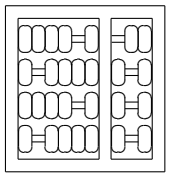


INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

A Branch-and-Cut-and-Price Approach for the Capacitated m-Ring-Star Problem

Edna A. Hoshino *Cid C. de Souza*

Technical Report - IC-10-15 - Relatório Técnico

May - 2010 - Maio

A Branch-and-Cut-and-Price Approach for the Capacitated m-Ring-Star Problem

Edna A. Hoshino* Cid C. de Souza†

Abstract

The Capacitated m-ring-star Problem is a variant of the classical one-depot capacitated vehicle routing problem in which a customer is either on a route or is connected to another customer or to some Steiner point present in a route. We develop a new exact algorithm for this problem using a branch-and-cut-and-price approach and compare its performance with that of a branch-and-cut algorithm proposed earlier in the literature. Computational results show that the new algorithm outperforms the branch-and-cut one in many instance classes.

1 Introduction

Let $G = (V, E \cup A)$ be a mixed graph, where E is the set of edges, defined for all pair of vertices in V, and A is the set of directed arcs called *connections*. Each edge e has a *routing cost* c_e and each arc ij has a *connection cost* w_{ij}. Vertices in V are partitioned in three groups: a special vertex 0, denoting a central depot, a set of customers U and a set of Steiner points W. Each arc in A is of the form uv where u is a customer and v is a vertex in $V \setminus \{0\}$.

A *ring-star* is a pair (R, S) where R is a subset of edges in E defining a cycle that includes the central depot, and S is a subset of arcs in $\{ij \in A : j \in V(R)\}$, where $V(X)$ denotes the set of vertices incident to the edges (arcs) in X. A ring-star is Q-*capacitated* if the number of customers in it is at most Q. The ring-star (R, S) covers a customer v if v belongs to $V(R)$ or there is a connection arc in S from v to some vertex in $V(R)$. The sets R and S of a ring-star are also called by *ring* and *star*, respectively.

In the *capacitated m-ring-star problem* (CmRSP) one has to find m Q-capacitated ring-stars covering all customers and minimizing the total sum of routing and connection costs. The CmRSP is easily seen to generalize the *Traveling Salesman Problem* and, therefore, is $\mathcal{NP}$-hard.

The CmRSP was introduced by Baldacci et al. [1] who describe an application in the design of a large optical fiber network. The authors proposed a branch-and-cut algorithm

*Faculty of Computing, University of Mato Grosso do Sul, Campo Grande MS, Brazil. Supported by Capes (Brazilian Ministry of Education).

†Institute of Computing, University of Campinas, Campinas SP, Brazil. Supported by CNPq – Conselho Nacional de Desenvolvimento Científico e Tecnológico – (Brazil) Grants # 301732/2007-8 and # 472504/2007-0.

for the problem and reported experiments where moderated-size instances were solved in reasonable time. To the best of our knowledge, this was the only exact algorithm available for the CmRSP until recently when, in [2], we investigated the adequacy of column generation to the problem. The resulting branch-and-price approach was validated by computational experiments that showed it is at least as good as the branch-and-cut algorithm. However, no dominance between the two approaches was observed.

Now, we can also view the CmRSP as a generalization of the classical one-depot Capacitated Vehicle Problem (CVRP). Some of the best results reported in the literature concerning the exact solution of the CVRP were obtained by a robust branch-and-cut-and-price (BCP) algorithm proposed in [3]. BCP algorithms embed cutting planes and column generation in a standard branch-and-bound procedure for solving Integer Programming (IP) problems. Encouraged by the success of this approach for the CVRP, in this work, we incorporate some of the cuts introduced in [1] to the branch-and-price (BP) algorithm we propose in [2]. This gives rise to a branch-and-cut-and-price algorithm whose performance is compared to that of a branch-and-cut algorithm for the CmRSP.

2 Mathematical Formulation

Let P be the set of all Q-capacitated ring-stars of an instance of CmRSP. A ring-star $p = (R, S)$ in P can be represented by two characteristic vectors r and s with dimensions $(|U| + |E|) \times 1$ and $|A| \times 1$, respectively. The components of r are such that r_i^p denotes the number of times vertex i belongs to $V(R)$ and r_{ij}^p the number of times edge (i, j) occurs in R. As for the elements in s, s_i^p is the number of times customer i is covered by p while s_{ij}^p refers to the number of times that arc ij occurs in S. Finally, if we associate a decision variable λ_p to p, a set covering model for CmRSP can be written as:

$$
\begin{aligned}
\text{(F)} \quad & \min \sum_{p \in P} c_p \lambda_p \\
\text{subject to} \quad & \sum_{p \in P} \lambda_p = m, && (1)\\
& \sum_{p \in P} (r_i^p + s_i^p)\lambda_p \geq 1, \ \forall i \in U, && (2)\\
& \sum_{p \in P} (\sum_{j \in V} r_{ij}^p)\lambda_p \leq 2, \ \forall i \in V', && (3)\\
& \sum_{p \in P} r_{ij}^p \lambda_p \leq u_{ij}, \ \forall (i,j) \in E, && (4)\\
& \sum_{p \in P} s_{ij}^p \lambda_p \leq 1, \ \forall ij \in A, && (5)\\
& \lambda_p \in \{0,1\}, \forall p \in P, && (6)
\end{aligned}
$$

where: c_p is the cost of the ring-star p, $V' = V \setminus \{0\}$ and $u_{ij} = 2$, if i or j is the depot, otherwise $u_{ij} = 1$. Notice that the former case is needed to allow rings of length two. Constraint (1) fixes the number of ring-stars to be selected from P. The covering constraints in (2) force each customer to be in a ring or covered by a star. Constraints

(3) forbid customers to be simultaneously in two or more rings. Constraints (4) limit the occurrences of an edge in a ring while constraints (5) do the same for an arc and a star. Finally, constraints (6) restrict the λ variables to assume 0–1 values depending on whether or not the associated *ring-star* is in the solution. Notice that, strictly speaking, one could think of representing a column by a binary vector indicating solely the vertices in the ring and in the star. By doing that, constraints (3), (4) and (5) become unnecessary to the model description. However, supported by some preliminary computational results, we decided to use the formulation above which allows us to establish a one-to-one correspondence between a column and a ring-star.

