Branch-and-bound algorithm for an arc routing problem

Fábio Luiz Usberti
Faculdade de Engenharia Elétrica e de Computação - UNICAMP
Cidade Universitária Zeferino Vaz, Av. Albert Einstein, 400. CEP: 13083-852, Campinas - SP
e-mail: fusberti@gmail.com

Paulo Morelato França
Departamento de Matemática, Estatística e de Computação - FCT/UNESP
R. Roberto Simonsen, 305. CEP: 19060-900, Presidente Prudente - SP
e-mail: paulo.morelato@fct.unesp.br

André Luiz Morelato França
Faculdade de Engenharia Elétrica e de Computação - UNICAMP
Cidade Universitária Zeferino Vaz, Av. Albert Einstein, 400. CEP: 13083-852, Campinas - SP
e-mail: morelato@dsee.fee.unicamp.br

Abstract

The Open Capacitated Arc Routing Problem (OCARP) is an NP-hard combinatorial optimization problem where, given an undirected graph, the objective is to find a minimum cost set of tours that services a subset of edges with positive demand under capacity constraints. This problem is related to the Capacitated Arc Routing Problem (CARP) but differs from it since OCARP does not consider a depot, and tours are not constrained to form cycles. Three lower bounds are proposed to the OCARP, one of them uses a subgradient method to solve a Lagrangian relaxation. These lower bounds are integrated within a branch-and-bound framework to conceive the first OCARP exact algorithm. The branch-and-bound algorithm is started with high-quality upper bounds obtained with a sucessful GRASP with evolutionary path-relinking, originally developed to solve the CARP. Computational tests compared the proposed branch-and-bound with a commercial state-of-the-art ILP solver. The results show that the branch-and-bound outperformed CPLEX in both overall average deviation from lower bounds and number of best lower bounds.

KEYWORDS: arc routing, Lagrangian relaxation, branch-and-bound, CPLEX.

1 Introduction

The objective of arc routing problems consists in determining the least cost traversal of a given subset of edges in a graph, with one or more collateral constraints. Many real world applications are modeled as arc routing problems, such as street sweeping, garbage collection, mail delivery, school bus routing, meter reading etc, and estimates on the expenditure involved in these services reach billions of dollars annually in the United States, thus revealing a substantial savings potential (Eiselt et al. 1995, Dror 2001, Hertz 2005, Corberán and Prins 2010).

The Capacitated Arc Routing Problem (CARP) (Golden and Wong 1981) is a combinatorial optimization problem defined in a connected undirected graph $G(V,E)$ with non-negative costs and demands on its edges. There is a fleet of identical vehicles with limited capacity that must service all edges with positive demand (required edges). The objective is to search for a set of minimum cost tours that start and finish in a distinguished node, called depot.

A CARP generalization consists in allowing both closed and open tours. An open tour can use different nodes to start (*source*) and end (*sink*) the tour, while a closed tour is a cycle in which the source and sink nodes are necessarily the same. This problem in which tours are not constrained to form cycles is called the Open Capacitated Arc Routing Problem (OCARP) (Usberti et al. 2011b). Many combinatorial optimization problems can be represented as an OCARP instance. For example, two practical applications for the OCARP are the Meter Reader Routing Problem (Stern and Dror 1979, Bodin and Levy 1989, Bodin and Levy 1991, Wunderlich et al. 1992) and the Cutting Path Determination Problem (Moreira et al. 2007, Rodrigues and Ferreira 2012).

The OCARP has been proven NP-hard (Usberti et al. 2011b). This is a negative result since it means that the complexity of an exact algorithm will not be polynomial, unless $P = NP$. Nonetheless, the extent by which OCARP instances can be optimally solved is still a worthy investigation.

This work proposes an algorithm that finds an optimal solution to any feasible OCARP instance. The complexity of this exact algorithm is exponential, however it performs better than solving the OCARP model using CPLEX, a state-of-the-art integer linear programming (ILP) solver.

2 ILP Formulation

The following OCARP integer linear programming model (Usberti et al. 2011b) is based on a CARP model (Golden and Wong 1981) adaptation to also consider open tours. This model is applied to a directed graph, hence each edge $(i,j) \in E$ from the undirected OCARP graph $G(V,E)$ is treated as two arcs (i,j) and (j,i). It is assumed that the number of vehicles M is a fixed parameter and $N(i)$ denotes the nodes adjacent to node i in G.

Two sets of decision variables are defined: $x_{ij}^k = 1$ if tour k traverses arc (i,j), $x_{ij}^k = 0$ otherwise; $l_{ij}^k = 1$ if tour k services arc (i,j), $l_{ij}^k = 0$ otherwise. There are the auxiliary variables u_S^k, v_S^k, w_S^k ($S \subseteq V, \tilde{S} = V \setminus S$) and α_i^k ($i \in V, k \in \{1,\ldots,M\}$): if $u_S^k = 0$ then tour k does not have a sufficient number of arcs to form an illegal subcycle in S; if $v_S^k = 0$ then tour k traverses the cut-set $(S,\tilde{S})$; if $w_S^k = 0$ then tour k contains a source node in S; if $\alpha_i^k = 0$ then i is not the source node of tour k.

