1988), Roy and Rousseau (1989), Stern and Dror (1979) and Wunderlich *et al.* (1992).

In this paper, local search heuristics are presented for the (undirected) **RPP**. Heuristic methods stand in contrast to exact methods (*i.e.*, algorithms that yield optimal solutions) in that they are designed with the aim of finding good (though not necessarily optimal) solutions without expending much computational execution time. The class of so–called local search heursitics is a family of methods that operates by iteratively performing transformations (referred to as *moves*) to existing solutions to an optimisation problem in a way that tends to (but is not guaranteed to) improve the solution as the search progresses. Typically, a local search heuristic operates by considering a number of candidate moves during each iteration, and selects the best one to perform on the solution. During the next iteration, the process is repeated on the transformed solution, and so on.

Perhaps the best known heuristic for the undirected **RPP** is Frederickson's heuristic (Frederickson, 1979). This heuristic is similar to Christofides' heuristic for the *Travelling Salesman Problem*[1] (**TSP**), and operates by adding artificial edges (representing shortest paths between the relevant vertices in $\mathcal{G}$) to the subgraph induced by E_r in a way that yields a connected, Eulerian graph (*i.e.*, one in which it is possible to find a closed route traversing each edge exactly once). Hertz *et al.* (1999) introduced a family of local search heuristics for the **RPP**, while Fernández de Córdoba *et al.* (1998) employed a heuristic based on Monte Carlo principles. Exact algorithms have been proposed by Christofides *et al.* (1986), Corberan and Sanchis (1991), Ghiani and Laporte (2000), and Letchford (1996). The algorithm by Ghiani and Laporte has been used to solve instances with 350 vertices to optimality, which seem to constitute the largest instances previously addressed by either an exact or heuristic method in the literature. The heuristic presented in this paper is capable of (approximately) solving considerably larger instances of the **RPP**, and the quality of its solutions compare very favourably to those documented in the literature for benchmark problems.

This paper is structured as follows. A local search framework is described in §2, which allows local search moves that have traditionally been applied successfully to *Vertex Routing Problems* (**VRPs**), such as the **TSP**, also to be applied to *Arc Routing Problems* (**ARPs**), such as the **RPP** and the *Capacitated Arc Routing Problem*[2]. This transition is achieved by the introduction of a complexity reduction method (in §2.3) in which both aspects of the *order* (in §2.1) and *direction* (in §2.2) of edge traversals are accommodated. The section is concluded with a worked example, before we report results obtained by the heuristics for a number of benchmark RPP instances in §3. The paper closes with a short conclusion in §4.

[1]The problem of finding a minimum–weight closed route, containing every vertex of a weighted graph.

[2]The problem of finding a set of closed routes of minimum total weight in a weighted graph, satisfying the following conditions: (i) each route starts and ends at a specified vertex, representing a depot, (ii) each edge in E_r is traversed at least once by some route, and (iii) the sum of demands of each route does not exceed vehicle capacity.

2 Local Search Framework

Denote a solution to the **RPP** by the sequence $\mathcal{S} = \langle (v_{s_1}, v_{t_1}), (v_{s_2}, v_{t_2}), \ldots, (v_{s_n}, v_{t_n}) \rangle$ of *required* edges in the order in which they are traversed. Traversals taking place between the required traversals are omitted from the sequence and are assumed to take place along routes corresponding to shortest distances between the required edges of the sequence. The total weight of the route is therefore given by

$$\mathcal{C}(\mathcal{S}) = \sum_{i=1}^{n} c(s_i, t_i) + \sum_{j=1}^{n-1} d(t_j, s_{j+1}) + d(t_n, s_1), \tag{1}$$

where $d(k, \ell)$ denotes the shortest distance between two vertices $v_k, v_\ell \in V(\mathcal{G})$, and $c(i, j)$ is the cost weight associated with the edge $v_i v_j$, as before.

2.1 Applying Local Search Moves

In a local search framework moves are performed on candidate solutions to the **RPP** that directly specify the order in which required edges are traversed in the transformed solution. An example of such a move is one that simply exchanges the order in which two required edges are traversed. Given, for example, the route

$$\mathcal{S}^1 = \langle (3,4), (4,1), (5,6), (5,4), (6,8) \rangle,$$

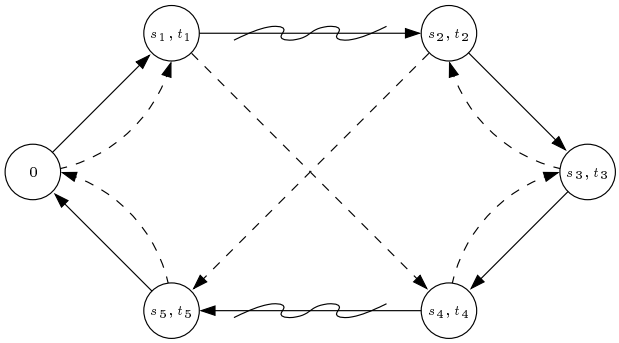
the edges $(3,4)$ and $(5,4)$ might be exchanged, to yield the transformed route

$$\mathcal{S}^{1*} = \langle (5,4), (4,1), (5,6), (3,4), (6,8) \rangle.$$

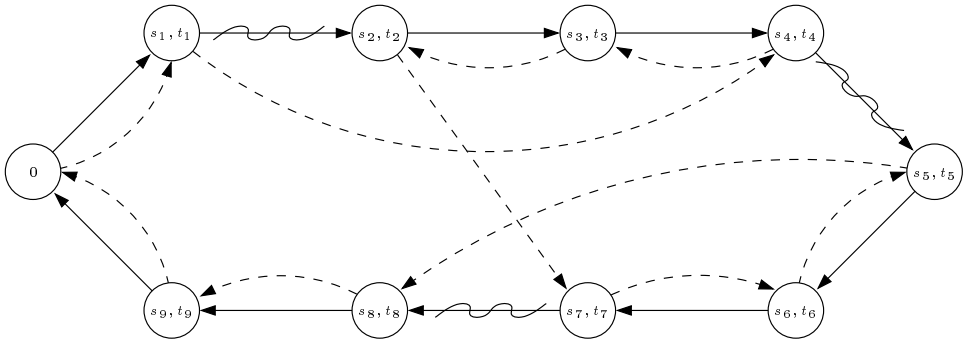
Typically, one would consider all pairs of these exchanges during a single iteration of the search and then perform one that yields a route of minimum overall cost. In the above example, the traversal order of required edges was altered, but not their traversal directions. However, it may be better to traverse the edge $(3,4)$ in $\mathcal{S}^{1*}$, for example, in the direction $(4,3)$ instead of in the direction $(3,4)$. Consequently, it is necessary to determine the optimal directions of traversals of required edges in $\mathcal{S}^{1*}$ after performing an exchange.

Applying a move therefore involves altering the *order* of the required edges in the route, and then determining their *directions of traversal*. This requirement for determining traversal directions results in an increased time complexity, when compared to applying the same type of move to a **VRP**. However, by using a complexity reduction method presented later, it is, in fact, possible to determine the cost of a route without redetermining the traversal directions of *all* of its required edges. This allows for the development of comparatively efficient procedures for many **ARPs**.

The moves considered in this paper for the **RPP** are slightly more involved than the above example, and are based on the well–known *Two–Opt* (Flood, 1956 and Croes, 1958) and *Three–Opt* (Bock, 1958 and Lin, 1965) procedures for the **TSP**. The application of the Two–Opt method is explained with the aid of Figure 1(a), depicting a solution for a small, hypothetical instance of the **RPP**. The vertices in the figure represent the required edges of the problem, and the solid arcs represent shortest paths between vertices incident to



(a) Two–Opt move



(b) Three–Opt move

Figure 1: *Operation of the Two–Opt and Three–Opt move types. Cancelled solid arcs are removed, and the route is reconnected as shown by the dashed lines.*

required edges. The Two–Opt procedure is applied by deleting two of the edges, and re–connecting the route so that it corresponds to the dotted arcs, again using shortest paths between the relevant vertices. Two–Opt is typically implemented as a post–optimisation procedure by performing the best Two–Opt move out of all possible moves, and then repeating this process until no improving solution can be found.