Since the number of variables in model (F) grows exponentially with the number of customers and Steiner points, it is hard even to compute its linear relaxation directly. Column generation is a classical and handy way of tackling this situation. Basically, the idea consists in solving iteratively two problems: the linear relaxation of the *restricted master problem* (RMP), that is the model (F) constrained only to the columns associated to a subset of P, and the *pricing problem* to generate new columns to add to the RMP. This technique is described in many textbooks on IP (cf., [4]).

The pricing problem Let π, μ, ν, β and α be the dual variables related to constraints (1), (2), (3), (4) and (5), respectively. Given a solution of the linear relaxation of the RMP, let $\tilde{c}_{ij} = c_{ij} - \beta_{ij} - k_i - k_j$, where $k_0 = 0$, and $k_i = \nu_i$ for $i \neq 0$, $\tilde{w}_{ij} = w_{ij} - \mu_i - \alpha_{ij}$ and $\tilde{p}_i = \mu_i$. Then, the reduced cost of a ring-star p is $\bar{c}_p = \sum_{e \in E} \tilde{c}_e r_e^p + \sum_{ij \in A} \tilde{w}_{ij} s_{ij}^p - \sum_{i \in U} \tilde{p}_i r_i^p + \pi$. The pricing problem of (F) requires the computation of $\min_{p \in P} \bar{c}_p$, that is a generalization of the Profitable Tour Problem [5].

The relaxed pricing problem To deal with the $\mathcal{NP}$-hardness of the pricing problem, we relaxed it, as in [3] for the CVRP, to yield more general structures which include the ring-stars. In [2], we analyzed three kinds of relaxations and developed pseudo-polynomial time algorithms to solve them. Here, we review the relaxation that led to the best performance of the BP algorithm. To explain the idea we first introduce some additional notation.

The principle behind this relaxation is that ring-stars are relaxed to allow for vertex repetitions. A solution to the subproblem is then represented by a string of vertex labels as follows. Given a relaxed ring-star, an initial string is built whose first element corresponds to the depot, while the remaining elements correspond to the sequence of labels of the vertices visited when the ring is traversed in an arbitrarily chosen direction (starting from the depot). Finally, the label of all vertices in the star are inserted to the string immediately before the position of the vertex to which it is connected (you may assume that the labels are inserted in alphabetical order whenever two or more vertices of the star are connected to the same vertex of the ring). Now, given a string $s = \{x_1, x_2, \ldots, x_t\}$, and integers $i \in [1, t-1]$ and $k \in [1, t-i]$, the substring $\{x_i, \ldots, x_{i+k}\}$ is called a *k-stream* of s if $x_i = x_{i+k}$. Figure 1 illustrates the definitions in this paragraph.

We are now able to describe the best relaxation according to the results we obtained in [2]. It consists in finding a 3-*stream-free relaxed Q-capacitated ring-star* whose reduced cost is minimum. For simplicity, in the discussion that follows, a 3-stream-free relaxed

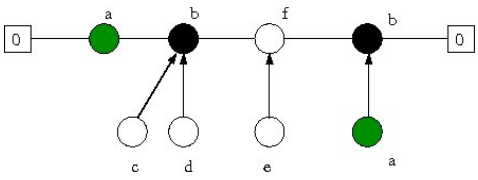


Figure 1: The string representation of a relaxed ring-star with a 6-stream ($acdbefa$).

Q-capacity ring-star is just called a *relaxed ring-star*.

The pricing problem arising from the discussion above is solved by a dynamic programming algorithm having as one of its key elements the usage of a dominance rule responsible for the identification and the elimination of non-useful relaxed ring-stars, i.e., ring-stars that are not necessary to produce an optimal solution. In [2], we show how to accomplish this task by means of a deterministic finite automaton (DFA). Figure 2 shows partially a digraph, called **intersection digraph**, that represents the transition diagram of the DFA associated to the problem of identify non-useful relaxed ring-star. White nodes represent states whose transitions are not completely described in the figure. The complete transition diagram can be derived from the intersection digraph proposed by Irnich and Villeneuve in [6].

The alphabet of the DFA consists of all relaxed ring-stars and an input string of the DFA is a sequence of those relaxed ring-stars in non-decreasing order of reduced cost.

To identify non-useful ring-stars, we just need to consider the two predecessors of the last vertex in the string associated to each ring-star. If the depot is one of these predecessors, we represent the ring-star just by the other predecessor (see arc labels in the transition diagram of Figure 2).

Each state of the DFA represents the result of intersections of self-hole sets (see [6] for a definition) of the relaxed ring-stars in any dipath from DFA's initial state until it. A relaxed ring-star of the sequence is non-useful if there is no transition for it on the DFA. In this case, we delete it from the sequence, keep in the current state and continue processing the next ring-star.

The DFA implementation was instrumental in the performance of the BP algorithm. As a matter of fact, it led to a reduction of the overall processing time to almost 30% [2]. The relaxed subproblem we described in this section together with the DFA-based algorithm discussed above are among the main ingredients of the branch-and-cut-and-price algorithm we develop to solve the CmRSP. This algorithm is further detailed next.

3 Implementation details and computational experiments

This section is devoted to the description of the algorithms we implemented, especially the BCP, the computational environment, the instance classes, the tests we carried out with the algorithms and the analysis of the results.