The objective function (1) minimizes the total solution cost. Constraints (2,3) maintain routes continuity (every node, except for the source and sink, must have equal indegree and outdegree). Constraints (4) state that serviced arcs must also be traversed; constraints (5) force each required edge (represented by two arcs) to be serviced in a unique direction by a single vehicle; constraints (6) are related to the vehicle capacity; and constraints (7) eliminate illegal subcycles. OCARP has the two following properties, proven in (Usberti et al. 2011b):

Property 1 *Given any OCARP instance with M vehicles and at least M required edges, there exists an optimal solution which uses all vehicles.*

Property 2 *Given any OCARP instance, there exists an optimal solution in which all tours are formed by alternating required edges with shortest paths linking one required edge to another.*

(OCARP)

$$MIN \quad \sum_{k=1}^{M} \sum_{(i,j)\in E} c_{ij} x_{ij}^{k} \tag{1}$$

s.t.

$$\sum_{j\in N(i)} (x_{ij}^{k} - x_{ji}^{k}) \leqslant \alpha_i^k \qquad (i \in V, k \in \{1,\ldots,M\}) \tag{2}$$

$$\sum_{i\in V} \alpha_i^k \leqslant 1 \qquad (k \in \{1,\ldots,M\}) \tag{3}$$

$$x_{ij}^k \geqslant l_{ij}^k \qquad ((i,j) \in E_R, k \in \{1,\ldots,M\}) \tag{4}$$

$$\sum_{k=1}^{M} (l_{ij}^k + l_{ji}^k) = 1 \qquad ((i,j) \in E_R) \tag{5}$$

$$\sum_{(i,j)\in E_R} d_{ij} l_{ij}^k \leqslant D \qquad (k \in \{1,\ldots,M\}) \tag{6}$$

$$\left.\begin{aligned} &\sum_{(i,j)\in(S,S)} x_{ij}^k - |S|^2 u_S^k \leqslant |S| - 1 \\ &\sum_{(i,j)\in(S,\tilde{S})} x_{ij}^k + v_S^k \geqslant 1 \\ &\sum_{i\in S} \alpha_i^k + w_S^k \geqslant 1 \\ &u_S^k + v_S^k + w_S^k \leqslant 2 \end{aligned}\right\} \qquad (S \subseteq V, \tilde{S} = V \setminus S, k \in \{1,\ldots,M\}) \tag{7}$$

$$x_{ij}^k \in \{0,1\} \qquad ((i,j) \in E, k \in \{1,\ldots,M\}) \tag{8}$$

$$l_{ij}^k \in \{0,1\} \qquad ((i,j) \in E_R, k \in \{1,\ldots,M\}) \tag{9}$$

$$\alpha_i^k \in \{0,1\} \qquad (i \in V, k \in \{1,\ldots,M\}) \tag{10}$$

$$u_S^k, v_S^k, w_S^k \in \{0,1\} \qquad (k \in \{1,\ldots,M\}, S \subseteq V) \tag{11}$$

3 OCARP augmented graph

The exact algorithm proposed to solve OCARP is motivated by a transformation performed on the original OCARP graph $G(V,E)$. This graph can be augmented into an equivalent full graph $\tilde{G}(\tilde{V},\tilde{E})$ by replacing all non-required edges of G with minimum shortest paths between required edges. Figure 1 shows an example of this transformation with an OCARP instance containing 24 edges, six of them required. In the example, the augmented graph has 66 edges, six required (unchanged from the original graph), and 60 non-required edges connecting every pair of required edges. The costs of required edges remain the same in the augmented graph. As for the non-required edges, their costs are shortest path costs to traverse the original graph from one node to another, both nodes belonging to required edges. In case of two adjacent required edges, there will be a zero cost non-required edge connecting them in the augmented graph.

The augmented graph $\tilde{G}$ is frequently used in literature to solve the CARP (Lacomme et al. 2004, Beullens et al. 2003) since two required edges are always connected by a shortest path in an optimal solution (Property 2).

An optimal OCARP solution in G, when migrated to the augmented graph $\tilde{G}$, will represent a spanning forest $\tilde{F}(\tilde{V},\tilde{E}_S)$. The set of edges $\tilde{E}_S$ represents OCARP tours and it is formed by the union of required edges ($\tilde{E}_R$) and a subset of non-required edges ($\tilde{E}_{NR}$) from the augmented graph ($\tilde{E}_S = \tilde{E}_R \cup \tilde{E}_{NR}$).

A spanning forest $\tilde{F}$ representing an optimal OCARP solution has the following properties:

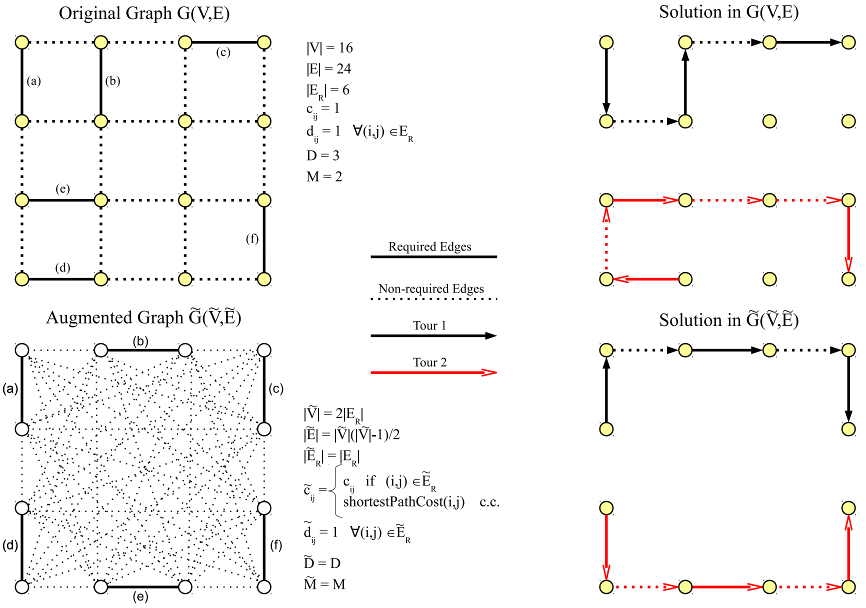
Original Graph G(V,E)
(a)
(b)
(c)
(e)
(f)
(d)
|V| = 16
|E| = 24
$|E_R| = 6$
$c_{ij} = 1$
$d_{ij} = 1 \quad \forall(i,j) \in E_R$
D = 3
M = 2
Solution in G(V,E)
Required Edges
Non-required Edges
Tour 1
Tour 2
Augmented Graph $\tilde{G}(\tilde{V},\tilde{E})$
(b)
(a)
(c)
(d)
(f)
(e)
$|\tilde{V}| = 2|E_R|$
$|\tilde{E}| = |\tilde{V}|(|\tilde{V}|-1)/2$
$|\tilde{E}_R| = |E_R|$
$\tilde{c}_{ij} = \begin{cases} c_{ij} & \text{if } (i,j) \in \tilde{E}_R \\ \text{shortestPathCost(i,j)} & \text{c.c.} \end{cases}$
$\tilde{d}_{ij} = 1 \quad \forall(i,j) \in \tilde{E}_R$
$\tilde{D} = D$
$\tilde{M} = M$
Solution in $\tilde{G}(\tilde{V},\tilde{E})$