The Three–Opt procedure, depicted in Figure 1(b), is similar to Two–Opt, but three edges are deleted instead and are replaced with shortest paths in a differently connected traversal order. Note that there are eight possible ways of connecting three route segments in this way. Of course the Three–Opt procedure includes Two–Opt as a special case. The vertex 0 in Figure 1 represents a *domicile* vertex, from which a route is considered to start and end, and may represent a depot in a practical context. However, for the **RPP**, in which a specific domicile vertex is not specified, the vertex 0 may be taken as any vertex incident to a required edge.

Returning to our previous small example sequence for the **RPP**, the solution sequence

$$\mathcal{S}^2 = \langle (3,4), (5,4), (5,6), (4,1), (6,8) \rangle$$

is found upon applying the Two–Opt procedure to $\mathcal{S}^1$, deleting the shortest path between the first and second required edges, and between the fourth and the fifth required edges in $\mathcal{S}^1$ and assuming that the traversal directions of the edges are not also reversed. The Three–Opt procedure for the **RPP** may be applied in a similar manner.

However, as mentioned earlier, the optimal traversal directions for the edges also have to be determined after performing the standard Two–Opt or Three–Opt operation.

2.2 Determining Traversal Directions

The algorithm described in this section may be used to compute optimal traversal directions for the edges in a solution sequence $\mathcal{S}$ for the **RPP**, given a fixed order of the edges in the sequence. The algorithm may be applied after each local search move performed in order to yield the smallest cost weight for the new ordering of $\mathcal{S}$.

Consider a routing sequence $\mathcal{S} = \langle (v_{s_1}, v_{t_1}), (v_{s_2}, v_{t_2}), \ldots, (v_{s_n}, v_{t_n}) \rangle$ and construct, from the solution sequence, a directed, layered auxilliary graph $\mathcal{L}$ with vertex set $V(\mathcal{L}) = \{b, s_1t_1, t_1s_1, s_2t_2, t_2s_2, \ldots, s_nt_n, t_ns_n, e\}$. The first and last layer of the auxilliary graph each consists of a single vertex, and the other layers each consist of two vertices. Each internal layer of the graph represents the two possible active traversal directions $v_{s_i}v_{t_i}$ and $v_{t_i}v_{s_i}$ $(0 < i \leq n)$ of an edge in $\mathcal{S}$. The labels b and e represent the vertices at which the route is to begin and end. For a closed route $v_b = v_e$, and in the **RPP** $v_b = v_e$ is incident to an edge in E_r.

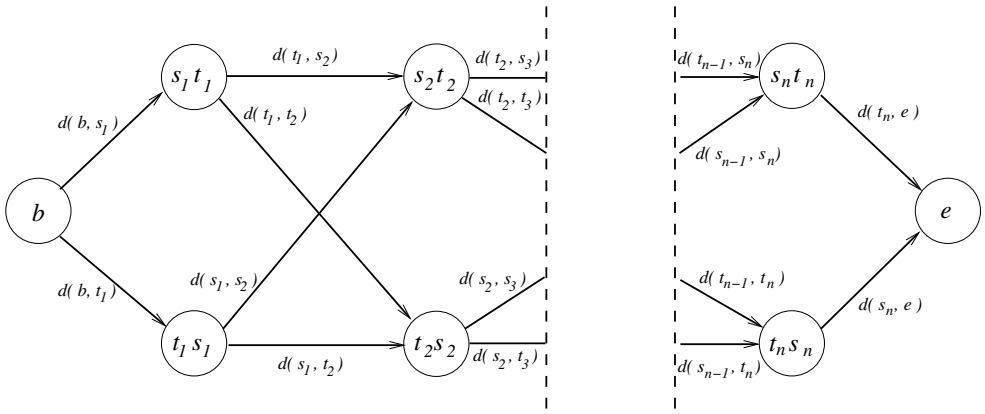


Figure 2: *The layered graph $\mathcal{L}$, corresponding to the solution sequence $\mathcal{S} = \langle (v_{s_1}, v_{t_1}), (v_{s_2}, v_{t_2}), \ldots, (v_{s_n}, v_{t_n}) \rangle$.*

For each i $(0 < i < n)$ construct edges in $\mathcal{L}$ directed from s_it_i to $s_{i+1}t_{i+1}$ & $t_{i+1}s_{i+1}$, and from t_is_i to $s_{i+1}t_{i+1}$ & $t_{i+1}s_{i+1}$. Also add edges directed from b to s_1t_1 & t_1s_1 and from s_nt_n & t_ns_n to e. Assign a weight of $d(t_i, s_{i+1})$ [$d(s_i, t_{i+1})$, respectively] to the edge in $\mathcal{L}$ between s_it_i and $s_{i+1}t_{i+1}$ [t_is_i and $t_{i+1}s_{i+1}$, respectively] for every $0 < i < n$, where $d(i, j)$ denotes the cost of a shortest path from v_i to v_j, as before. Similarly assign a weight of $d(t_i, t_{i+1})$ [$d(s_i, s_{i+1})$, respectively] to the edge in $\mathcal{L}$ between s_it_i and $t_{i+1}s_{i+1}$ [t_is_i and $s_{i+1}t_{i+1}$, respectively] for every $0 < i < n$. Assign a weight of $d(b, s_1)$ [$d(b, t_1)$

respectively] to the edge in $\mathcal{L}$ between b and s_1t_1 [t_1s_1 respectively] and a weight $d(t_n, e)$ [$d(s_n, e)$ respectively] to the edge between s_nt_n [t_ns_n respectively] and e.

Each route from b to e in $\mathcal{L}$ represents one way of arranging the traversal directions of edges within $\mathcal{S}$, and a shortest path from b to e in $\mathcal{L}$ represents a set of optimal directions by which to traverse the edges of $\mathcal{S}$. For example, if vertex t_2s_2 is on a calculated shortest path, then the second edge of $\mathcal{S}$ should be traversed from v_{t_2} to v_{s_2}, and hence coded as (v_{t_2}, v_{s_2}) in $\mathcal{S}$, instead of (v_{s_2}, v_{t_2}). Note that the total weight of a route may be found by adding the sum of the weights of the required edges to the weight of the shortest path.

The computational complexity of finding a shortest path in a directed, acyclic graph, such as $\mathcal{L}$, is $O(E(\mathcal{L})+V(\mathcal{L}))$ (Mehlhorn and Näher, 1999), because no updating of information, such as occurs in Dijkstra's method (Dijkstra, 1959), is necessary during the algorithm execution. However, because no earlier layer of $\mathcal{L}$ can be reached from a later layer, and because each layer consists of a predetermined number of vertices, and is connected to the other layers in the particular manner shown, this complexity may be reduced further to $O(V(\mathcal{L}))$.

The algorithm for computing a shortest path between b and e in $\mathcal{L}$ is a straightforward one, and is given by the pseudo–code listing in Algorithm 1. Let $d_{\mathcal{L}}(i, j)$ be the distance between vertices i and j in the auxilliary graph $\mathcal{L}$, and let $w_{\mathcal{L}}(i, j)$ be the weight of the edge between i and j in $\mathcal{L}$. In the algorithm $pred(i)$ is a variable that stores the predecessor vertex of vertex i in a shortest path from b to e in $\mathcal{L}$.

Algorithm 1 (Shortest path through the layered graph $\mathcal{L}$)
Input: *A layered graph $\mathcal{L}$, as described earlier, with edge weights $w_{\mathcal{L}}(i, j)$ for each edge $ij \in E(\mathcal{L})$.*
Outputs: *(1) The shortest path from b to e in $\mathcal{L}$, stored in the variables $pred(i)$, for all $i \in V(\mathcal{L})$, (2) The shortest distance, $d_{\mathcal{L}}(b, e)$, from b to e in $\mathcal{L}$.*

1. $d_{\mathcal{L}}(b, s_1t_1) \leftarrow w_{\mathcal{L}}(b, s_1t_1)$, $d_{\mathcal{L}}(b, t_1s_1) \leftarrow w_{\mathcal{L}}(b, t_1s_1)$,
   $pred(s_1t_1) \leftarrow b$, $pred(t_1s_1) \leftarrow b$.