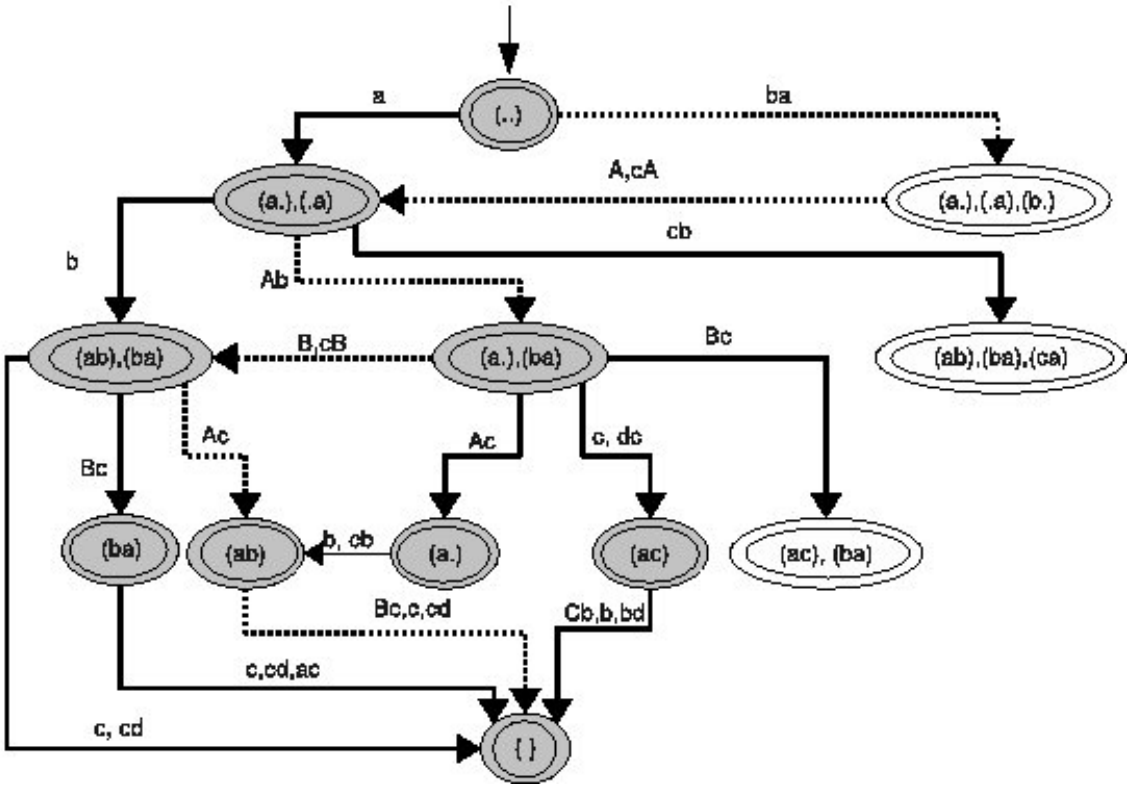


Figure 2: Example of a intersection digraph representing the transition diagram of the DFA.

The algorithms We implemented the BCP algorithm using the same branching rules, node selection criterion, initial basis, and pricing algorithm as we did for the BP algorithm introduced in [2]. For completeness, we briefly discuss some of these issues below.

In our tests, the initial basis is composed of artificial columns and populated with additional columns generated by a naïve heuristic based on a greedy randomized strategy akin to the construction phase of a greedy randomized adaptive search procedure or GRASP for short (cf., [7]). The traditional best bound strategy is used for node selection criterion while the formula presented by Lasdon in [8] is computed to obtain dual bounds at each node of the enumeration tree. In our implementation, according to the discussion in the previous section, the relaxed pricing problem is solved by a dynamic programming algorithm with the dominance rule implemented through the DFA-based algorithm.

The branching rule implemented in the BCP algorithm is derived from the work of Lysgaard et al. [9] and is based on the following valid inequality (known as C_i-ring-capacity cut) for the CmRSP:

$$\sum_{e\in\delta(S)}\sum_{p\in P} r_e^p \lambda_p \geq 2\left\lceil \frac{|\{i\in U|\hat{C}_i\subseteq S\}|}{Q}\right\rceil, S\subseteq V', S\cap U\neq\emptyset, \tag{7}$$

where $C_i=\{j\in V : ij\in A\}$, $\hat{C}_i = C_i\cup\{i\}$ and $\delta(S)$ denotes the set of edges with precisely

one end node in S. The term $|\{i \in U | \hat{C}_i \subseteq S\}|$ refers to the number of customers that can be covered only by vertices in S. Thus, $\left\lceil \frac{|\{i \in U | \hat{C}_i \subseteq S\}|}{Q} \right\rceil$ is a lower bound on the number of ring-stars necessary to cover customers in S. Therefore, the right-hand side (RHS) of inequality (7) defines the minimum number of edges of $(S, V \setminus S)$ that belong to a ring in any feasible solution.

Denote by $f(S)$ the difference between the left-hand side and the RHS of inequality (7). Since the RHS is always even, a fractional solution λ^* violates this inequality if $0 < f(S) < 2$. In this case, the following branching cuts can be used: $f(S) = 0$ and $f(S) \geq 2$. The heuristic to separate these branching cuts are based on the separation routines developed for the branch-and-cut algorithm described in [1] and is presented in Algorithm 1. As the separation routines were originally designed to work on the compact model of Baldacci et al., the variables of that model have to be computed from the λ variables of the set covering model. Before we show how this is done, we first have to define the variables of the compact model. This will be also useful for the presentation of the valid inequalities used as cuts in our BCP algorithm.

The variables of the compact model are divided into three sets. The x variables are defined for each edge (i, j) of E and are binary, except when one of the end points of the edge is the depot, in which case, the variable can also be set to 2. The value of x_{ij} is 1 if and only if the edge (i, j) belongs to a ring. If $x_{ij} = 2$ and, say, i is the depot, we have a ring formed solely by the depot and the vertex j. The y variables are defined for each vertex i of G in such a way that y_i is 1 if i belongs to a ring and 0 otherwise. Finally, the z variables are defined for each arc ij in A with z_{ij} taking value 1 if the vertex (customer) i is connected to vertex j in the solution. With these definitions, these three groups of variables can be easily computed from the λ variables of the (F) model through the formulas: $x_e = \sum_{p \in P} r_e^p \lambda_p$, $y_i = (\sum_{p \in P} \sum_{j \in \delta(i)} r_{ij}^p \lambda_p)/2$ and $z_{ij} = \sum_{p \in P} s_{ij}^p \lambda_p$. Notice that depending on the λ values these variables can be fractional.