Figure 1: Transforming OCARP graph $G(V,E)$ into the augmented graph $\tilde{G}(\tilde{V},\tilde{E})$.

- $\tilde{F} = \{\tilde{T}_1, \tilde{T}_2, \ldots, \tilde{T}_M\}$ – one tree per OCARP vehicle.
- $|\tilde{E}_S| = |\tilde{V}| - M \Rightarrow |\tilde{E}_{NR}| = |\tilde{E}_R| - M$ – number of non-required edges.
- $1 \leqslant \delta(v) \leqslant 2 \qquad (v \in \tilde{V}, \delta(v) = \text{degree of } v)$ – node degree constraints.
- $\sum_{(i,j) \in \tilde{T}_k} d_{ij} \leqslant D \qquad (k \in \{1, 2, \ldots, M\})$ – vehicle capacity constraints.

One advantage in solving the OCARP using the augmented graph $\tilde{G}$ is that the problem becomes equivalent to that of finding a minimum cost spanning forest with M trees, constrained by tree capacity and node degree. In other words, the combinatorics of the problem resides in finding the best $|R| - M$ non-required edges of minimum cost, attending these constraints. In the following, this problem is addressed as the *capacity and degree constrained minimum spanning forest problem* (CDCMSFP).

A second advantage in solving the OCARP as a minimum spanning forest is that there are easily computable lower bounds. These lower bounds will be useful within the branch-and-bound framework to solve OCARP.

4 Branch-and-Bound for the CDCMSFP

The branch-and-bound implementation decisions regarding the branching scheme, node selection and lower bound calculations are described.

First of all, a solution of the CDCMSFP, partial or complete, is represented by the set of required and non-required edges $\tilde{E}_S$ of $\tilde{G}$ contained in the solution. Also, consider that all non-required edges $\tilde{E} \setminus \tilde{E}_R$ are sorted in a list $L = \{e_1, e_2, \ldots, e_n\}$ by increasing order of cost.

With respect to the branching scheme, Figure 2 assists the comprehension of how the solution space is partitioned. In each level of the search tree, a non-required edge is included into the partial solution from the level above. Thus each search tree node containing a subset of non-required edges is branched to include one additional non-required edge. The branching scheme takes the highest index i from the most costly edge e_i in the current search tree node, as long as $i < n$, and divides the solution space into $n-i$ mutually exclusive partitions. In each partition, an edge e_j $(\forall j \in [i+1,n])$ is included in the corresponding partial solution.

The node selection scheme adopted in the proposed branch-and-bound algorithm is the least-lower-bound. One advantage of this scheme is that the current node lower bound is also a global lower bound, since it is, by definition, the smallest lower bound among all active nodes. In addition, if the current node represents a complete feasible solution, then it must be optimal. Another advantage relies on the fact that this scheme emphasizes the improvement of the global lower bound. This may be viewed as negative as well, because complete solutions are discovered less often than using, for example, the depth-first search.

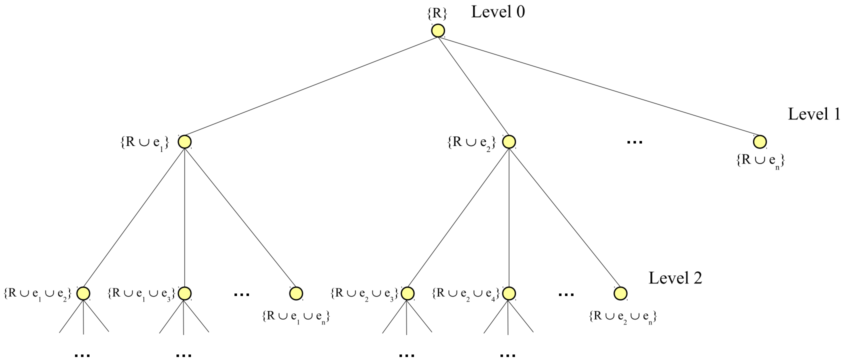

Figure 2: Branch-and-bound search tree for OCARP.

As for the lower bound calculation, the proposed algorithm uses three procedures detailed in the following section.

5 Lower Bounds

In the developed branch-and-bound algorithm, three lower bounds were tested, with different trade-offs of tightness and computational effort, and this section shows how these lower bounds were combined with the algorithm. Assume that $\tilde{E}_{PS}$ is the subset of edges representing a given partial CDCMSFP solution, $L = \{e_1, e_2, \dots, e_n\}$ is a sorted list of all non-required edges $\tilde{E} \setminus \tilde{E}_R$ in increasing order of cost, and I_L is the list of the indices for each edge in L $(I_L[e_i] = i)$. The following describes three proposed lower bounds.

