An Ant Colony Optimization Metaheuristic for the Undirected Rural Postman Problem

D. Laganà, G. Laporte, F. Mari, R. Musmanno, O. Pisacane
G–2007–106
December 2007
An Ant Colony Optimization Metaheuristic for the Undirected Rural Postman Problem

Demetrio Laganà

Dipartimento di Elettronica, Informatica e Sistemistica
Università della Calabria, 87036 Rende (CS), Italy
lagana@hpcc.unical.it

Gilbert Laporte

GERAD and Canada Research Chair in Distribution Management
HEC Montréal, Montréal (Québec) Canada, H3T 2A7
gilbert@crt.umontreal.ca

Francesco Mari

Supercomputing Center for Computational Engineering (CESIC), NEC Italy
Ponte Pietro Bucci 22B 87036 Rende (CS), Italy
francesco.mari@cesic.neceur.com

Roberto Musmanno
Ornella Pisacane

Dipartimento di Elettronica, Informatica e Sistemistica
Università della Calabria, 87036 Rende (CS), Italy
musmanno@unical.it, opisacane@deis.unical.it

December 2007

Les Cahiers du GERAD

G–2007–106

Copyright © 2007 GERAD
Abstract

This paper describes a new heuristic for the well-known Undirected Rural Postman Problem. It consists of two steps: it first constructs a partial solution using the Ant Colony Optimization metaheuristic, and the remaining required edges are then gradually inserted. Computational results on a set of benchmark instances are presented and comparisons with alternative heuristics are performed. The optimality gap is also computed by running a branch-and-cut algorithm.

Key Words: Ant Colony Optimization, Arc Routing, Undirected Rural Postman Problem, Heuristics.

Acknowledgments: This work was partly supported by the Ministero dell’Università e della Ricerca Scientifica (MURST), the Center of Excellence on High-Performance Computing and the University of Calabria, Italy, and by the Canadian natural Sciences and Engineering Research Council under grant 39682-05. The authors also benefited from the support of the Department of Electronics, Informatics and Systems (DEIS-University of Calabria-Italy). This support is gratefully acknowledged.
1 Introduction

The Undirected Rural Postman Problem (URPP) is defined on an undirected graph $G = (V, E)$, where $V$ is the vertex set, $E$ is the edge set, $c_{ij} \geq 0$ is the cost of traversing edge $(v_i, v_j) \in E$, $d_{ij}$ is the cost of a shortest chain between $v_i$ and $v_j$, and $E_R \subseteq E$ is a set of required edges. Let $V_R$ be the set of vertices incident to a required edge, and let $G_R = (V_R, E_R)$ the subgraph induced by $E_R$. The graph $G_R$ consists of $p$ disconnected subgraphs called connected components. Let $C_i$ be the $i^{th}$ connected component of the subgraph $G_R$, and let $V_i$ be its vertex set. The URPP consists of determining a minimum cost tour traversing each edge of $E_R$ at least once. An URPP feasible solution is represented by a closed sequence of vertices $S = (v_0, \ldots, v_n = v_0)$, covering all edges of $E_R$. A URPP partial solution is a sequence of vertices $\overline{S} = (v_0, \ldots, v_n = v_0)$, covering a subset $E_R'$ of required edges.

The URPP was introduced by Orloff [33] and was shown to be NP-hard by Lenstra and Rinnooy Kan [31]. Some URPP applications arise in the control of plotting and drilling machines [27], and in the optimization of laser-plotter beam movements [24]. Exact algorithms for the URPP have been proposed by Christofides et al. [5], Corberán and Sanchis [6], Letchford [32], Ghiani and Laporte [26] and Fernández et al. [21]. Constructive heuristics for the URPP have been presented by Frederickson [22], and by Pearn and Wu [34], while improvement procedures have been described by Hertz, Laporte and Nanchen-Hugo [29], and Groves and van Vuuren [28].