2. *For* $i \leftarrow 2, \ldots, n$:

   2a. $val1 \leftarrow d_{\mathcal{L}}(b, s_{i-1}t_{i-1}) + w_{\mathcal{L}}(s_{i-1}t_{i-1}, s_it_i)$,
   $val2 \leftarrow d_{\mathcal{L}}(b, t_{i-1}s_{i-1}) + w_{\mathcal{L}}(t_{i-1}s_{i-1}, s_it_i))$.

   2b. *if* $(val1 < val2)$ *then* $\{d_{\mathcal{L}}(b, s_it_i) \leftarrow val1$, $pred(s_it_i) \leftarrow s_{i-1}t_{i-1}\}$
   else $\{d_{\mathcal{L}}(b, s_it_i) \leftarrow val2$, $pred(s_it_i) \leftarrow t_{i-1}s_{i-1}\}$.

   2c. $val1 \leftarrow d_{\mathcal{L}}(b, s_{i-1}t_{i-1}) + w_{\mathcal{L}}(s_{i-1}t_{i-1}, t_is_i)$,
   $val2 \leftarrow d_{\mathcal{L}}(b, t_{i-1}s_{i-1}) + w_{\mathcal{L}}(t_{i-1}s_{i-1}, t_is_i)$.

   2d. *if* $(val1 < val2)$ *then* $\{d_{\mathcal{L}}(b, t_is_i) \leftarrow val1$, $pred(t_is_i) \leftarrow s_{i-1}t_{i-1}\}$
   else $\{d_{\mathcal{L}}(b, t_is_i) \leftarrow val2$, $pred(t_is_i) \leftarrow t_{i-1}s_{i-1}\}$.

3. $val1 \leftarrow d_{\mathcal{L}}(b, s_nt_n) + w_{\mathcal{L}}(s_nt_n, e)$,
   $val2 \leftarrow d_{\mathcal{L}}(b, t_ns_n) + w_{\mathcal{L}}(t_ns_n, e)$
   if$(val1 < val2)$ *then* $\{d_{\mathcal{L}}(b, e) \leftarrow val1$, $pred(e) \leftarrow s_nt_n\}$
   else $\{d_{\mathcal{L}}(b, e) \leftarrow val2$, $pred(e) \leftarrow t_ns_n\}$. ■

Every vertex of $\mathcal{L}$ (except b) is considered once by the algorithm, and its complexity is therefore $O(V(\mathcal{L}))$. The operation of assigning the computed distance of a vertex (from some other vertex) is referred to as *labelling* in the literature on shortest path algorithms. Each step of the algorithm performs a labelling operation on one or two vertices with respect to vertex b. In the next section, it is argued that the above procedure does not necessarily need to be applied each time that a move is evaluated.

2.3 Reducing Computational Complexity

The task of computing a shortest path in the layered graph in §2.2 is of linear computational complexity. Hence, if the shortest path is computed each time that a move is evaluated, it adds an order of magnitude to the computational complexity of the search heuristic. However, an approach is presented in this section that allows a move to be evaluated without having to re–compute the shortest path through the layered graph, provided that certain shortest path information is known about the untransformed solution (*i.e.*, the solution before the move was implemented).

For the purposes of describing the method of computational complexity reduction, a move performed in a candidate solution $\mathcal{S}$ to obtain a transformed candidate solution $\mathcal{S}^*$ may be viewed as a sequence of z subsequences, containing pairs, given by

$$\begin{aligned}\text{TRANSFORM}(\mathcal{S} \rightarrow \mathcal{S}^*) \;=\; & \langle\, \langle\, (o_1, n_1), \ldots, (o_{l_1}, n_{l_1})\, \rangle_1\ ,\ \langle\, (o_{l_1+1}, n_{l_1+1}), \ldots, (o_{l_2}, n_{l_2})\, \rangle_2, \\ & \ldots\ ,\ \langle\, (o_{l_{z-1}+1}, n_{l_{z-1}+1}), \ldots, (o_{l_z}, n_{l_z})\, \rangle_z\, \rangle,\end{aligned}$$

where each subsequence represents adjacent elements in $\mathcal{S}$ involved in the move. In each pair, o and n represent respectively the old position (in $\mathcal{S}$) and the new position (in $\mathcal{S}^*$) of an edge in $\mathcal{S}$ that had its position changed in the sequence during the move. The edges of each subsequence are adjacent both before and after the move (else they are placed in separate subsequences), and the subsequences are listed in increasing order of the positions of their edges after the move, *i.e.* no subsequence placed later than another will contain a smaller n value than the earlier subsequence, and the pairs within the subsequences themselves are arranged in increasing order of n. To illustrate this notation, consider again our hypothetical solution sequence $\mathcal{S}^1$ in §2.1. Assuming that a move involves moving the second edge to the end of the sequence (which incidentally is one of the moves considered by a Three–Opt procedure), the resulting sequence is

$$\mathcal{S}^3 = \langle (3,4), (5,6), (5,4), (6,8), (4,1) \rangle,$$

and the move may be expressed as

$$\text{TRANSFORM}(\mathcal{S}^1 \rightarrow \mathcal{S}^3) = \langle\, \langle (3,2), (4,3), (5,4) \rangle\ ,\ \langle (2,5) \rangle\, \rangle,$$

where the second subsequence, consisting of a single pair, represents the edge that is moved from position 2 to position 5, as reflected by its entry $(2,5)$. The first subsequence, consisting of 3 pairs, represents the edges that move to the left to make place for the edge that moved to the end.

The complexity reduction method is based on the observation that edges of the same subsequence will retain their distances with respect to each other during a move. Note, however,

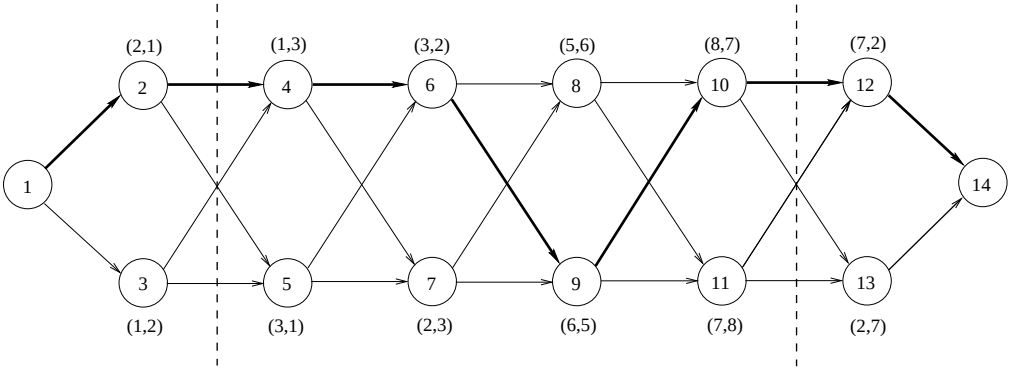


Figure 3: *The layered graph,* $\mathcal{L}^4$*, corresponding to the solution sequence* $\mathcal{S}^4 = \langle(2,1),(1,3),(3,2),(5,6),(8,7),(7,2)\rangle$*. A shortest path from the first to the last layer is shown using bold edges.*

that when a subsequence describes edges that are reversed in the transformed sequence, the shortest paths take place along routes that are perhaps not as intuitive as in the case where the edges are not reversed. Consider, for example, the layered graph $\mathcal{L}^4$, corresponding to a hypothetical solution sequence $\mathcal{S}^4 = \langle(2,1),(1,3),(3,2),(5,6),(8,7),(7,2)\rangle$, shown in Figure 3. The bold edges in the figure indicate a shortest path (and optimal traversal directions) of $\mathcal{S}^4$. Consider the Two–Opt move TRANSFORM($\mathcal{S}^4 \rightarrow \mathcal{S}^{4*}$) = $\langle\ \langle(5,2),(4,3),(3,4),(2,5)\rangle\ \rangle$, which reverses the segment between (and including) the second and fifth edges in $\mathcal{S}^4$. The layered graph of the transformed solution, $\mathcal{L}^{4*}$, is shown in Figure 4. A shortest path between vertices 5 and 11 of $\mathcal{L}^{4*}$ (shown as bold edges) has the same length as the shortest path from vertex 4 to 10 in $\mathcal{L}^4$. Similarly, the distance from 4 to 10 in $\mathcal{L}^{4*}$ equals the distance from 5 to 11 in $\mathcal{L}^4$. The distance from 4 to 11 [5 to 10, respectively] in $\mathcal{L}^{4*}$ equals the distance from 4 to 11 [5 to 10, respectively] in $\mathcal{L}^4$.