5.1 LB_1 – minimum cost edges

Lower bound LB_1 is computed by relaxing the illegal subcycles, degree and capacity constraints of the CDCMSFP. Supposing $e_{\max}$ is the maximum cost edge in $\tilde{E}_{PS}$, then LB_1 can be calculated by adding to the cost of the partial solution, the cost of the cheapest n_{me} edges in L starting from index $I_L[e_{\max}]+1$, where $n_{me} = |\tilde{E}_R| - M - |\tilde{E}_{PS}|$ is the number of missing edges to complete the solution.

Lower bound LB_1 can be computed in $O(1)$ by simply fetching the missing edges cost from a matrix $C_{m \times n}$ $(m = |\tilde{E}|$, $n = |\tilde{E}_R| - M)$. The pre-processing phase of constructing matrix C requires $O(|\tilde{E}||\tilde{E}_R|)$ of computational effort, however the pre-processing is performed only once, while LB_1 is calculated numerous times during execution.

The row index of matrix C represents the edge index in list L from which to start the sum of the missing edges costs, and this index should be the highest cost edge index from the partial solution plus one $(I_L[e_{\max}]+1)$. The column index of matrix C represents the number of missing edges (n_{me}). Eq. (12) shows how to fill matrix C for any OCARP instance.

$$C[i,j] = \begin{cases} \displaystyle\sum_{k=i}^{i+j-1} cost(L[k]) & \text{if } i+j-1 \leqslant |\tilde{E}| \\ \infty & \text{otherwise} \end{cases} \tag{12}$$

Algorithm 1 shows how to determine LB_1 for a given partial solution $\tilde{E}_{PS}$.

Algorithm 1 LB1 – minimum cost edges lower bound

```
Input: $\tilde{E}_{PS}$ – set of non-required edges from the partial solution
Output: $LB_1$ – minimum cost edges lower bound
$e_{\max} \leftarrow$ getMaxCostEdge($\tilde{E}_{PS}$) – non-required edge with highest cost in $\tilde{E}_{PS}$
$n_{me} \leftarrow |\tilde{V}| - M - |\tilde{E}_{PS}|$ – number of missing edges in $\tilde{E}_{PS}$
$i \leftarrow I_L[e^{\max}] + 1$ – row of matrix $C$
$j \leftarrow n_{me}$ – column of matrix $C$
$LB_1 \leftarrow cost(\tilde{E}_{ps}) + C[i,j]$
return $LB_1$
```

5.2 LB_2 – minimum cost spanning forest

Lower bound LB_2 is computed by relaxing the degree and capacity constraints of the CDCMSFP. In other words, lower bound LB_2 computes the minimum cost spanning forest that completes the partial solution, and this is done by adding the cheapest $n_{me} = |E_R| - M - |\tilde{E}_{ps}|$ edges which do not form a cycle in the solution (Algorithm 2). This is done within almost-linear time $O(|\tilde{E}|\alpha(\tilde{V}))$[1] with Kruskal algorithm implemented using disjoint-set-forest and union-by-rank (Cormen et al. 2001). The algorithm halts once the solution complies $|E_R| - M$ non-required edges (or equivalently M trees).

Algorithm 2 LB2 – minimum cost spanning forest lower bound

```
Input: $\tilde{E}_{PS}$ – set of non-required edges from the partial solution
Output: $S$ – spanning forest with $M$ trees
$e_{\max} \leftarrow$ getMaxCostEdge($\tilde{E}_{PS}$) – non-required edge with highest cost in $\tilde{E}_{PS}$
$n_{me} \leftarrow |\tilde{V}| - M - |\tilde{E}_{PS}|$ – number of missing edges in $\tilde{E}_{PS}$
$S \leftarrow \tilde{E}_{PS}$
$i \leftarrow I_L[e^{\max}] + 1$
while ($n_{me} > 0$) do
    $e \leftarrow L[i]$
    if (containsCycle($S \cup \{e\}$) = false) then
        $S \leftarrow S \cup \{e\}$
        $n_{me} \leftarrow n_{me} - 1$
    end if
    $i \leftarrow i + 1$
end while
return $S$
```

5.3 LB_3 – degree constrained minimum cost spanning forest

Lower bound LB_3 is computed by relaxing only the capacity constraints of the CDCMSFP. This lower bound is based on a heuristic for the *degree constrained minimum spanning tree problem* (DCMSTP) (Andrade et al. 2006), which uses Lagrangian dual information to derive the lower

[1] $\alpha(.)$ denotes the inverse Ackermann function, which, for all practical purposes, regards the inequality $\alpha(.) \leqslant 5$.

bounds. Let $E(S) = \{(i,j) \in E | i \in S, j \in S\}$ be the set of edges contained in the node set S, and $N(i) = \{j \in V | (i,j) \in E\}$ the set of nodes adjacent to node i. An ILP model for the DCMSTP is:

$$\begin{aligned}
&\text{(DCMSTP)} \\
&MIN \quad Z = \sum_{(i,j)\in E} c_{ij}x_{ij} && && (13)\\
&\text{st} \\
&\sum_{(i,j)\in E} x_{ij} = |V| - 1 && && (14)\\
&\sum_{(i,j)\in E(S)} x_{ij} \leqslant |S| - 1 && \forall S \subset V && (15)\\
&\sum_{j\in N(i)} x_{ij} \leqslant d_i && \forall i \in V && (16)\\
&x_{ij} \in \{0,1\} && \forall (i,j) \in E && (17)
\end{aligned}$$