Now, let us come back to Algorithm 1 that implements the branching rule of our BCP algorithm. The basic idea is to find a subset S of vertices for which inequality (7) is "more" violated, i.e., for which $f(S)$ is as close to 1.0 as possible. According to the formulas presented earlier, the x variables of the compact formulation are precomputed from the current fractional solution λ^* in lines (1) and (2). For each vertex u, the loop from lines (4) to (8) calls the procedure `ComputeCuts`(x^*, u, G) to construct a subset S' containing u and such that the value of $f(S')$ is close to 1.0. Among those, the subset S for which $f(S)$ gets closer to 1.0 is selected to define the branching constraints in lines (9) to (12).

Procedure ComputeCuts(x^*, u, G) is detailed in Algorithm 2. It employs a greedy strategy based on the weight function $f : V \mapsto \mathbb{R}$ which, to each vertex i in the graph, computes the increase in the value of $f(S)$ if i is included in the current set S. The next vertex entering S is chosen as being the one that brings $f(S)$ closer to 1.0. After that, the weight function is updated. The process is repeated until no vertex exits that improves the value of $f(S)$.

The computations on lines (1) to (3) are $O(|A|)$, while lines (4) to (7) consume $O(|V|^2)$ time. The loop at line (9) repeats at most $|V|$ times and, since the internal operations spend $O(|V|)$, all computations on lines (9) to (25) take $O(|V|^2)$. Thus, the complexity time of

Algorithm 1: Heuristic of separation for branching cuts.

Input: fractional solution λ^*, graph $G = (V, E \cup A)$
Output: branching cuts
for $ij \in E$ **do**
  $x^*_{ij} = \sum_{p \in \mathcal{P}} r^p_{ij} \lambda^*_p$
end
$fS = \infty$;
for $u \in U$ **do**
  (S', fS') =ComputeCut(x^*, u, G);
  if $|fS - 1.0| > |fS' - 1.0|$ **then**
    $fS = fS'$;
    $S = S'$;
  end
end
if $fS > 0$ *and* $fS < 2$ **then**
  Return the branching cuts:;
  $\sum_{ij \in \delta(S)} \sum_{p \in \mathcal{P}} r^p_{ij} \lambda_p = 2 \left\lceil \frac{|\{i \in U | \hat{C}_i \subseteq S\}|}{Q} \right\rceil$;
  $\sum_{ij \in \delta(S)} \sum_{p \in \mathcal{P}} r^p_{ij} \lambda_p \geq 2(\left\lceil \frac{|\{i \in U | \hat{C}_i \subseteq S\}|}{Q} \right\rceil + 1)$;
else
  fail.;
end

the heuristic to separate the branching cuts is $O(|U||V|^2)$.

Notice that the heuristic in Algorithm 1 may fail to find branching cuts with left-hand sides similar to that of inequality (7). In this case, the branching rule splits the solution space into two subspaces by forcing an edge (i, j) either to be in a ring or not. To do so, the constraints $\sum_{p \in \mathcal{P}} r^p_{ij} \lambda_p \geq 1$ and $\sum_{p \in \mathcal{P}} r^p_{ij} \lambda_p \leq 0$ are added to the formulation of the appropriate branches.

We now focus on the cutting planes generated in the BCP algorithm. To strengthen the linear relaxations of the set covering model (F), three families of valid inequalities proposed in [1] are used: the connectivity inequalities, the ring multistar inequalities and the rounded ring-capacity inequalities. Below, we briefly describe these families and comment on their respective separation routines.

The connectivity inequalities read:

$$\sum_{e \in \delta(S)} x_e + 2 \sum_{j \in C_u \setminus S} z_{uj} \geq 2, \forall S \subseteq V', \forall u \in S \cap U. \tag{8}$$

Given u and S, the inequality simply states that at least two edges of a ring intersect the cutset $(S, V \setminus S)$ or there exists an arc joining u to some vertex of C_u outside S.

The validity of the inequalities presented next can be derived from the capacity Q of a feasible ring-star. Essentially, for a given vertex set S, these inequalities force the number of

Algorithm 2: Algorithm ComputeCuts.

Input: fractional solution x^*, vertex u, graph G
Output: subset $S \subset V'$ and the value $f(S)$

$S = \hat{C}_u$;
$\mathtt{b}[i] = |\hat{C}_i \setminus S|, \forall i \in U$;
$\mathtt{d}[j] = |\{i \in U : \mathtt{b}[i] = 1 \text{ e } j \in \hat{C}_i\}|, \forall j \in V'$;
$fS = \sum_{ij \in \delta(S)} x^*_{ij} - 2\left\lceil \frac{|\{i \in U : \mathtt{b}[i] = 0\}|}{Q} \right\rceil$;
$r = mod(|\{i \in U : \mathtt{b}[i] = 0\}|, Q)$;
/* compute the contribution of each vertex to fS */;
for $i \in V' \setminus S$ **do**
    $f[i] = \sum_{j \notin S} x^*_{ij} - \sum_{j \in S} x^*_{ij}$;
end
$\mathtt{dif} = \min_{i \in V' \setminus S} |fS + f[i] - 2\big(\left\lfloor \frac{r + \mathtt{d}[i] - 1}{Q} \right\rfloor \big) - 1.0|$;
while $\mathtt{dif} < |fS - 1.0|$ **do**
    $\mathtt{best} = \arg\min_{i \in V' \setminus S} |fS + f[i] - 2\big(\left\lfloor \frac{r + \mathtt{d}[i] - 1}{Q} \right\rfloor \big) - 1.0|$;
    /* Update fS */;
    $fS = fS + f[\mathtt{best}] - 2\big(\left\lfloor \frac{r + \mathtt{d}[\mathtt{best}] - 1}{Q} \right\rfloor \big)$;
    $r = mod(r + \mathtt{d}[\mathtt{best}] - 1, Q)$;
    $S = S + \{\mathtt{best}\}$;
    /* Update individual contribution of each vertex */;
    if $\mathtt{best} \in U$ **then**
        $\mathtt{b}[\mathtt{best}] = \mathtt{b}[\mathtt{best}] - 1$;
        if $\mathtt{b}[\mathtt{best}] = 1$ **then**
            for $j \in \hat{C}_{\mathtt{best}} \setminus S$ **do**
                $\mathtt{d}[j] = \mathtt{d}[j] + 1$;
            end
        end
    end
    for $i \in U : \mathtt{best} \in \hat{C}_i$ **do**
        $\mathtt{b}[i] = \mathtt{b}[i] - 1$;
        if $\mathtt{b}[i] = 1$ **then**
            $\mathtt{d}[i] = \mathtt{d}[i] + 1$;
        end
    end
    for $i \in V' \setminus S$ **do**
        $f[i] = f[i] - 2x^*_{i,\mathtt{best}}$;
    end
    $\mathtt{dif} = \min_{i \in V' \setminus S} |fS + f[i] - 2\big(\left\lfloor \frac{r + \mathtt{d}[i] - 1}{Q} \right\rfloor \big) - 1.0|$;
end
Return (S, fS);