The aim of this paper is to describe an ant colony optimization (ACO) metaheuristic for the URPP. The remainder of the paper is organized as follows. In Section 2 we introduce the ACO metaheuristic for the URPP, called ANTURPP, and we describe the procedures that have been implemented. In Section 3 we introduce F-Race, a procedure employed to set the ANTURPP parameters. Computational results follow in Section 4.

2 ANTURPP: An ant colony heuristic for the URPP

ACO is a metaheuristic introduced by Dorigo [11] and successively extended by Dorigo in [12, 13, 14, 15], Cordón et al. [7], and Dorigo and Stützle in [16, 17]. Interesting applications of ACO were described by Gambardella, Taillard and Agazzi [23] and Doerner et al. [9] for the Vehicle Routing Problem. Recent applications were proposed by Doerner et al. [10] and Lacomme, Prins and Tanguy [30] for the Capacitated Arc Routing Problems. The ACO paradigm is inspired from the behaviour of real ants. Ants build a shortest path between a food source and their nest. Initially, they explore the area surrounding their nest in a random manner. When an ant finds a source of food, it carries some of it to the nest and deposits a pheromone trail on the ground. The quantity of deposited pheromone depends on the quantity and quality of the food found and will guide other ants to the food source. This indirect communication between the ants via the pheromone trails allows them to find a shortest path between their nest and food sources. In ACO algorithms, the pheromone
trails are simulated via a parameterized probabilistic model called the pheromone model, which consists of a set of parameters whose values are called the pheromone values, and used to probabilistically generate solutions. In general the ACO optimization strategy consists of iterating the following two steps:

1. solutions are built using a pheromone model, that is, a parameterized probability distribution over the solutions space;
2. the solutions built in earlier iterations are used to modify the pheromone values so as to bias the search toward high quality solutions.

In the following, we introduce the basic procedures of ANTURPP. An ant is a simple interacting software agent capable of building a partial solution for the URPP. An anthill \( k \) is a set of \( b_k \) ants, and \( B \) is the set of anthills. For all anthills \( k \in B \) and for all ants \( j = 1, \ldots, b_k \), \( M^j_1 \) is the set of indices of the connected components that remain to be served at the beginning of iteration \( t \). When a connected component \( C_i \) is traversed by ant \( j \) at iteration \( t \), \( i \) is removed from \( M^j_t \). The idea is that an ant decides at each iteration which connected component should be served and how it should be served. These decisions are made by the ant in a probabilistic manner, taking into account the pheromone values associated with each possible way of linking two connected components.

2.1 Algorithm structure

The aim of this section is to present the general structure of the ANTURPP algorithm. It starts constructing an auxiliary multigraph \( G^a = (N, A) \) from \( G = (V, E) \), using the procedure AUXILIARYGRAPH (Section 2.2.2). Let \( L \) be the chain linking together as many as possible required edges belonging to the connected component \( C_i \). Let \( L \) be the set of all \( L_i \). This set is determined by the procedure CHAIN (Section 2.2.1). To each chain \( L_i \) corresponds a vertex \( n_i \) of \( N \). The arc set \( A \) is defined as \( \{(n_i, n_h) : n_i, n_h \in N, i \neq h\} \) (Section 2.2.2).

Two anthills are located at each vertex of \( N \). An anthill is added at the end vertices of each chain \( L_i (i = 1, \ldots, p) \) corresponding to node \( n_i \) of the auxiliary multigraph \( G^a \). Therefore, the number of anthill is \( |B| = 2p \). Each ant starts from its anthill and moves on \( G^a \) to construct a partial URPP solution. The movement of the ants is made in a probabilistic manner and is driven by the pheromone trails updated by an ant only after coming back to its anthill. For each anthill, when all the ants have completed their job, the minimum cost partial URPP solution found by them is selected. The best partial URPP solution is obtained by taking the best partial solution found at each anthill and choosing the least cost one. The output of ANTURPP algorithm is an upper bound obtained adding to the best partial URPP solution the remaining required edges. An anthill is added at the end vertices of each chain \( L_i (i = 1, \ldots, p) \) corresponding to node \( n_i \) of the auxiliary multigraph \( G^a \). Therefore, the number of anthill is \( |B| = 2p \).