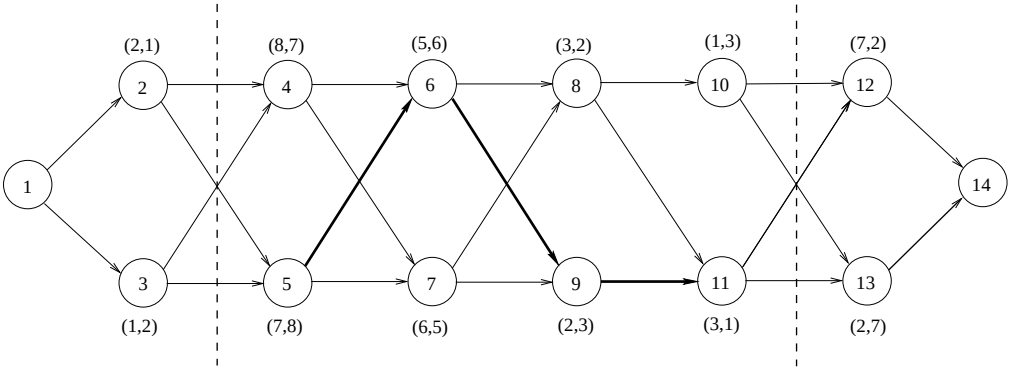


Figure 4: *The layered graph* $\mathcal{L}^{4*}$ *of the transformed solution sequence* $\mathcal{S}^{4*}$*. The path indicated by the bold edges, between vertices* 5 *and* 11*, represents the same path as the path between vertices* 4 *and* 10 *in Figure 3.*

The complexity reduction method is now described in general, using the alternative nota-

tion for a move introduced earlier in the section. The method is described in some detail in the pseudo–code listing below, and may be read in conjunction with subsequent example, provided after the algorithm listing, to simplify understanding. Define *label*($\mathcal{L}$,i,j) as a function that performs the labelling operation of a vertex j (with respect to a vertex i) in order to yield $d_{\mathcal{L}}(i,j)$ (see §2.2). Each iteration of a local search procedure proceeds as listed in Algorithm 2.

Algorithm 2 (Local search iteration using the complexity reduction method)
***Input**: A solution sequence $\mathcal{S}$ with corresponding layered graph $\mathcal{L}$, as described in §2.2.*
***Output**: A transformed solution sequence, with the minimum total cost out of all the moves considered.*

1. *Calculate, in $\mathcal{L}$, the shortest distance from all vertices to e (the distances from each of these vertices to all vertices in later layers becomes known).*
2. *For each possible move (denote the transformed solution $\mathcal{S}^*$, and its auxilliary graph $\mathcal{L}^*$):*

   2.1 Set $d_{\mathcal{L}^*}(b, s_{n_1-1}t_{n_1-1}) \leftarrow d_{\mathcal{L}}(b, s_{n_1-1}t_{n_1-1})$, $d_{\mathcal{L}^*}(s, t_{n_1-1}s_{n_1-1}) \leftarrow d_{\mathcal{L}}(b, t_{n_1-1}s_{n_1-1})$

   2.2 For each subsequence $k \leftarrow 1, \ldots, z$ *in TRANSFORM($\mathcal{S} \rightarrow \mathcal{S}*$):*

      2.2.1 Assume $l_0 = 0$, $n_0 = 0$; *label*$(\mathcal{L}^*, b, s_{n_{l_{k-1}+1}}t_{n_{l_{k-1}+1}})$, *label*$(\mathcal{L}^*, b, t_{n_{l_{k-1}+1}}s_{n_{l_{k-1}+1}})$

      2.2.2 If $n_{l_k} > n_{l_{k-1}+1}$ *(i.e., more than one pair in subsequence) calculate* $d_{\mathcal{L}^*}(b, s_{n_{l_k}}t_{n_{l_k}})$ *and* $d_{\mathcal{L}^*}(b, t_{n_{l_k}}s_{n_{l_k}})$ *as follows: (else they are already known)*

      If $o_{l_k} > o_{l_k-1}$ (i.e., *if the edges of subsequence are not reversed):*

$$d_{\mathcal{L}^*}(b, s_{n_{l_k}}t_{n_{l_k}}) \leftarrow min\begin{cases} d_{\mathcal{L}^*}(b, s_{n_{l_{k-1}+1}}t_{n_{l_{k-1}+1}}) + d_{\mathcal{L}}(s_{o_{l_{k-1}+1}}t_{o_{l_{k-1}+1}}, s_{o_{l_k}}t_{o_{l_k}}) \\ d_{\mathcal{L}^*}(b, t_{n_{l_{k-1}+1}}s_{n_{l_{k-1}+1}}) + d_{\mathcal{L}}(t_{o_{l_{k-1}+1}}s_{o_{l_{k-1}+1}}, s_{o_{l_k}}t_{o_{l_k}}) \end{cases}$$

$$d_{\mathcal{L}^*}(b, t_{n_{l_k}}s_{n_{l_k}}) \leftarrow min\begin{cases} d_{\mathcal{L}^*}(b, s_{n_{l_{k-1}+1}}t_{n_{l_{k-1}+1}}) + d_{\mathcal{L}}(s_{o_{l_{k-1}+1}}t_{o_{l_{k-1}+1}}, t_{o_{l_k}}s_{o_{l_k}}) \\ d_{\mathcal{L}^*}(b, t_{n_{l_{k-1}+1}}s_{n_{l_{k-1}+1}}) + d_{\mathcal{L}}(t_{o_{l_{k-1}+1}}s_{o_{l_{k-1}+1}}, t_{o_{l_k}}s_{o_{l_k}}) \end{cases}$$

      *else (*i.e., *edges of subsequence are reversed):*

$$d_{\mathcal{L}^*}(b, s_{n_{l_k}}t_{n_{l_k}}) \leftarrow min\begin{cases} d_{\mathcal{L}^*}(b, s_{n_{l_{k-1}+1}}t_{n_{l_{k-1}+1}}) + d_{\mathcal{L}}(t_{o_{l_k}}s_{o_{l_k}}, t_{o_{l_{k-1}+1}}s_{o_{l_{k-1}+1}}) \\ d_{\mathcal{L}^*}(b, t_{n_{l_{k-1}+1}}s_{n_{l_{k-1}+1}}) + d_{\mathcal{L}}(t_{o_{l_k}}s_{o_{l_k}}, s_{o_{l_{k-1}+1}}t_{o_{l_{k-1}+1}}) \end{cases}$$

$$d_{\mathcal{L}^*}(b, t_{n_{l_k}}s_{n_{l_k}}) \leftarrow min\begin{cases} d_{\mathcal{L}^*}(b, s_{n_{l_{k-1}+1}}t_{n_{l_{k-1}+1}}) + d_{\mathcal{L}}(s_{o_{l_k}}t_{o_{l_k}}, t_{o_{l_{k-1}+1}}s_{o_{l_{k-1}+1}}) \\ d_{\mathcal{L}^*}(b, t_{n_{l_{k-1}+1}}s_{n_{l_{k-1}+1}}) + d_{\mathcal{L}}(s_{o_{l_k}}t_{o_{l_k}}, s_{o_{l_{k-1}+1}}t_{o_{l_{k-1}+1}}) \end{cases}$$

      2.2.3 If ($k < z$ AND $n_{l_k+1} - n_{l_k} > 1$) OR ($k = z$ AND $n_{l_z} < |S|$):

$$label(\mathcal{L}^*, b, s_{n_{l_k}+1}t_{n_{l_k}+1}),\ label(\mathcal{L}^*, b,\ t_{n_{l_k}+1}s_{n_{l_k}+1}).$$

   2.3 If $n_{l_z} < |S|$*:*

$$d_{\mathcal{L}^*}(b, e) \leftarrow min\begin{cases} d_{\mathcal{L}^*}(b, s_{n_{l_z}+1}t_{n_{l_z}+1} + d_{\mathcal{L}}(s_{n_{l_z}+1}t_{n_{l_z}+1}, t) \\ d_{\mathcal{L}^*}(b, t_{n_{l_z}+1}s_{n_{l_z}+1} + d_{\mathcal{L}}(t_{n_{l_z}+1}s_{n_{l_z}+1}, t) \end{cases}$$

   else: label$(\mathcal{L}^*, b, e)$.