The objective function (13) minimizes the total solution cost; constraint (14) forces $|V| - 1$ edges in the solution; constraints (15) are for illegal subcycle elimination; constraints (16) limit the degrees of the nodes. If the problem is to generate a forest (more than one tree), then the right-hand side of constraint (14) should be replaced by $|V| - t$, where t is the number of desired trees ($t = M$ in this work). The Lagrangian relaxation used by (Andrade et al. 2006) and equally in this work, is obtained by dualizing contraints (16), which turns into the following relaxed λ-DCMSTP formulation.

$$\begin{aligned}
&(\lambda\text{-DCMSTP}) \\
&MIN \quad Z_\lambda = \sum_{(i,j)\in E} c_{ij}x_{ij} - \sum_{i\in V} \lambda_i \left(d_i - \sum_{j\in N(i)} x_{ij} \right) && && (18)\\
&\text{st} \\
&\sum_{(i,j)\in E} x_{ij} = |V| - 1 \\
&\sum_{(i,j)\in E(S)} x_{ij} \leqslant |S| - 1 && \forall S \subset V \\
&x_{ij} \in \{0,1\} && \forall (i,j) \in E
\end{aligned}$$

Eq. (19) is obtained by algebrically manipulating the objective function (Eq. (18)).

$$\begin{aligned}
Z_\lambda &= \sum_{(i,j)\in E} c_{ij}x_{ij} - \sum_{i\in V} \lambda_i \left(d_i - \sum_{j\in N(i)} x_{ij} \right) \\
&= \sum_{(i,j)\in E} c_{ij}x_{ij} + \sum_{i\in V} \lambda_i \left(\sum_{j\in N(i)} x_{ij} \right) - \sum_{i\in V} \lambda_i d_i \\
&= \sum_{(i,j)\in E} c_{ij}x_{ij} + \sum_{(i,j)\in E} (\lambda_i + \lambda_j)x_{ij} - \sum_{i\in V} \lambda_i d_i \\
&= \sum_{(i,j)\in E} (c_{ij} + \lambda_i + \lambda_j)x_{ij} - \sum_{i\in V} \lambda_i d_i \qquad (19)
\end{aligned}$$

The relaxed problem λ-DCMSTP is to find the minimum spanning tree with Lagrangian costs $l_{ij} = c_{ij} + \lambda_i + \lambda_j$. From the Lagrangian duality theory, it is straightforward to establish that the optimal cost Z^*_λ is a lower bound for the optimal cost Z^* for any given set of multipliers λ. There is also the Lagrangian dual problem, which is to find the set of multipliers λ^* which maximizes Z^*_λ, in other words, finds the maximum lower bound for Z^*. To solve the Lagrangian dual, the subgradient method (Beasley 1993) is employed to generate a sequence of multipliers which converges to λ^*.

Algorithm 3 computes lower bound LB_3 by solving several times a minimum cost spanning forest with Lagrangian costs on the edges using Algorithm 2. Both parameters θ and τ, regarding the subgradient step size, have been set with values suggested in literature (Beasley 1993) (lines 1 and 2). The Lagrangian multipliers are initially set with zeros (line 5) so that in the first iteration the Lagrangian costs are equal to the original costs. In line 9, the costs of the edges are updated by the Lagrangian costs ($l_{ij} = c_{ij} + \lambda_i + \lambda_j$). The nodes degree infeasibilities of the spanning forest S_λ generated in line 10 is measured by the subgradients (line 16), which are used to determine the step size (line 19) for the Lagrangian multipliers adjustment (line 24).

Algorithm 3 LB3 – degree constrained minimum cost spanning forest lower bound

```
Input: $\tilde{E}_{PS}$ – set of non-required edges from the partial solution, $UB$ – an upper bound for the $\lambda$-DCMSTP, obtained
    heuristically, $maxIter$ – maximum number of iterations, used as stopping criterion
Output: $LB_3$ – the best lower bound obtained by solving the dual Lagrangian problem
 1: $\theta \leftarrow 2.0$ – adjusts the step size depending on how tight is the $GAP$ between $LB$ and $UB$
 2: $\tau \leftarrow 1.05$ – prevents the step size becoming too small when the same happens with the $GAP$
 3: $iter \leftarrow 0$
 4: for ($\forall i \in V$) do
 5:     $\lambda_i \leftarrow 0$
 6: end for
 7: $LB_3 \leftarrow -\infty$
 8: while (($iter \leqslant maxIter$) and ($UB > LB_3$)) do
 9:     updateEdgeCosts($\tilde{E}_{PS}, \lambda$) – apply Lagrangian costs to the edges
10:     $S_\lambda \leftarrow$ LB2($\tilde{E}_{PS}$) – solve $\lambda$-DCMSTP with Kruskal
11:     $solCost \leftarrow cost(S_\lambda) - \sum_{i \in V} \lambda_i d_i$ – total cost of the $\lambda$-DCMSTP solution, Eq. (19)
12:     if ($solCost > LB_3$) then
13:         $LB_3 \leftarrow solCost$
14:     end if
15:     for ($\forall i \in V$) do
16:         $G_i \leftarrow$ nodeDegree($S_\lambda, i$) $- d_i$
17:     end for
18:     if ($\sum_{i \in V} G_i > 0$) then
19:         $t \leftarrow \dfrac{\theta(\tau UB - LB_3)}{\sum_{i \in V} G_i^2}$
20:     else
21:         return $LB_3$ – optimal $\lambda$-DCMSTP solution cost
22:     end if
23:     for ($\forall i \in V$) do
24:         $\lambda_i \leftarrow \max(0, \lambda_i + tG_i)$
25:     end for
26:     $iter \leftarrow iter + 1$
27: end while
28: return $LB_3$
```

5.4 Lower Bounds Tightness

A trivial lower bound LB_0 for an optimal OCARP solution cost may be computed by summing all required edges costs ($LB_0 = \sum_{e \in E_R} c_e$). This however can be very loose when optimal solutions require deadheading. The following paragraphs will show that $LB_3 \geqslant LB_2 \geqslant LB_1 \geqslant LB_0$.