edges belonging simultaneously to the rings of a feasible solution and to the cutset $(S, V \setminus S)$ to exceed a certain amount that is a function of Q. Somehow, they all originate from the inequality below which is part of the compact formulation given in [1]:

$$\sum_{e \in \delta(S)} x_e \geq \frac{2}{Q}\Big(\sum_{i \in U \cap S} y_i + \sum_{i \in U} \sum_{j \in S \cap C_i} z_{ij}\Big), \forall S \subseteq V', S \neq \emptyset. \tag{9}$$

The ring multistar inequalities are defined by:

$$\sum_{e \in \delta(S)} x_e \geq \frac{2}{Q}\Big(\sum_{i \in U \cap S} y_i + \sum_{i \in U} \sum_{j \in S \cap C_i} z_{ij} + \sum_{i \in U \setminus S} \sum_{j \in S} x_{ij}\Big), \forall S \subseteq V'. \tag{10}$$

The rounded ring-capacity inequalities are obtained from rounded versions of inequality (9). Three types of such inequalities were used in our implementation: the C_i-ring-capacity inequalities, the rounded-capacity inequalities I and the rounded-capacity inequalities II. The first one corresponds to inequality (7) presented earlier. The rounded-capacity inequalities I can be expressed as:

$$\sum_{e \in \delta(S)} x_e + \frac{2}{mod(|U|, Q)}\Big(\sum_{i \in U} \sum_{j \in C_i \setminus S} z_{ij} + \sum_{i \in U \setminus S} y_i\Big) \geq 2 \left\lceil \frac{|U|}{Q} \right\rceil, \forall S \subseteq V', S \cap U \neq \emptyset. \tag{11}$$

Finally, the rounded-capacity inequalities II are given by:

$$\sum_{e \in \delta(S)} x_e \geq 2\Big(\left\lceil \frac{|U \cap S|}{Q} \right\rceil - \frac{\sum_{i \in U \cap S} \sum_{j \in C_i \setminus S} z_{ij}}{mod(|U \cap S|, Q)}\Big), \forall S \subseteq V', S \cap U \neq \emptyset. \tag{12}$$

The procedures given in [1] were used to separate inequalities (8), (10) and (11). Given a fractional solution λ, we compute the values of the variables x, y and z of the compact model, and construct a graph $\overline{G} = (\overline{V}, \overline{E})$ with edge capacity $\overline{c} : \overline{E} \mapsto \mathbb{R}$. The separation routines consist in solving a s-t min-cut problem over $\overline{G}$ to find the most violated inequality, if one exists. For example, to separate connectivity inequalities, we must consider $\overline{V} = \{i \in V : y_i > 0\} \cup \{0\}$, $\overline{E} = \{e \in E : x_e > 0\}$, $s = 0$, $t = u$, $\overline{c}_e = x_e$, $\forall e \notin \delta(u)$, and $\overline{c}_{uj} = x_{uj} + 2z_{uj}$. So, given a minimum 0-$u$ cut $(S, \overline{V} \setminus S)$ with $u \in S$, the subset S defines the most violated inequality (8) if the cut capacity is strictly smaller than two. A similar approach is applied to separate (10) and (11) (see [1] for details). On the other hand, we apply a simple heuristic to separate inequalities (7) and (12) which just checks if they are violated by the subsets S generated by the separation routines for (8), (10) and (11).

We also implemented a branch-and-cut algorithm, herein after denoted by BC, based on the ideas presented in [1]. The cuts and the respective separation routines used in the BCP algorithm are also part of BC. However, contrarily to what is done for the branch-and-cut algorithm in [1], we have implemented neither a strong branching nor a primal heuristic in both our algorithms.

Computational environment All our programming was done in C language using `gcc 4.1.2` compiler. To implement the BC algorithm we used the libraries provided by XPRESS-MP [10] version `17.01.01`. In the BCP algorithm, XPRESS was used solely to compute linear relaxations. The experiments were ran on a Pentium IV 3.4 GHz and 4Gb of RAM and all the running times are reported in seconds. We limited the running time for all experiments to 1800 seconds.