At iteration \( t \), ant \( j \) of the \( k \)th anthill \( (k = 1, \ldots, 2p) \) starts from node \( n_i \), and considers all feasible ways of connecting to another node \( n_h \). The ant gradually builds a partial URPP
solution by moving from its anthill to all nodes of $G^a$, selecting at each step the next chain to be served. The movement of ant $j$ is controlled by procedure MOVE (Section 2.2.4). At iteration $t$, when all nodes of $N$ have been traversed by ant $j$, the algorithm returns the partial solution $S^t_j$. At the end of the algorithm, the set $S(k) = \{S_1, \ldots, S_b\}$, for each $k \in B$, contains all the partial URPP solutions built by each ant of the $k$th anthill. Let $F(k) = \{F_1, \ldots, F_b\}$ be the set of related values of the URPP objective function, $F(k)_{\text{min}} = \min\{F_j : j = 1, \ldots, b\}$ is defined as the minimum value of the objective function found by the ants of the $k$th anthill. Let $\overline{S}(k)$ be the related best partial solution found by the ants of the $k$th anthill. Then $\overline{S}$ is the best partial solution such that $F = \min\{F(k)_{\text{min}} : k = 1, \ldots, 2p\}$. To create a feasible URPP solution, the partial solution $\overline{S}$ is increased by adding all required edges that do not belong to any chain $L_i (i = 1, \ldots, p)$. Let $E$ be the set of the required edges belonging to $\overline{S}$. Then the procedure ADD [29] returns the feasible solution whose cost $\overline{F}$ represents an upper bound for the URPP.

2.2 Description of the main procedures of ANTURPP

In this section, the procedures outlined in the previous section are described in more detail. A step-by-step description of the ANTURPP algorithm is presented in Figure 1.

1. Call CHAIN(C) (Section 2.2.1).
2. Construct the auxiliary graph $G^a$ (Section 2.2.2).
3. Set the ACO parameters (list of the parameters) (Section 3.1).
4. Initialize
   (a) all the values of $\Gamma$ to 1;
   (b) all the values of $P$ to the initial probabilities;
   (c) all the values of $\Delta \Gamma$ to 0;
   (d) the set $\overline{S} := \emptyset$ and $\overline{F} := \infty$.
5. Construct the best partial solution $\overline{S}$. For each anthill $k \in B$ and for each ant $j = 1, \ldots, b_k$, call
   (a) MOVE (Section 2.2.4);
   (b) UPDATEPROBABILITY (Section 2.2.3).
6. Determine the best upper bound $\overline{S}$ calling, for each edge $e = (v_u, v_w) \notin E$, ADD procedure [29].

Figure 1: Step-by-step description of the ANTURPP algorithm
2.2.1 Procedure \textsc{Chain}(C)

This procedure uses as input the set $C$ of connected components. For each connected component $C_i \in C$, the chain $L_i$ with the maximum number of required edges is built. The length of a chain $L_i$ is defined as the number of required edges it contains.

- **Step 1**: For all $v_u \in V_i$ ($i = 1, \ldots, p$), find the minimum spanning tree $\text{MST}_u^i$ starting from $v_u$. Select the minimum spanning tree $\text{MST}^i$ ($i = 1, \ldots, p$) with the smallest number of odd degree vertices.

- **Step 2**: For all $\text{MST}^i$ ($i = 1, \ldots, p$), find the set of vertices with the maximum degree, called cross-road vertices. For each cross-road vertex $v_r$ the two longest disjoint chains, starting from $v_r$, are built (Figure 2).

- **Step 3**: For all $\text{MST}^i$ ($i = 1, \ldots, p$) and for each cross-road vertex $v_r$, merge the two disjoint longest chains in $v_r$ and define a set $I_i$ ($i = 1, \ldots, p$) of chains for each connected component $C_i$ ($i = 1, \ldots, p$).