3 *Select the move for which* $d_{\mathcal{L}^*}(b, e)$ *is a minimum and perform this move on* $\mathcal{S}$. ■

Note that Algorithm 2 does not apply any of the moves during step 2, but computes what the total route cost would be if they were to be applied. The sequence $\mathcal{S}^*$ is therefore not seen as an existing sequence, but rather as one that may be inferred from TRANSFORM($\mathcal{S} \rightarrow \mathcal{S}^*$).

The computational complexity of step 1 is $O(|V(\mathcal{L})|^2)$. The complexity of step 2 depends on two factors, the first of which is the number of moves evaluated and the second, the time taken to evaluate a move. Using Two–Opt and Three–Opt, for example, there are respectively $O(|\mathcal{S}|^2)$ and $O(|\mathcal{S}|^3)$ moves to be evaluated. The method takes $O(z)$ time to evaluate a move, and the complexity of step 3 is $O(|\mathcal{S}|)$. Therefore, the worst–case computational complexity of using Two–Opt and Three–Opt in the pseudo–code algorithm shown, is respectively $O(|\mathcal{S}|^2)$ and $O(|\mathcal{S}|^3)$. The average–case execution time of the procedures are of the same orders of magnitude.

Finally, although it was stated earlier that TRANSFORM($\mathcal{S} \rightarrow \mathcal{S}^*$) describes only edges that have their position changed in the transformed solution, in one particular case an edge that does not have its positions changed may be included in the notation to simplify the execution of the complexity reduction method. Consider the Two–Opt move TRANSFORM($\mathcal{S}^5 \rightarrow \mathcal{S}^{5*}$) $= \langle\ \langle(5,1),(4,2)\rangle\ \langle(2,4),\ (1,5)\rangle\ \rangle$ performed on a hypothetical solution sequence, $\mathcal{S}^5$. This move could also be encoded as TRANSFORM($\mathcal{S}^5 \rightarrow \mathcal{S}^{5*}$) $= \langle\ \langle(5,1),(4,2),(3,3),(2,4),\ (1,5)\rangle\ \rangle$. The inclusion of the pair $(3,3)$ does not influence the result of the complexity reduction method, but improves its execution time slightly, because the method evaluates fewer subsequences. Therefore, in a move where a sequence consisting of an odd number of edges is reversed, without otherwise moving their positions in the sequence, it is more efficient to include, in the notation TRANSFORM($\mathcal{S} \rightarrow \mathcal{S}^*$), the edge that does not have its position changed. Nevertheless, in a computer implementation, this is not a practical concern, because a move would not be encoded as in TRANSFORM($\mathcal{S} \rightarrow \mathcal{S}^*$) during execution, since this would take $O(|\mathcal{S}|)$ time to encode. Rather, the computer program would keep track of the untransformed and transformed positions of the endpoints of each subsequence, and in doing so, would automatically ensure this benefit. The operation of the complexity reduction method is now illustrated by means of example.

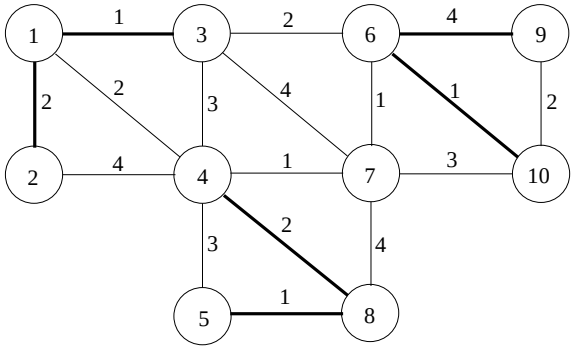


Figure 5: *An example* **RPP** *problem instance. Edge cost weights are shown next to each edge, and required edges are denoted in bold face.*

Example 1 *The complexity reduction method is demonstrated in this example by applying it to the graph in Figure 5, in which the required edges are denoted in bold face. Assume that an initial solution to this instance has been determined, and is given by*

$$\mathcal{S}_0 = \langle (1,3),(4,8),(8,5),(6,9),(10,6),(2,1)\rangle.$$

The corresponding layered graph, denoted $\mathcal{L}_0$*, is depicted in Figure 6. Its vertex set is given by* $V(\mathcal{L}_0) = \{1,\ldots,2\times 6+2=14\}$*. The cost of* $\mathcal{S}_0$ *may be determined by adding the sum of the cost weights of the required edges (i.e., the value* 11*) to* $d_{\mathcal{L}_0}(1,14) = 15$ *to yield a total cost of* 26*.*

The evaluation of a single move by the complexity reduction method is described next. The move under consideration is the Two–Opt move $\textsc{transform}(\mathcal{S}_0 \to \mathcal{S}_0^*) = \langle\ \langle (5,2),(4,3),(3,4),(2,5)\rangle\ \rangle$*, in which the sequence of edges between the first and the last edges in* $\mathcal{S}_0$ *are reversed. The resulting layered graph, as it would appear if the move were applied, is shown in Figure 7. Note that the edges of this graph are omitted for the sake of clarity.*

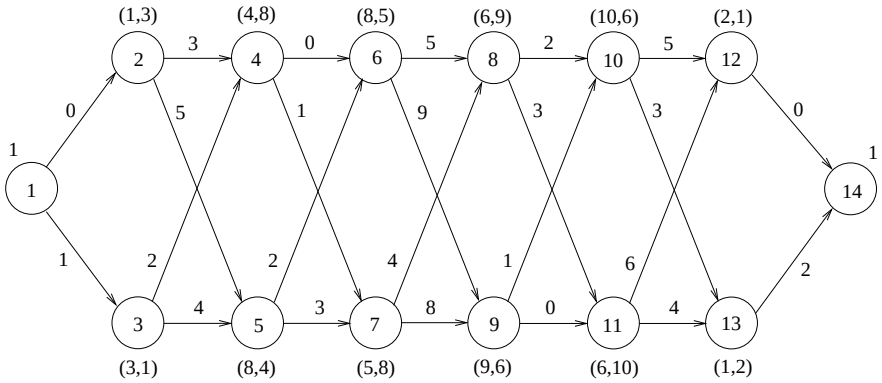


Figure 6: *The layered graph,* $\mathcal{L}_0$*, corresponding to the initial solution seqence* $\mathcal{S}_0 = \langle (1,3),(4,8),(8,5),(6,9),(10,6),(2,1)\rangle$ *of the* **RPP** *instance in Example 1.*

The relevant cost labels required to compute the route weight are shown on the graph, as computed by the complexity reduction method. The computations performed in step 2 of the complexity reduction method proceed as follows:

2.1 Set $d_{\mathcal{L}_0^*}(1,2) \leftarrow d_{\mathcal{L}_0}(1,2) = 0$*,* $d_{\mathcal{L}_0^*}(1,3) \leftarrow d_{\mathcal{L}_0}(1,3) = 1$*.*

2.2 For the only subsequence $k \leftarrow 1$ *in* $\textsc{transform}(\mathcal{S}_0)$*:*

2.2.1 $\textsc{label}(\mathcal{L}_0^*,1,4)$*,* $\textsc{label}(\mathcal{L}_0^*,1,5)$ *(yielding* $d_{\mathcal{L}_0^*}(1,4) = 3$*,* $d_{\mathcal{L}_0^*}(1,5) = 2$*).*