Instance classes Both algorithms were tested on three instance classes: A, B, and C. Classes A and B are defined as in [1]. Each class has 91 instances generated from TSPLIB instances [11] named `eil51.tsp`, `eil76.tsp`, and `eil101.tsp`. The instances `eil26.tsp` contain 26 vertices and are generated from the first 26 points of `eil51.tsp`. Each TSPLIB instance contains coordinates of a set of points in the plane. Each point corresponds to a vertex of the graph. The first point defines the coordinates of the depot, the next $|U|$ points are associated to customers and the remaining points determine the Steiner ones. To each TSPLIB instance, by varying α in $\{0.25, 0.5, 0.75, 1.0\}$ and m, the size of the fleet, in $\{3, 4, 5, 7, 10, 14\}$, we construct a group of CmRSP instances, where the total of customers $|U|$ is given by $\lfloor \alpha(n-1) \rfloor$, $|W| = n - |U| - 1$ and $Q = \left\lceil \frac{|U|}{0.9m} \right\rceil$. Obviously, instances with Q less than 2 are ignored. Routing cost and connection costs for each pair of vertices i and j are defined by using the Euclidean distance e_{ij} between the points associated to these vertices. Both costs are integer and obtained by some rounding involving the Euclidean distance. We consider two types of edge weights specified in the TSPLIB [12]: EUC_2D and CEIL_2D. For a given pair of vertices (i, j), using CEIL_2D we have $c_{ij} = \beta * \lceil e_{ij} \rceil$ and $w_{ij} = (10 - \beta) * \lceil e_{ij} \rceil$, whereas using EUC_2D we compute $c_{ij} = \beta * \lfloor e_{ij} + 0.5 \rfloor$ and $w_{ij} = (10 - \beta) * \lfloor e_{ij} + 0.5 \rfloor$. For classes A and C we have $\beta = 5$ and for class B $\beta = 7$. A connection arc is actually created only if its connection cost is at most equal to $f * \sum_{i \in U} \sum_{j \in V'} w_{ij} / (|U| * (|V| - 2))$. For classes A and B, we used $f = 0.2$ as in [1], and for class C we fixed $f = 0.5$ which, in principle, leads to instances having more connection arcs. In [1], classes A and B are generated using the procedure above, except that the size of the fleet is considered in $\{3, 4, 5\}$, and the distance EUC_2D is adopted. In a personal communication, though not required in the original statement of the problem, the authors informed that the ring-stars considered in their solutions are forced to be canonicals. Canonical ring-stars are those where a Steiner point appears in a ring only if there exists one connection arc in the star linking some customer to it. It is easy to prove that, when the routing costs satisfy the triangle inequalities, there exists an optimal solution composed exclusively by canonical ring-stars. Contrarily to the CEIL_2D weights, there is no warranty that the weights EUC_2D satisfy the triangle inequalities. Thus, initially, we used the weights CEIL_2D to compute the routing and connection costs for all instance classes. But, to make possible the comparison between the results of BCP and those reported in [1], we also experimented with instances having EUC_2D weights. These tests are discussed in the end of this section.

It is worth mentioning that the entire benchmark used in our experiments with the known optima is available in [13].

Results and analysis As expected, preliminary experiments confirmed that the BP from [2] is dominated by BCP, i.e., it is worthwhile to combine the cutting-plane and the column generation procedures to solve the CmRSP. Therefore, our comparative analysis is restricted to the BC and to the BCP algorithms. Tables 1, 2, 3, 4, 5 and 6 show the results of the BCP and the BC algorithms for classes A, B and C. Each instance is represented by the columns `inst`, `m`, `U` and `Q`, that correspond to the name of the instance, the number of ring-stars, the number of customers and the capacity of the vehicle, respectively. The value of the best integer solution found by all the algorithms, including that of the primal heuristic used to populate the initial basis, is presented in column z^*. Column `gap` reports the gap percentage of the dual bound obtained by the algorithm with respect to z^*. The running time is showed in column `time`. Column `nodes` exhibits the total number of nodes processed in the branch-and-bound tree while column `root` displays the percentage of z^* in respect of the dual bound obtained at the root node for each algorithm. The superscripts 1 and 2 in columns `gap`, `time`, `node` and `root` are used to refer to algorithms BCP and BC, respectively.

As can be inspected in columns $\texttt{root}^1$ and $\texttt{root}^2$, the dual bounds obtained by the BCP algorithm are tighter than those of the BC algorithm. On average, the improvement observed in the dual bound was of about 11% in favor of the BCP algorithm in all instance classes. This helped the BCP algorithm to solve more instances than BC and also to get smaller final gaps as can be better appreciated in Table 7. In the later table column BEST BOUND shows the percentage of instances in which BCP provided a better dual bound at root node than BC. The three next columns under the head UNSOLVED, display the total of instances that were not solved to optimality by both algorithms (BOTH), by BCP and by BC, respectively. Finally, column BEST GAP presents the percentage of instances that remained unsolved by both algorithms but for which BCP got smaller final gap than BC. These results clearly show that besides obtaining better dual bounds at the root node in almost all cases, BCP was capable to prove the optimality of much more instances and, when this was not the case, to obtain smaller gaps than BC. Another observation is that instances in classes B and C seem to be more difficult than those in class A.

Another view of the results obtained by the algorithms for each instance class is given in Table 8 whose rows refer to groups of instances generated from the same original TSPLIB instance. Column GAP displays the average reduction in the percentage of duality gap obtained by BCP with respect to BC, taken over all the instances that remain unsolved by both algorithms after the execution time limit was reached. This gap was calculated relative to the best known primal solution. Now, considering only the instances whose optimum was proved by both algorithms, column TIME shows the speed-up rate defined as the average running time of BC divided by that of BCP. Finally, the contents of column OPT are of the form x/y where y ($y-x$) is the number of instances solved to optimality by BCP (BC), in other words, x is the number of instances solved by BCP in excess of that solved by BC. With these definitions, it is clear that the value in some cells may be undefined, which is denoted by a "*".