- **Step 4**: Select the chain $L_i \in I_i$ ($i = 1, \ldots, p$) with the maximum length (Figure 3).

2.2.2 Procedure \textsc{Auxiliarygraph}(G, L)

This procedure returns the auxiliary multigraph $G^a = (N, A)$, where $N = \{n_1, \ldots, n_p\}$ is the set of nodes associated with the $p$ longest chains of $L$, and $A$ is the set of edges associated with all possible ways of reconnecting every pair of chains in $L$.

Let $v_u^i$ and $v_w^i$ be the extremes vertices of the chain $L_i$ ($i = 1, \ldots, p$). For each pair of chains $L_i, L_h \in L$, there are four ways of connecting them by vertices $v_u^i, v_w^i \in L_i$ and $v_u^h, v_w^h \in L_h$ (Figure 4):

1. starting from $v_u^i$ and arriving at $v_u^h$ by a shortest chain of length $d_{uh}^i$ (Figure 5a);
2. starting from $v_u^i$ and arriving at $v_w^h$ by a shortest chain of length $d_{uw}^i$ (Figure 5b);
First longest chain $L_1$ starting from $v_4$

Second longest chain $L_2$ starting from $v_4$

Longest chain generated by merging $L_1$ and $L_2$

Figure 3: Longest chain $L$

3. starting from $v^i_u$ and arriving at $v^h_u$ by a shortest chain of length $d_{uw}^{ih}$ (Figure 5c);
4. starting from $v^i_u$ and arriving at $v^h_w$ by a shortest chain of length $d_{wu}^{ih}$ (Figure 5d);

Therefore, there are $2p(p - 1)$ connection possibilities.
Figure 4: The auxiliary multigraph $G^a$

(a) First reconnection possibility
(b) Second reconnection possibility
(c) Third reconnection possibility
(d) Fourth reconnection possibility

Figure 5: Four ways of reconnecting two chains

2.2.3 Procedure UPDATEPROBABILITY($G^a, \Gamma, \Delta \Gamma, P$)

The matrix $\Gamma$ contains the intensity of chains, the matrix $\Delta \Gamma$ is made up by the trails of pheromones deposited on the selected chains, and the matrix $P$ contains the probabilities associated with each edge of $G^a$. This procedure returns the updated matrix $\overline{P}$. 
Let \( m = \sum_{k \in B} b_k \) be the total number of ants. At the beginning of each iteration \( t \), an ant \( j \) leaves from each anthill \( k \) to build a partial solution \( S^t_j \). The ants coming from the two anthills corresponding to the same node \( n_i \in N \) visit in the opposite direction the shortest chain \( L_i \) of \( C_i \) \((i = 1, \ldots, p) \) represented by this node. At the end of iteration \( t \), the ants have constructed \( 2p \) partial URPP solutions \( S^t_j \) \((j = 1, \ldots, 2p) \). They then deposit pheromones on the edge of \( G^a \) chosen to build the partial solutions \( S^t_j \) \((j = 1, \ldots, 2p) \). On the other edges the pheromone evaporates, and their probability of being selected by the next ants thus decreases. A chain connecting two nodes \( n_i \) and \( n_h \) of \( G^a \) corresponds to an edge \( e = (n_i, n_h) \in A \). The updating of the pheromone value on edge \( e \) is determined by each ant between iterations \( t \) and \( t + 1 \). Let \( \tau^e(t+1) \) be the intensity of the chain generated at the beginning of iteration \( t+1 \):

\[
\tau^e(t+1) = \rho \tau^e(t) + \Delta \tau^e(t, t+1),
\]