2.2.2 $n_{l_1} = 5 > 2 = n_{l_0+1}$*, therefore* $d_{\mathcal{L}_0^*}(1,10)$ *and* $d_{\mathcal{L}_0^*}(1,11)$ *are computed in this step.* $o_{l_1} = 2 < 3 = o_{l_1-1}$ *(i.e. edges of subsequence reversed):*

$$d_{\mathcal{L}_0^*}(1,10) \leftarrow \mathit{min}\left\{\begin{array}{l} d_{\mathcal{L}_0^*}(1,4) + d_{\mathcal{L}_0}(5,11) = 3+10 \\ d_{\mathcal{L}_0^*}(1,5) + d_{\mathcal{L}_0}(5,10) = 3+8 \end{array}\right\} = 11$$

$$d_{\mathcal{L}_0^*}(1,11) \leftarrow \mathit{min}\left\{\begin{array}{l} d_{\mathcal{L}_0^*}(1,4) + d_{\mathcal{L}_0}(4,11) = 3+8 \\ d_{\mathcal{L}_0^*}(1,5) + d_{\mathcal{L}_0}(4,10) = 2+7 \end{array}\right\} = 9.$$

2.2.3 $k = z = 1$ *AND* $n_{l_z} = 5$*, therefore:*
$label(\mathcal{L}_0^*, 1, 12)$*,* $label(\mathcal{L}_0^*, 1, 13)$ *(yielding* $d_{\mathcal{L}_0^*}(1, 12) = 13$*,* $d_{\mathcal{L}_0^*}(1, 13) = 11$*).*

2.3 $n_{l_z} = 5 < 6 = |\mathcal{S}|$*, therefore:*

$$d_{\mathcal{L}_0^*}(1, 14) \leftarrow min \left\{ \begin{array}{l} d_{\mathcal{L}_0^*}(1, 12) + d_{\mathcal{L}_0}(12, 14) = 13 + 0 \\ d_{\mathcal{L}_0^*}(1, 13) + d_{\mathcal{L}_0}(13, 14) = 11 + 2 \end{array} \right\} = 13.$$

The total cost of the solution after performing the move would be 24 *(i.e., the sum of the cost weights of the required edges,* 11*, added to* $d_{\mathcal{L}_0^*}(1, 14) = 13$*). The example illustrates how the total cost of the route may be found without actually performing the move or computing an entire shortest path from vertex* 1 *to* 14*. Note also that the optimal traversal directions need not be redetermined when the complexity reduction method is used.* ■

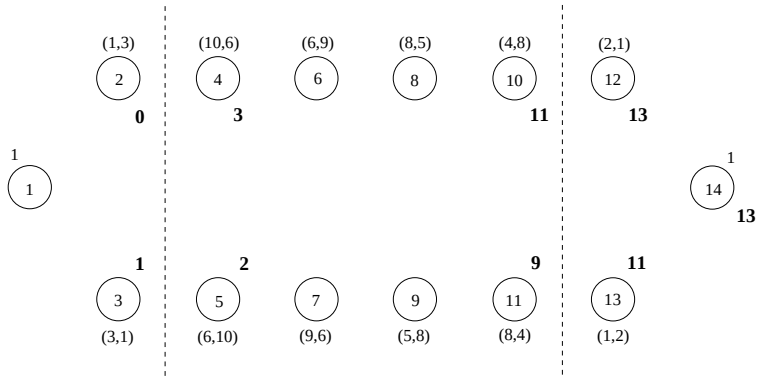


Figure 7: *The layered graph* $\mathcal{L}_0^*$ *obtained by performing the move* $TRANSFORM(\mathcal{S}_0 \rightarrow \mathcal{S}_0^*) = \big\langle\langle(5, 2), (4, 3), (3, 4), (2, 5)\rangle\big\rangle$*. The labels calculated by the complexity reduction procedure are shown as bold numbers beside the relevant vertices.*

3 Computational Results for the RPP

The results obtained from applying the Two–Opt and Three–Opt heuristics (described in §2) to benchmark problems and new test data are presented in this section and contrasted with results previously obtained by others.

3.1 Details of Heuristics

A Two–Opt and Three–Opt procedures, described in §2.1, were used to arrive at the results presented in this section. The heuristic of Frederickson (1979), of complexity $O(|V|^3)$, was used to generate starting solutions for the Two–Opt and Three–Opt procedures, and the method described in §2.3 was used to reduce the complexity of each iteration of Two–Opt procedure [Three–Opt procedure, respectively] to $O(|E_r|^2)$ [$O(|E_r|^3)$, respectively]. A one–dimensional array of integer values was used to represent the order in which the edges of the route are traversed, and the Two–Opt and Three–Opt procedures were performed on this array.

3.2 Benchmark Graph Instances

The results obtained from applying the Two–Opt and Three–Opt heuristics to a set of benchmark test instances are shown in Table 1. The instances numbered 1 to 24 in Table 1 are the instances used by Christofides *et al.* (1986), and the other two instances are two of the so–called *Albaida* instances[3] used by Corberán and Sanchis (1991). Hertz *et al.* (1999) and Fernández de Córdoba *et al.* (1998) present results for the same data. Note that the execution times listed in the columns for the Two–Opt and Three–Opt procedures of Table 1 are the total execution time, including the time taken to generate an initial solution.

				Optimal	Frederickson		Two–Opt		Three–Opt	
Instance	$\|V\|$	$\|E\|$	$\|E_r\|$	cost	Cost	Time	Cost	Time	Cost	Time
1	11	13	7	76	76	0	76	0	76	0
2	14	33	12	152	155	0	153	0	152	0
3	28	58	26	102	105	0	103	0	103	0
4	17	35	22	84	84	0	84	0	84	0
5	20	35	16	124	130	0	124	0	124	0
6	24	46	20	102	107	0	107	0	102	0
7	23	47	24	130	130	0	130	0	130	0
8	17	40	24	122	122	0	122	0	122	0
9	14	26	14	83	83	0	83	0	83	0
10	12	20	10	80	80	0	80	0	80	0
11	9	14	7	23	26	0	23	0	23	0
12	7	18	5	19	22	0	19	0	19	0
13	7	10	4	35	35	0	35	0	35	0
14	28	79	31	202	207	0	204	0	202	0
15	26	37	19	441	445	1	441	0	441	0
16	31	94	34	203	215	0	205	0	203	0
17	19	44	17	112	116	0	112	0	112	0
19	33	55	29	257	274	0	271	0	266	0
20	50	98	63	398	402	0	400	0	400	1
21	49	110	67	366	372	0	372	0	372	0
22	50	184	74	621	633	0	622	0	622	0
23	50	158	78	475	479	0	477	0	477	2
24	41	125	55	405	411	0	405	0	405	0
AlbA	102	160	99	10 599	10 599	0	10 599	0	10 599	2
AlbB	90	144	88	8 629	8 629	0	8 629	0	8 629	1

Table 1: *Results for* **RPP** *benchmark test instances. Execution times are measured in seconds and were achieved on an Intel Pentium IV processor (2.8 GHz) with 512 Mb memory.*

Note also that instance 18 of the Christofides problems is omitted from this analysis due to inconsistencies in its reported optimal value. Hertz *et al.* (1999) report that the optimal solution for this graph instance has a cost weight of 147 and that a solution of this cost is found by their heuristics, while Fernández de Córdoba *et al.* (1998) report that it is 148 and that their procedure also finds a solution of this cost. However, the heuristics described in §2.1 and §2.3 found a solution of cost 146. A manual examination of the route confirms that a feasible solution of cost 146 exists.

[3] Albaida is a Spanish town, on whose streets the graphs are based.

Frederickson's heuristic found an optimal solution 9 times out of the 25 instances considered. When the Two–Opt procedure was also applied, this number increased to 15. When the Three–Opt procedure was used in conjunction with Frederickson's heuristic instead, an optimal solution was found 19 times out of 25. The heuristic of Fernández de Córdoba *et al.* (1998) found an optimal solution 11 times out of 25, while the Two–Opt based heuristic of Hertz *et al.* (1999) found an optimal solution 22 times out of 25. However, the high computational complexity of the Hertz *et al.* (1999) Two–Opt heuristic ($O(|E|^5)$ per iteration) and the fact that actual running times closely match theoretical worst–case times in exchange–based moves (because all exchanges are considered), limits its applicability to small problem instances, in our opinion.