In all classes, the BCP outperformed BC in the number of instances solved to optimality, running time and final gap. Although this cannot be deduced from the data displayed in Table 8, it is worth noting that BCP solved around 90% of instances faster than BC.

inst	m	U	Q	z*	gap[1]	gap[2]	time[1]	time[2]	nodes[1]	nodes[2]	root[1]	root[2]
eil26.tsp	3	6	3	178	0.0	0.0	0.0	0.1	1	3	178.0	173.9
eil26.tsp	4	6	2	198	0.0	0.0	0.0	0.1	1	3	198.0	195.0
eil26.tsp	5	6	2	215	0.0	0.0	0.0	0.0	1	1	215.0	215.0
eil26.tsp	3	12	5	254	0.0	0.0	0.0	0.8	2	23	252.5	246.7
eil26.tsp	4	12	4	271	0.0	0.0	0.0	0.3	1	6	271.0	268.6
eil26.tsp	5	12	3	304	0.0	0.0	0.0	0.1	1	3	304.0	300.8
eil26.tsp	7	12	2	373	0.0	0.0	0.0	0.1	1	3	373.0	368.9
eil26.tsp	10	12	2	429	0.0	0.0	0.0	0.0	1	1	429.0	429.0
eil26.tsp	3	18	7	312	0.0	0.0	0.7	4.2	25	108	306.7	297.5
eil26.tsp	4	18	5	349	0.0	0.0	0.1	2.5	2	30	348.5	339.7
eil26.tsp	5	18	4	387	0.0	0.0	0.1	3.8	5	79	385.5	370.2
eil26.tsp	7	18	3	462	0.0	0.0	0.0	1.6	1	44	462.0	449.0
eil26.tsp	10	18	2	590	0.0	0.0	0.0	0.3	1	5	589.5	586.2
eil26.tsp	14	18	2	680	0.0	0.0	0.0	0.1	1	1	680.0	680.0
eil26.tsp	3	25	10	346	0.0	0.0	4.1	6.4	58	266	339.9	336.5
eil26.tsp	4	25	7	379	0.0	0.0	0.2	2.3	3	55	378.2	372.0
eil26.tsp	5	25	6	398	0.0	0.0	0.1	2.0	1	35	398.0	390.4
eil26.tsp	7	25	4	490	0.0	0.0	0.1	8.1	2	79	489.0	474.4
eil26.tsp	10	25	3	601	0.0	0.0	0.0	4.1	1	78	601.0	575.7
eil26.tsp	14	25	2	771	0.0	0.0	0.0	0.1	1	1	771.0	771.0
eil51.tsp	3	12	5	254	0.0	0.0	0.2	47.2	3	195	252.3	237.0
eil51.tsp	4	12	4	271	0.0	0.0	0.1	2.3	1	11	271.0	254.7
eil51.tsp	5	12	3	303	0.0	0.0	0.1	3.2	1	11	303.0	280.8
eil51.tsp	7	12	2	373	0.0	0.0	0.1	3.8	1	11	373.0	337.8
eil51.tsp	10	12	2	420	0.0	0.0	0.1	1.6	1	7	420.0	398.0
eil51.tsp	3	25	10	343	0.0	0.0	5.9	86.5	43	690	338.4	328.5
eil51.tsp	4	25	7	378	0.0	0.0	1.3	80.4	16	352	376.6	347.7
eil51.tsp	5	25	6	395	0.0	0.0	0.2	4.6	1	13	395.0	371.4
eil51.tsp	7	25	4	489	0.0	0.0	0.2	175.9	3	0	488.0	446.7
eil51.tsp	10	25	3	595	0.0	0.0	0.1	47.6	1	175	595.0	535.6
eil51.tsp	14	25	2	769	0.0	0.0	0.1	5.3	1	19	769.0	717.5
eil51.tsp	3	37	14	406	0.0	0.0	34.2	6.9	27	42	400.8	396.8
eil51.tsp	4	37	11	437	0.0	0.0	7.2	71.4	16	478	434.5	417.2
eil51.tsp	5	37	9	467	0.0	0.2	67.8	1801.1	377	11237	459.1	436.4
eil51.tsp	7	37	6	547	0.0	1.1	5.8	1801.1	91	8143	541.4	490.5
eil51.tsp	10	37	5	626	0.0	1.1	10.0	1801.1	199	7725	619.4	583.3
eil51.tsp	14	37	3	829	0.0	0.0	1.0	898.5	39	3150	824.4	768.0
eil51.tsp	3	50	19	496	0.6	0.0	1800.2	136.2	351	1878	487.1	485.1
eil51.tsp	4	50	14	530	0.0	0.0	347.1	909.7	281	7876	523.1	510.2
eil51.tsp	5	50	12	560	0.0	0.5	400.6	1801.1	575	9625	553.5	531.9
eil51.tsp	7	50	8	648	0.0	2.2	225.8	1801.1	797	5847	640.4	596.9
eil51.tsp	10	50	6	754	0.0	2.6	80.5	1801.1	671	6385	747.1	686.7
eil51.tsp	14	50	4	954	0.0	2.0	35.3	1801.1	600	4993	948.3	864.7