where \( \rho \) is the evaporation coefficient, and the values \( \Delta \tau^e(t, t+1) \) represent the quantity of pheromones deposited on edge \( e \) by the ant at the end of iteration \( t \). The values \( \Delta \tau^e(t, t+1) \) are stored in the matrix \( \Delta\Gamma \). Let \( \tau^e(0) \) be the intensity of the chain associated with edge \( e \) at iteration \( t = 0 \), whose value can be arbitrarily set to 1 in our experiments. The values of \( \tau^e(t) \) are stored in a matrix \( \Gamma \). The ant \( j \), being on a node \( n_i \) at iteration \( t \), reads the indices of vertices to be visited from the memory list \( M^t_j \). If \( M^t_j \) is not empty, the probability of visiting the other nodes is distributed as follows. Let \( d^e \) be the length of the shortest chain associated to the edge \( e \) and let \( \eta^e = 1/d^e \) be the visibility of the chain. The transition probability from vertex \( n_i \in N \) to vertex \( n_h \in N \) is defined as:

\[
p^e(t) = \frac{[\tau^e(t)]^\alpha [\eta^e(t)]^\beta}{\sum_{w \in A : w=(n_i,n_v), v \in M^t_j} [\tau^w(t)]^\alpha [\eta^w(t)]^\beta}.
\]

Equation (2) is used if \( t \geq 1 \), when \( 2p \) ants have already computed their partial solution. For the ants that leave from the anthills located at node \( n_i \) at iteration \( t = 0 \), the distribution of the transition probability is computed using a formula that attributes a larger probability to serve the closest nodes than the others. It is called the greedy initial probability. Therefore, we can write:

\[
p^e(0) = \frac{\sum_{w \in A : w=(n_i,n_v), v \in M^0_j} d^w - d^e}{(\sum_{w \in A : w=(n_i,n_v), v \in M^0_j} d^w)(2|M^0_j| - 1)}.
\]

It is worth noting that

\[
\sum_{w \in A : w=(n_i,n_v), v \in M^0_j} p^w(0) = 1.
\]

The values of the probabilities are stored in the matrix \( P \).
### 2.2.4 Procedure MOVE($G^a, j, k, M^t_j, S^t_j, \Delta \Gamma$)

This procedure returns the next anthill at node $n_h \in N$, the new memory list $M^t_j$ of ant $j$, the new current partial URPP solution $S^t_j$ at iteration $t$ generated by ant $j$ starting from anthill $k$, and the updated matrix $\Delta \Gamma$.

We apply the ACO to connect all vertices of the auxiliary multigraph $G^a$. This way, a cycle that crosses all connected components of $G$ through the associated longest chains (Figure 6) is generated. This cycle represents a partial URPP solution because some required edges are not included in it. Procedure MOVE implements the single decision that ant $j$ has to make every time it is at node $n_i \in N$ and has to serve another node $n_h \in N$, with $h \in M^t_j$. We observe that the new memory list $M^t_j$ is obtained by eliminating the index corresponding to node $n_h$. This procedure is called iteratively as long as a partial URPP solution has not been built. Otherwise, $M^t_j$ is empty and the ant $j$ can go back to its anthill.

![Figure 6: Cycle connecting the longest chains correspondent to the connected components $C_i$, $C_j$ and $C_z$](image-url)
At the beginning the ant is located at its anthill. We have positioned two anthills at each node \( n_i \in N \), one for each way of serving the longest chain \( L_i \): from \( v^i_i \) to \( v^i_w \) and vice versa. The ant is initially forced to serve the required edge corresponding to its anthill in a given direction. For example, suppose that ant \( j \) is initially positioned at node \( n_i \in N \), corresponding to the longest chain \( L_i \), and suppose it is forced to serve \( L_i \) from \( v^i_i \) to \( v^i_w \). The ant has to make the following decisions: which longest chain \( L_h \) should be served next time, then which node of \( G^a \) should be reached next. If the ant decides to serve the longest chain \( L_h \) with the shortest chain from \( v^i_w \) to \( v^h_w \), it will have to move from \( v^h_w \) to \( v^h_w \) (Figure 7). Otherwise, if it serves \( L_h \) with the shortest chain from \( v^w_i \) to \( v^h_w \), it will have to move from \( v^w_i \) to \( v^h_w \) (Figure 8).