1. $LB_1 \geqslant LB_0$: Lower bound LB_1 considers the cost of the partial solution ($\tilde{E}_{PS}$), which in turn contains all required edges ($E_R \subseteq \tilde{E}_{PS}$).

2. $LB_2 \geqslant LB_1$: Both lower bounds LB_1 and LB_2 are formed by adding the same number of edges (n_{me}) to the partial solution. However, LB_1 always adds the least cost edges.

3. $LB_3 \geqslant LB_2$: The Lagrangian multipliers in Algorithm 3 are initially set with zeros, meaning that the first minimum cost spanning forest obtained in line 10 will have cost equal to LB_2.

5.5 Lower Bounds Composition

Empirical studies have revealed that the branch-and-bound algorithm performs better when the lower bounds are combined, rather than used individually. This is because there are moments during the search tree exploration when a fast computable lower bound is preferred instead of a costly one, despite this last being a tighter bound.

As explained in Section 4, for every child node generated in the branching phase a non-required edge is added to the corresponding partial solution. In the proposed branch-and-bound, these child nodes are formed in an orderly manner of decreasing costs, starting with a child node that receives the highest cost non-required edge. This denotes that the first child nodes should likely contain a relatively high cost partial solutions. Thus lower bounds for these solutions do not need to be tight, and fast lower bounds would save substantial amounts of execution time.

Let LB and UB be the incumbent best lower and upper bounds at a given branch-and-bound iteration. At the start of a branching phase, the lower bounding procedure of choice is LB_1, the fastest one. If a child node results in a lower bound LB_1 such that $LB_1 < LB + (UB - LB)/10$, then the lower bounding procedure is replaced by LB_2. This test is repeated for every generated child node, and if LB_2 also fails to pass the test, it is replaced by LB_3, which assumes the job of deriving lower bounds until the last child node is created in the course of that branching phase. The choice of using the relative value $(UB - LB)/10$ was made after several empirical tests.

6 Complexity Study

The computational complexity of proposed exact algorithm can be bounded by the number of nodes in the tree. This can be quantified with support of Figure 2. Each i-th level of this tree contains all possible solutions (not necessarily feasible) with i non-required edges. Therefore, the total amount of nodes in the search tree (N_{tree}) corresponds to the sum of nodes in all its $|E_R| - M$ levels (Eq. (20)).

$$N_{tree} = \sum_{i=0}^{|E_R|-M} \binom{|\tilde{E}|}{i} < \sum_{i=0}^{|\tilde{E}|} \binom{|\tilde{E}|}{i} = 2^{|\tilde{E}|} \tag{20}$$

Each node in the search tree goes through a heap insertion, a heap deletion, and possibly the calculation of all three lower bounds LB_1, LB_2, and LB_3. Thus the computational effort performed on each node is bounded by $O(|\tilde{E}|\alpha(\tilde{V}))$. These calculations are performed N_{tree} times, making the computational complexity of the proposed method to be bounded by $O(|\tilde{E}|\alpha(\tilde{V})2^{|\tilde{E}|})$.

7 Computational Experiments

The computational tests were performed on a standard set of CARP instances[2], which includes 23 *gdb* (7-27 nodes, 11-55 edges) (Golden et al. 1983), 34 *val* (24-50 nodes, 34-97 edges) (Benavent et al. 1992), 24 *egl* (77-140 nodes, 98-190 edges) (Li and Eglese 1996), 32 *A*, and 24 *B* instances

[2] http://www.uv.es/~belengue/carp.html; http://www.hha.dk/~sanw

(10-40 nodes, 15-69 edges) (Wøhlk 2008), totaling 137 instances. The depot was considered a common node while the rest of the data left intact, leading to five groups of instances referred as *ogdb*, *oval*, *oegl*, *oA*, and *oB*.

Since OCARP is only meaningful with a fixed M (Property 1), the computational tests have considered three classes of parameterization: $M = M^*$, $M = M^* + 1$ and $M = M^* + 2$, where M^* represents the minimum number of vehicles necessary for a feasible solution. Consequently, from each of the 137 CARP instances, three different numbers of vehicles are considered, thus deriving 411 OCARP instances (Usberti et al. 2011b).

Table 1 lists the branch-and-bound parameters and their values used in the computational experiments. The same time limit and startup upper bounds used by the branch-and-bound were also adopted to CPLEX. The branch-and-bound algorithm was implemented in C programming language, and all tests were executed in a Intel Core 2 Quad 3.0 GHz with 4 Gb of RAM.

The computational experiments use as the starting upper bound (UB) the best solution obtained by a sucessful GRASP with PR metaheuristic (Usberti et al. 2011a).

Table 1: Branch-and-bound parameters.

$timeLimit = 3600$	execution time limit in seconds.
$heapSize = 2 \times 10^7$	heap size limit in number of nodes.
$\lambda_0 = [0, 0, \ldots, 0]$	initial Lagrangian multipliers for the subgradient method.
$\theta = 2.0$	adjusts the step size depending on the *GAP* in the subgradient method.
$\tau = 1.05$	prevents the step size from becoming too small in the subgradient method.
$maxIter = 100$	maximum number of iteration for the subgradient method.