3.3 Graph Instances with Euclidean and Random Edge Weights

The graph instances of Table 1 are clearly well within the capabilities of all the heuristics considered, and two larger sets of data were therefore generated, in order to establish bounds of practical feasiblity for the heuristics, in terms of problem instance sizes that could be considered. The results of applying the heuristics to these larger instances are shown in Table 2.

The instances of the first data set in Table 2(a) were generated with Euclidean edge weights[4], and the instances of the data set in Table 2(b) with random edge weights[5]. The dimensions of the graphs were chosen uniformly in the ranges $500 \leq |V| \leq 1\,000$ and $2\,500 \leq |E| \leq 3\,500$, and the number of required edges in each graph (*i.e.*, $|E_r|$) was set to approximately 10% of the size of the graph. The edges designated as *required* were selected iteratively — a random as yet *non–required* edge was chosen and then designated as *required* only if the subgraph induced by the required edges would be disconnected if it were to be chosen. The process was repeated until the desired number of required edges had been selected. The graphs were generated according to the procedure outlined in Algorithm 3.

Algorithm 3 (Generate Random Graph Instance)
Inputs: *Order p and size $p-1 \leq q \leq \binom{p}{2}$ of problem instance.*
Output: *Random instance of a connected graph $\mathcal{G}$, of order p and size q.*

1. *Initialise the graph $\mathcal{G}$ to contain p vertices (and no edges).*
2. *Select two different components of $\mathcal{G}$ at random, and add an edge between a randomly chosen vertex from each component. Repeat this step until $\mathcal{G}$ is connected.*
3. *While $\mathcal{G}$ has fewer than q edges, add an edge between two randomly chosen vertices not yet joined by an edge.*
4. *Output $\mathcal{G}$.* ■

The algorithm used to determine the lower bound values in Table 2 operates in a similar way to the well–known Edmonds & Johnson algorithm for the **CPP**. A minimum cost maximum cardinality matching is performed on the odd–degree vertices of the subgraph induced by the required edges (Edmonds and Johnson, 1973).

[4]The vertices of each graph were each randomly assigned a position in a unit square (according to a uniform distribution) and the edge weights were set equal to the value obtained by multiplying the Euclidean distance between vertices by 1 000 and rounding the result to the nearest integer.

[5]The edge weights of the random instances were uniformly chosen integer values between 1 and 1 000.

				Lower	Frederickson		Two–Opt		Three–Opt	
Instance	$\|V\|$	$\|E\|$	$\|E_r\|$	bound	Cost	Time	Cost	Time	Cost	Time
1	691	3 163	316	218 119	225 662	14	222 787	14	221 170	427
2	550	2 929	293	184 325	189 387	10	187 192	10	186 838	228
3	798	3 035	304	216 423	232 963	22	227 112	24	22 5146	658
4	949	2 964	296	236 528	266 009	36	254 086	38	252 358	539
5	942	2 560	256	205 765	233 626	35	225 755	36	221 497	656
6	979	2 750	275	222 316	252 088	38	239 067	41	236 606	844
7	507	3 368	337	204 543	209 001	8	208 157	8	207 478	225
8	703	3 086	309	199 995	211 372	17	207 067	19	204 870	641
9	931	3 299	330	246 775	269 025	34	261 619	37	257 978	989
10	569	2 809	281	193 185	200 594	10	198 415	10	196 181	327
11	622	2 602	260	188 206	198 362	21	193 000	23	192 659	347
12	522	3 444	344	208 101	210 810	8	209 970	8	209 235	485
13	516	3 182	318	196 353	201 176	8	198 980	9	198 660	300
14	582	2 910	291	191 034	199 944	10	197 222	11	195 900	328
15	609	3 193	319	218 343	227 234	11	223 737	13	221 830	626
16	508	2 562	256	166 198	170 718	6	168 878	7	168 178	106
17	642	3 426	343	226 939	232 660	12	230 594	13	229 444	659
18	671	2 528	253	182 133	197 226	14	191 998	15	189 767	335
19	776	2 835	284	206 621	226 058	21	220 830	21	217 659	471
20	678	3 499	350	221 710	230 688	15	226 875	18	225 473	1 119

(a) Euclidean Distances

				Lower	Frederickson		Two–Opt		Three–Opt	
Instance	$\|V\|$	$\|E\|$	$\|E_r\|$	bound	Cost	Time	Cost	Time	Cost	Time
1	967	3 119	312	214 326	227 852	36	221 246	39	219 664	856
2	929	2 783	278	187 957	201 579	33	197 444	34	193 954	881
3	599	2 569	257	148 932	153 963	10	151 811	11	151 036	284
4	550	3 303	330	184 770	186 243	8	185 855	9	185 831	143
5	685	3 053	305	179 895	185 820	15	184 144	16	183 750	475
6	955	2 752	275	210 372	227 992	35	223 118	37	221 149	503
7	944	2 880	288	198 584	215 513	35	209 089	38	207 532	758
8	838	3 115	312	206 151	215 805	40	211 609	44	211 198	690
9	536	3 181	318	180 559	182 763	16	182 137	18	181 497	387
10	673	3 282	328	186 293	189 757	29	187 760	33	187 461	602
11	707	2 735	274	171 334	177 479	31	175 019	34	174 389	413
12	607	2 642	264	162 047	166 290	21	163 926	23	163 265	277
13	640	3 028	303	177 381	179 896	24	178 605	26	178 043	471
14	883	3 343	334	219 426	229 799	60	226 281	61	225 202	748
15	763	3 098	310	182 712	189 261	39	185 883	43	185 685	603
16	648	2 882	288	169 575	175 739	25	173 387	27	173 321	314
17	617	3 427	343	204 898	208 236	24	206 502	28	206 486	635
18	976	3 451	345	221 410	233 249	78	227 217	84	226 224	1 100
19	919	2 856	286	197 571	208 484	63	205 113	66	204 361	479
20	831	3 359	336	215 776	222 990	50	219 805	54	219 115	772

(b) Random Distances

Table 2: *Test results for two new* **RPP** *data sets, one consisting of Euclidean edge weights, and the other with random edge weights. Execution times are measured in seconds and were achieved on an Intel Pentium IV processor (2.8 GHz) with 512 Mb memory.*

Data Set	Frederickson	Two–Opt	Three–Opt
Euclidean	5.9%	3.8%	3.0%
Random	4.1%	2.5%	2.0%

Table 3: *Average percentage–gap–over–lower bound values obtained by the* **RPP** *heuristics on the test problem instances of Table 2.*

The performance of the heuristics, in relation to the lower bound values, are summarised in Table 3. The Two–Opt procedure is typically able to reduce the gap over lower bound values obtained by Frederickson's heuristic by a further $1.5 - 2.0\%$. The Three–Opt procedure is able to reduce this gap to approximately $2 - 3\%$.

3.4 Large Problem Instances

Computational results for a set of large test problems are shown in Table 4. These results were obtained using the Two–Opt heuristic. The instances of this table were generated in the same manner as those in Table 2, and were assigned random Euclidean edge weights. The order and size of each graph were chosen uniformly from the ranges $3\,000 \leq |V| \leq 5\,000$ and $20\,000 \leq |E| \leq 30\,000$ respectively.