The sequence of operations performed at iteration \( t \) is described as follows:

1. Ant \( j \) constructs the feasible decisions set \( D_j^t \) (Figure 9).
2. Ant \( j \) computes the probabilities assigned to each feasible case using the UPDATE-PROBABILITY procedure (Section 2.2.3).
3. Ant \( j \) randomly decides to cross the shortest chain \( \{v^i_u, \ldots, v^h_w\} \), and updates the memory list \( M_j^t = M_j^{t-1} \setminus \{h\} \) and the current partial solution \( S^t_j = S_j^{t-1} \cup \{v^i_u, \ldots, v^h_w\} \). Let \( d_{ih} \) be the length of a shortest chain between \( v^i_u \) and \( v^h_w \) and let \( Q_1 \) be a constant quantity of pheromone deposited by ant \( j \) on the shortest chain chosen at iteration \( t \), to go from \( v^i_u \) to \( v^h_w \).

Let \( e = (n_i, n_h) \in A \) be the edge associated to the shortest chain \( \{v^i_u, \ldots, v^h_w\} \) on \( G^a \), then it is possible to define

\[
\Delta\tau^e(t, t + 1) = \begin{cases} 
Q_1/d_{ih} & \text{if the ant } j \text{ goes from } n_i \text{ to } n_h \text{ between } t \text{ and } t + 1 \\
0 & \text{otherwise.}
\end{cases}
\]  

According to (5), the pheromone intensity increases on the shortest chain when ant \( j \) goes from \( n_i \) to \( n_h \) and is inversely proportional to \( d_{ih} \). Let \( \Delta\Gamma \) be the updated matrix \( \Delta\Gamma \) such that the element corresponding to the edge \( e = (n_i, n_h) \) is

\[
\Delta\Gamma^e = \Delta\Gamma^e + \Delta\tau^e(t, t + 1).
\]  

3 ACO parameters

As described in Section 2.2.3, in order to initialize the ANTURPP algorithm, it is necessary to define parameters \( \rho, \alpha, \beta \) and \( b_k \), the number of ants associated to the \( k \)th anthill. In our computational study we have tested following feasible ranges for the previous parameters: \( b_k \in \overline{B} = \{10, 15, 20, 25, 30, 35, 40, 45, 50, 55\} \), \( \rho \in \overline{R} = \{0.5, 0.6, 0.7, 0.8\} \), \( \alpha \in \overline{A} = \{1, 1.5, 2, 2.5, 3\} \), and \( \beta \in \overline{P} = \{1, 1.5, 2, 2.5, 3\} \).
Figure 7: Serving the longest chain \( L_h \) with the shortest chain from \( v_{wi} \) to \( v_{uh} \)

Figure 8: Serving the longest chain \( L_h \) with the shortest chain from \( v_{wi} \) to \( v_{wh} \)

Figure 9: The set of feasible decisions \( D^t_j \) for the ant \( j \) at iteration \( t \)
3.1 F-Race for the ANTURPP algorithm

The F-Race procedure is used to find an optimal setting for the parameters of a meta-heuristic approach to a combinatorial optimization (CO) problem [2]. Let $\theta = \{\theta_1, \ldots, \theta_n\}$ be the set of all possible parameters configurations, $I = \{I_1, \ldots, I_m\}$ the set of the benchmark instances for CO, and $\chi(m, n)$ a matrix whose number of rows is equal to $m$ and the number of columns is equal to $n$. The heuristic algorithm $\Upsilon$, used to solve CO runs for a fixed time $T$ on each couple $(I_i, \theta_j)$, where $I_i \in I$ and $\theta_j \in \theta$. Each element $\chi_{I_i \theta_j}$ is the best solution found by $\Upsilon$ at time $T$ on instance $I_i$, using configuration $\theta_j$. The F-Race procedure is able to find the optimal setting for the parameters using a statistical test, starting from data matrix $\chi$.