Table 2 contains the overall results of the branch-and-bound algorithm for each group-class of instances. These results show the quality of the branch-and-bound lower bounds (LB_{bb}), which are compared with lower bounds obtained by CPLEX (LB_{cplex}), solving an integer reduced OCARP model, neglecting all subtour elimination constraints. The trivial lower bounds (LB_0) and the best lower bounds ($LB_{best} = \max\{LB_{cplex}, LB_{bb}\}$) are also discriminated for a thorough comparison analysis. From the set of 411 instances, 216 solutions (52.55%) were proven optimal ($UB = LB_{best}$).

Considering the average deviation from lower bound ($\Delta LB = \frac{UB-LB}{LB}$) in Table 2, both LB_{bb} and LB_{cplex} performed equally to group of instances *ogdb*, not improving the trivial bounds. However, this group has only three instances yet to attain optimality, leaving a small margin for improvements. Considering groups *oval* and *oA*, the branch-and-bound yielded average deviations of 2.29% and 2.10%, respectively. CPLEX performed better for these groups, obtaining average deviations of 1.54% and 1.94%, respectively, and matching all best lower bounds deviations ($LB_{cplex} = LB_{best}$) for all instances in *oval*. On the other hand, the branch-and-bound outperformed CPLEX for groups *oegl* and *oB*, reducing the trivial lower bound deviations from 11.41% and 2.67% to 9.08% and 1.29%, respectively, against the CPLEX deviations of 10.89% and 2.10%, respectively. Moreover, the branch-and-bound had the best performance in the overall comparison, achieving an average deviation of 2.88%, while CPLEX yielded an average deviation of 3.12%, and the trivial deviation was 3.79%.

Table 2 also shows the number of best lower bounds (n_{best}) for each group-class of instances. These numbers draw practically the same conclusions as the average deviations from lower bounds. That is, for group *ogdb*, methods branch-and-bound and CPLEX performed equally; for groups *oval* and *oA*, CPLEX yielded 102 and 90 best lower bounds, respectively, against the branch-and-bound 77 and 89 best lower bounds, respectively; for groups *oegl* and *oB*, the branch-and-bound outperformed CPLEX with twice 72 best lower bounds, against the CPLEX 31 and 56 best bounds, respectively; lastly, for the overall comparison, the branch-and-bound delivered the highest number of best lower bounds, 379, while CPLEX provided only 348. This represents a meaningful difference of 8.91%, reinforcing the advantage of the branch-and-bound approach.

The performance complementarity between both algorithms has been registered by the overall best lower bounds deviations (2.62%) and number of best lower bounds (411), which are significantly better than the results presented by each algorithm alone. One cause of this complementarity could be the absense of capacity lower bounding schemes in the branch-and-bound algorithm. It may happen that the *oval* and *oA* instances are more susceptible to capacity constraints than node degree constraints. If this is the case, the OCARP model capacity constraints should have benefited CPLEX. Still, the reduced integer linear model was too onerous to solve the *oegl* and *oB* instances, which explains CPLEX poor performance to these groups.

Table 2: Branch-and-bound overall results.

		ΔLB				n_{best}			
group	class	ΔLB_0	ΔLB_{cplex}	ΔLB_{bb}	ΔLB_{best}	LB_0	LB_{cplex}	LB_{bb}	LB_{best}
ogdb	M^*	0.07	**0.07**	**0.07**	0.07	23	23	23	23
	M^*+1	0.05	**0.05**	**0.05**	0.05	23	23	23	23
	M^*+2	0.02	**0.02**	**0.02**	0.02	23	23	23	23
	overall	0.05	**0.05**	**0.05**	0.05	69	69	69	69
oval	M^*	3.38	**1.43**	3.16	1.43	16	34	16	34
	M^*+1	2.27	**1.75**	2.24	1.75	28	34	28	34
	M^*+2	1.46	**1.46**	1.46	1.46	33	34	33	34
	overall	2.37	**1.54**	2.29	1.54	77	102	77	102
oegl	M^*	18.50	17.38	**15.37**	15.37	9	9	24	24
	M^*+1	9.00	8.71	**6.75**	6.75	11	11	24	24
	M^*+2	6.72	6.58	**5.13**	5.13	11	11	24	24
	overall	11.41	10.89	**9.08**	9.08	31	31	72	72
oA	M^*	5.53	**3.07**	3.45	2.75	20	30	27	32
	M^*+1	2.38	**1.44**	1.72	1.35	26	30	30	32
	M^*+2	1.39	1.32	**1.14**	1.14	30	30	32	32
	overall	3.10	**1.94**	2.10	1.75	76	90	89	96
oB	M^*	5.72	4.23	**2.9**	2.90	13	16	24	24
	M^*+1	1.68	1.47	**0.73**	0.73	19	19	24	24
	M^*+2	0.61	0.61	**0.23**	0.23	21	21	24	24
	overall	2.67	2.10	**1.29**	1.29	53	56	72	72
overall	M^*	6.38	4.87	**4.8**	4.21	81	112	114	137
	M^*+1	3.00	2.56	**2.28**	2.07	107	117	129	137
	M^*+2	1.98	1.94	**1.57**	1.57	118	119	136	137
	overall	3.79	3.12	**2.88**	2.62	306	348	379	411

$LB_0 = \sum_{e \in R} c_e$: trivial lower bound. LB_{cplex}: cplex lower bounds.
LB_{bb}: branch-and-bound lower bounds. $\Delta LB_{best} = \min\{\Delta LB_{cplex}, \Delta LB_{bb}\}$: best lower bound.
ΔLB: average deviation from lower bound (%). n_{best}: number of best lower bounds.
M^*: minimum number of vehicles to attain a feasible solution.