				Lower	Frederickson		Two–Opt	
Instance	$\|V\|$	$\|E\|$	$\|E_r\|$	bound	Cost	Time	Cost	Time
1	4 462	27 506	2 751	1 729 797	1 764 008	3 718	1 752 799	4 339
2	3 130	22 342	2 234	1 363 165	1 374 875	1 270	1 369 920	1 518
3	4 555	24 398	2 440	1 577 885	1 628 666	3 859	1 610 481	4 622
4	4 592	22 492	2 249	1 502 920	1 564 479	3 945	1 539 635	4 852
5	4 740	28 547	2 855	1 809 672	1 846 532	4 342	1 833 930	5 079
6	4 927	25 007	2 501	1 632 439	1 692 323	4 836	1 669 227	5 901
7	3 260	29 835	2 984	1 743 054	1 749 300	1 508	1 747 615	1 641
8	4 221	23 490	2 349	1 497 199	1 536 811	3 014	1 523 367	3 568
9	4 519	20 839	2 084	1 413 368	1 480 931	3 678	1 458 962	4 288
10	3 908	20 660	2 066	1 326 576	1 373 923	3 365	1 353 635	4 538

Table 4: *Test results for an* **RPP** *data set consising of large graphs. Execution times are measured in seconds and were achieved on an Intel Pentium IV processor (2.8 GHz) with 512 Mb memory.*

The instances of Table 4 are the largest known **RPP** instances to which either a heuristic or exact method have been applied successfully. The largest graphs previously reported seem to be those of Ghiani and Laporte (2000), consisting of 350 vertices and an average of 1370 edges (of which approximately 140 are required edges).

4 Conclusion

New heuristics for the Rural Postman Problem (**RPP**) were introduced in this paper. The heuristics are based on a local search framework that can also be used to optimise routes in other types of arc routing problems (*e.g.*, the *Capacitated Arc Routing Problem*), and seem to consistute the best heuristic approach currently available for the **RPP**.

Acknowledgements

The authors are grateful to Prof Angel Corberán of the University of Valencia for providing the benchmark **RPP** graph instances used in this paper. Work towards this paper was supported by the South African National Research Foundation under grant number GUN 2053755 and Research Sub–Committee B at the University of Stellenbosch.

References

[1] Angel RD, Caulde WL, Noonan R & Whinston A, 1972, *Computer–assisted school bus scheduling*, Management Science, **B18**, pp. 279–288.

[2] Ball MO, Magnanti TL, Monma CL & Nemhauser GL (Eds.), 1995, *Network routing*, North-Holland, Amsterdam.

[3] Beltrami EL & Bodin LD, 1974, *Networks and vehicle routing for municipal waste collection*, Networks, **4**, pp. 65–94.

[4] Bennett B & Gazis D, 1972, *School bus routing by computer*, Transportation Research, **6**, p. 317.

[5] Bock F, 1958, *An algorithm for solving the "travelling salesman" and related network optimization problems*, Unpublished manuscript associated with a paper presented at the 14$^{\text{th}}$ ORSA National Meeting.

[6] Bodin LD & Berman O, 1979, *Routing and scheduling of school buses by computer*, Transportation Science, **13**, pp. 113–129.

[7] Bodin LD, Fagin G, Welebny R & Greenberg J, 1989, *The design of a computerized sanitation vehicle routing and scheduling system for the town of Oyster Bay, New York*, Computers and Operations Research, **16**, pp. 45–54.

[8] Bodin LD & Kursh SJ, 1979, *A detailed description of a computer system for the routing and scheduling of street sweepers*, Computers and Operations Research, **6**, pp. 181–198.

[9] Braca J, Bramel J, Posner B & Simchi-Levi D, 1993, *A computerized approach to the New York City school bus routing project*, Working Paper, Columbia University, New York (NY).

[10] Christofides N, Campos V, Corberán A & Mota E, 1986, *An algorithm for the rural postman problem on a directed graph*, Mathematical Programming Study, **26**, pp. 155–166.

[11] Corberán A & Sanchis JM, 1991, *A polyhedral approach to the rural postman problem*, European Journal of Operational Research, **79**, pp. 95–114.

[12] Croes GA, 1958, *A method for solving travelling salesmen problems*, Operations Research, **6**, pp. 791–812.

[13] Desrosiers J, Ferland JA, Rousseau J, Lapalme G & Chapleau L, 1986, *TRANSCOL: A multi–period school bus routing and scheduling system*, Management Science, **22**, pp. 47–71.

[14] Dijkstra EW, 1959, *A note on two problems in connection with graphs*, Numerische Mathematik, **1**, pp. 267–271.

[15] Dror M (Ed.), 2000, *Arc routing: Theory, solutions and applications*, Kluwer Academic Publishers, Boston (MA).

[16] Edmonds J & Johnson EL, 1973, *Matching, Euler tours and the Chinese postman problem*, Mathematical Programming, **5**, pp. 88–124.

[17] Eglese RW, 1994, *Routing winter gritting vehicles*, Discrete Applied Mathematics, **48**, pp. 231–244.

[18] Eglese RW & Murdock H, 1991, *Routing road sweepers in a rural area*, Journal of the Operational Research Society, **42(4)**, pp. 281–288.

[19] Eiselt HA, Gendreau M & Laporte G, 1995, *Arc routing problems, part I: The Chinese postman problem*, Operations Research, **43(2)**, pp. 231–242.

[20] Eiselt HA, Gendreau M & Laporte G, 1995, *Arc routing problems, part II: The rural postman problem*, Operations Research, **43(3)**, pp. 399–414.

[21] Fernándes de Córdoba P, Garcia Raffi LM & Sanchis JM, 1998, *A heuristic algorithm based on Monte Carlo methods for the rural postman problem*, Computers and Operations Research, **25(12)**, pp. 1097–1106.

[22] Flood MM, 1956, *The travelling salesman problem*, Operations Research, **4**, pp. 61–75.

[23] Frederickson GN, 1979, *Approximation algorithms for some routing problems*, Journal of the Association for Computing Machinery, **26**, pp. 538–554.

[24] Gelders LF & Cattrysse DG, 1991, *Public waste collection: A case study*, Belgian Journal of Operations Research, Statistical and Computing Science, **31**, pp. 3–15.

[25] Ghiani G & Improta G, 2001, *The laser–plotter beam routing problem*, Journal of the Operational Research Society, **52(8)**, pp. 945–951.

[26] Ghiani G & Laporte G, 2000, *A branch–and–cut algorithm for the undirected rural postman problem*, Mathematical Programming, **87**, pp. 467–481.

[27] Gondran M & Minoux M, 1984, *Graphs and algorithms*, John Wiley & Sons, Chichester.

[28] Grötschel M, Jünger M & Reinelt G, 1991, *Optimal control of plotting and drilling machines: A case study*, Operations Research, **35**, pp. 61–84.

[29] Guan M, 1962, *Graphic programming using odd and even cycles*, Chinese Mathematics, **1**, pp. 237–277.

[30] Hertz A, Laporte G & Nanchen-Hugo P, 1999, *Improvement procedures for the undirected rural postman problem*, INFORMS Journal on Computing, **11(1)**, pp. 53–62.

[31] Lacomme P, Prins C & Ramdane-Chérif W, 2002, *Fast algorithms for general arc routing problems*, Paper presented at the 16th Triennial Conference of the International Federation of Operations Research Societies, Edinburgh.

[32] Letchford AN, 1996, *Polyhedral results for some constrained arc routing problems*, PhD Dissertation, Lancaster University, Lancaster.

[33] Lenstra JK & Rinnooy Kan AHG, 1976, *On general routing problems*, Networks, **6**, pp. 273–280.

[34] Levy L & Bodin LD, 1988, *Scheduling the postal carriers for the United States postal service: An application of arc partioning and routing*, pp. 359–394 in Golden BL & Assad AA (Eds.), *Vehicle routing: Methods and studies*, North-Holland, Amsterdam.

[35] Lin S, 1965, *Computer solutions of the travelling salesman problem*, Bell Systems Technical Journal, **44**, pp. 2245–2269.

[36] Mehlhorn K & Näher S, 1999, *LEDA: A platform for combinatorial and geometric computing*, Cambridge University Press, Cambridge.

[37] Ramdane–Chérif W, 2002, *Problèmes de Tournées sur Arcs*, PhD Dissertation, University of Troyes, Troyes.

[38] Roy S & Rousseau J, 1989, *The capacitated Canadian postman problem*, Information Systems and Operational Research, **27**, pp. 58–73.

[39] Stern H & Dror M, 1979, *Routing electric meter readers*, Computers and Operations Research, **6**, pp. 209–223.

[40] Wunderlich J, Collette M, Levy L & Bodin LD, 1992, *Scheduling meter readers for Southern California Gas Company*, Interfaces, **22(3)**, pp. 22–30.