In our specific case, the set of all possible configurations has cardinality $n = |\theta| = |B| \times |R| \times |A| \times |P| = 10 \times 4 \times 5 \times 5 = 1000$, and algorithm $\Upsilon$ is the ANTURPP.

4 Computational results

ANTURPP was implemented in C++ and was executed on a processor Itanium (1 GHz) NEC TX-7. It was tested on two sets of the URPP benchmark instances introduced by Ghiani and Laporte [26]: Type A graphs whose vertices are randomly generated in a plane. To ensure they are connected a specific test is used. In practice, $E_R$ is always disconnected in these graphs. Type C graphs whose vertex degrees are equal to 4 and $E_R$ is disconnected.

We have tested our algorithm on 29 instances of type A and 20 of type C.

For each instance, we compared the ANTURPP heuristic with the classical Frederickson algorithm [22] and with the recent constructive insertion heuristic [25]. Moreover, we compared our solutions with those generated by the application of the 2-OPT procedure [29] to the solutions provided by Frederickson and the insertion heuristics. Finally, we compared our solutions with the optimal ones provided by a branch-and-cut algorithm [25]. Computational results are presented in Tables 1 and 2.

The column headings are as follows:

- **OBJ**: objective function value. In the case of ANTURPP, it is the best value found over ten simulation runs;
- **SEC**: computational time (in seconds) corresponding to the best solution identified by ANTURPP;
- **Fred**: solution provided by the Frederickson heuristic [22];
- **Ins**: solution value given by the Ghiani, Laganá and Musmanno heuristic [25];
- **Fred2-OPT**: solution value provided by the Frederickson constructive algorithm [22], followed by the 2-OPT improvement procedure of Hertz, Laporte and Nanchen-Hugo [29];
• **Ins2-OPT**: solution value given by the constructive algorithm of Ghiani, Lagana and Musmanno [25] followed by the 2-OPT improvement procedure of Hertz, Laporte and Nanchen-Hugo [29];

• **ANTURPP**: the result provided by the ANTURPP algorithm;

• **BC**: optimal solution value provided by the branch-and-cut algorithm of Ghiani and Laporte [26];

• **R1**: best solution value provided by the ANTURPP heuristic divided the value provided by the **Frederickson + 2-OPT** heuristic;

• **R2**: best solution value provided by the ANTURPP heuristic divided the value provided by the **Ins+ 2-OPT** heuristic;

• **DEV1 (%)**: percentage gap between the best value provided by ANTURPP heuristic and the optimal value.

Our results (Tables 1, 2 and 3) show that we obtain the best performance on eight type A instances out of 29, on 14 type C instances out of 20 and, finally, on 13 Christofides instances out of 23. Regarding the type A instances, we can see that the deviation among the ANTURPP upper bound and the optimal objective value does not exceed 1.67% for

Table 1: Computational results for type A instances

<table>
<thead>
<tr>
<th>Instance Name</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
type A instances, and 5.17% for type C instances. For both instance types, our algorithm requires a very low CPU time to identify the final solution compared with other ones. On the Christofides instances, we can see that, except for the instance Chr3, the maximum
deviation among the ANTURPP solution and the optimal value is about 3.95%. The instance Chr3 is the only one in which the optimality gap is high. After examining the results related to this instance, we can deduce that the high optimality gap is due to a stagnation of the search. The reason could be related to the structure of this instance and to the use of an ACO algorithm which, contrary to a MAx Min Ant System (MMAS, [35]), does not consider a limit on the amount of pheromone. In other words, this high gap could be related to the fact that the best solution, found by the ants, with a high optimality gap, has a high pheromone concentration. This concentration could tend to increase itself and could force the ants to build the same solution. It is important to observe that this problem occurs only on this instance, because, with the other, the ANTURPP is able to find good solutions.

Therefore the ANTURPP algorithm obtains, within a reasonable computational time, a tight upper bound, in particular with the instances defined on a large number of required edges. Moreover, we have shown that the proposed method outperforms all existing URPP algorithms in terms of computational times.

References