8 Conclusions

An exact algorithm based on the branch-and-bound paradigm was developed to solve OCARP. The proposed algorithm used an equivalence between the OCARP and a problem related to minimum spanning tree, called *capacity and degree constrained minimum spanning forest* (CDCMSF). Deriving lower bounds to the CDCMSF seemed more intuitive, and this motivated the approach.

Three lower bounding procedures to the CDCMSF were implemented: the *minimum cost edges* (LB_1) relied on the number of missing edges to fulfill a solution; the *minimum cost spanning forest* (LB_2) computed adapting the Kruskal's algorithm; and the *degree constrained minimum spanning forest* (LB_3), which is NP-hard itself, but has a Lagrangian relaxation/subgradient method (Andrade et al. 2006) that efficiently computes good lower bounds. Each of these three lower bounds has a trade-off between the tightness of the bound and the computational effort to obtain it. The branch-and-bound stopping criterion is defined by three conditions: optimality, execution time, and memory use. After meeting with one of these criteria, the algorithm returns the best lower bound encountered in the process.

The computational complexity of the algorithm is exponential, meaning that the execution time may be too high for realistic size instances. Nonetheless, if not to solve all instances to optimality, the method at least improved the known trivial lower bounds, thus reducing the gaps in average.

Computational experiments with the exact algorithm were conducted using a benchmark set of 411 instances. The lower bounds obtained by the method proposed in this work were compared with the bounds generated solving an OCARP model with a high-end ILP solver, both equally time limited. The results showed that the branch-and-bound algorithm outperformed CPLEX in the overall average deviation from lower bounds and in the number of best lower bounds.

Acknowledgments

This work was supported by CNPq (Brazilian Research Agency).

References

Andrade, R., Lucena, A. and Maculan, N.: 2006, Using lagrangian dual information to generate degree constrained spanning trees, *Discrete Applied Mathematics* **154**, 703–717.

Beasley, J. E.: 1993, *Lagrangian relaxation*, John Wiley & Sons, Inc., New York, NY, USA, pp. 243–303.

Benavent, E., Campos, V., Corberán, A. and Mota, E.: 1992, The capacitated arc routing problem: lower bounds, *Networks* **22**, 669–690.

Beullens, P., Muyldermans, L., Cattrysse, D. and Oudheusden, D. V.: 2003, A guided local search heuristic for the capacitated arc routing problem, *European Journal of Operational Research* **147**, 629–643.

Bodin, L. and Levy, L.: 1989, The arc oriented location routing problem, *INFOR* **27**, 74–94.

Bodin, L. and Levy, L.: 1991, The arc partitioning problem, *European Journal of Operational Research* **53**, 393–401.

Corberán, A. and Prins, C.: 2010, Recent results on arc routing problems: An annotated bibliography, *Networks* **56**(1).

Cormen, T. H., Stein, C., Rivest, R. L. and Leiserson, C. E.: 2001, *Introduction to Algorithms*, 2nd edn, McGraw-Hill Higher Education.

Dror, M.: 2001, *Arc routing: theory, solutions and applications*, 1st edn, Kluwer Academic Press.

Eiselt, H. A., Gendreau, M. and Laporte, G.: 1995, Arc Routing Problems, Part I: The Chinese Postman Problem, *Operations Research* **43**(2), 231–242.

Golden, B. L., DeArmon, J. S. and Baker, E. K.: 1983, Computational experiments with algorithms for a class of routing problems, *Computers and Operations Research* **10**(1), 47–59.

Golden, B. L. and Wong, R. T.: 1981, Capacitated arc routing problems, *Networks* **11**, 305–315.

Hertz, A.: 2005, *Recent trends in arc routing. In Sharda R., Voß S., Golumbic M. C., Hartman I. B. A., Graph theory, combinatorics and algorithms*, Springer US.

Lacomme, P., Prins, C. and Ramdane-Chérif, W.: 2004, Competitive memetic algorithms for arc routing problems, *Annals of Operations Research* **131**, 159–185.

Li, L. Y. O. and Eglese, R. W.: 1996, An interactive algorithm for vehicle routing for winter-gritting, *Journal of the Operational Research Society* **47**, 217–228.

Moreira, L. M., Oliveira, J. F., Gomes, A. M. and Ferreira, J. S.: 2007, Heuristics for a dynamic rural postman problem, *Computers and Operations Research* **34**, 3281–3294.

Rodrigues, A. M. and Ferreira, J. S.: 2012, Cutting path as a rural postman problem: solutions by memetic algorithms, *International Journal of Combinatorial Optimization Problems and Informatics* **3**(1), 31–46.

Stern, H. I. and Dror, M.: 1979, Routing electric meter readers, *Computers and Operations Research* **6**, 209–223.

Usberti, F. L., França, P. M. and França, A. L. M.: 2011a, Grasp with evolutionary path-relinking for the capacitated arc routing problem, *Computers and Operations Research* . doi: 10.1016/j.cor.2011.10.014.

Usberti, F. L., França, P. M. and França, A. L. M.: 2011b, The open capacitated arc routing problem, *Computers and Operations Research* **38**(11), 1543 – 1555.

Wøhlk, S.: 2008, A decade of capacitated arc routing, *in* R. Sharda, S. Voß, B. Golden, S. Raghavan and E. Wasil (eds), *The vehicle routing problem: latest advances and new challenges*, Vol. 43, Springer US, pp. 29–48.

Wunderlich, J., Collette, M., Levy, L. and Bodin, L.: 1992, Scheduling meter readers for southern california gas company, *Interfaces* **22**, 22–30.