Table 1: Results obtained by (1) BCP and (2) BC for small instances in class A

inst	m	U	Q	z*	gap[1]	gap[2]	time[1]	time[2]	nodes[1]	nodes[2]	root[1]	root[2]
eil76.tsp	3	18	7	338	0.0	3.0	201.9	1801.4	263	1311	325.9	262.5
eil76.tsp	4	18	5	399	0.0	4.5	16.7	1801.2	71	1201	392.7	296.7
eil76.tsp	5	18	4	460	0.0	3.1	12.6	1801.4	95	1149	454.1	322.9
eil76.tsp	7	18	3	558	0.0	3.7	0.6	1801.2	1	1317	558.0	390.5
eil76.tsp	10	18	2	746	0.0	1.4	0.4	1801.0	1	1373	746.0	545.6
eil76.tsp	14	18	2	814	0.0	1.2	0.4	1801.1	1	1459	814.0	635.5
eil76.tsp	3	37	14	469	10.1	10.9	1800.1	1801.7	315	1339	415.5	397.0
eil76.tsp	4	37	11	490	4.0	6.1	1800.0	1801.2	645	1885	460.2	419.7
eil76.tsp	5	37	9	501	0.0	0.4	2.8	1801.1	1	2171	501.0	433.8
eil76.tsp	7	37	6	641	0.0	4.7	1.0	1801.3	1	1795	641.0	489.9
eil76.tsp	10	37	5	748	0.0	5.6	3.4	1801.0	41	1697	746.0	584.7
eil76.tsp	14	37	3	1045	0.0	3.2	0.5	1801.1	1	1647	1044.0	825.7
eil76.tsp	3	56	21	575	15.0	14.1	1800.2	1801.2	107	3327	493.9	482.5
eil76.tsp	4	56	16	626	15.5	15.5	1800.2	1800.0	425	1343	536.3	510.7
eil76.tsp	5	56	13	584	0.0	1.6	1355.8	1801.3	439	2377	578.7	521.5
eil76.tsp	7	56	9	772	10.1	15.6	1800.4	1800.0	3323	951	693.7	597.9
eil76.tsp	10	56	7	824	0.0	5.4	436.5	1801.1	1450	1997	817.8	697.4
eil76.tsp	14	56	5	1030	0.0	4.6	124.0	1801.1	887	1941	1024.5	865.8
eil76.tsp	3	75	28	618	1.5	1.0	1800.2	1801.2	49	6733	607.0	599.7
eil76.tsp	4	75	21	758	17.9	17.7	1801.3	1801.1	147	4603	640.6	617.2
eil76.tsp	5	75	17	811	18.2	20.0	1800.8	1801.1	313	2953	682.8	634.6
eil76.tsp	7	75	12	882	11.5	16.5	1800.2	1801.1	1209	2113	785.4	698.7
eil76.tsp	10	75	9	1029	11.0	18.0	1800.3	1801.1	3225	1951	921.0	808.1
eil76.tsp	14	75	6	1180	0.0	6.6	769.9	1800.0	2385	1093	1173.8	971.9
eil101.tsp	3	25	10	381	0.0	0.3	48.4	1801.2	5	1025	377.1	341.2
eil101.tsp	4	25	7	433	0.0	3.3	134.6	1801.3	69	939	425.6	362.8
eil101.tsp	5	25	6	469	0.0	4.5	91.0	1801.3	145	911	461.5	384.8
eil101.tsp	7	25	4	576	0.0	4.7	40.5	1801.3	269	751	570.8	439.8
eil101.tsp	10	25	3	693	0.0	4.1	2.5	1801.8	4	769	691.3	541.9
eil101.tsp	14	25	2	905	0.0	2.0	1.0	1801.3	4	773	904.0	716.1
eil101.tsp	3	50	19	614	16.1	16.1	1801.0	1800.0	5	532	528.2	504.4
eil101.tsp	4	50	14	661	17.0	18.2	1800.2	1801.5	15	1157	563.5	519.5
eil101.tsp	5	50	12	706	18.9	21.3	1801.7	1801.2	19	1043	591.9	536.4
eil101.tsp	7	50	8	786	13.1	18.4	1800.1	1801.1	81	861	692.4	582.5
eil101.tsp	10	50	6	819	0.0	5.8	440.9	1801.6	107	857	811.9	664.4
eil101.tsp	14	50	4	1044	0.2	5.8	1800.1	1800.0	1415	354	1034.6	816.9
eil101.tsp	3	75	28	648	1.7	0.0	1801.3	1543.6	5	1365	633.2	624.6
eil101.tsp	4	75	21	801	19.6	19.7	1801.9	1800.0	13	848	669.5	641.6
eil101.tsp	5	75	17	855	21.4	22.8	1800.3	1801.3	35	1421	702.3	658.8
eil101.tsp	7	75	12	956	21.3	25.5	1800.6	1801.3	77	1051	784.7	700.3
eil101.tsp	10	75	9	1046	16.4	21.8	1800.3	1800.0	183	536	894.1	778.6
eil101.tsp	14	75	6	1293	15.2	21.9	1800.1	1801.2	1225	823	1117.1	923.6
eil101.tsp	3	100	38	838	17.4	16.9	1803.0	1801.3	1	2883	713.1	692.7
eil101.tsp	4	100	28	868	17.5	17.0	1815.2	1801.6	17	2697	738.2	716.6
eil101.tsp	5	100	23	909	17.9	18.1	1800.1	1801.2	47	1899	769.0	733.5
eil101.tsp	7	100	16	1018	19.3	22.1	1800.2	1801.2	195	1065	850.0	781.3
eil101.tsp	10	100	12	1119	17.1	20.7	1800.1	1801.3	595	1027	953.0	856.9
eil101.tsp	14	100	8	1350	15.0	20.6	1800.6	1801.2	1827	935	1169.0	1014.6

Table 2: Results obtained by (1) BCP and (2) BC for large instances in class A

inst	m	U	Q	z*	gap[1]	gap[2]	time[1]	time[2]	nodes[1]	nodes[2]	root[1]	root[2]
eil26.tsp	3	6	3	1246	0.0	0.0	0.0	0.1	1	3	1246.0	1217.2
eil26.tsp	4	6	2	1386	0.0	0.0	0.0	0.1	1	3	1386.0	1365.0
eil26.tsp	5	6	2	1505	0.0	0.0	0.0	0.0	1	1	1505.0	1505.0
eil26.tsp	3	12	5	1778	0.0	0.0	0.0	0.6	4	13	1767.5	1727.0
eil26.tsp	4	12	4	1897	0.0	0.0	0.0	0.3	1	6	1897.0	1878.3
eil26.tsp	5	12	3	2128	0.0	0.0	0.0	0.1	1	3	2128.0	2105.7
eil26.tsp	7	12	2	2611	0.0	0.0	0.0	0.1	1	3	2611.0	2582.3
eil26.tsp	10	12	2	3003	0.0	0.0	0.0	0.0	1	1	3003.0	3003.0
eil26.tsp	3	18	7	2184	0.0	0.0	0.7	3.6	33	101	2146.7	2074.6
eil26.tsp	4	18	5	2443	0.0	0.0	0.1	2.8	2	37	2439.5	2378.1
eil26.tsp	5	18	4	2709	0.0	0.0	0.1	2.6	2	78	2698.5	2591.1
eil26.tsp	7	18	3	3234	0.0	0.0	0.0	2.1	1	60	3234.0	3143.2
eil26.tsp	10	18	2	4130	0.0	0.0	0.0	0.3	1	5	4126.5	4103.5
eil26.tsp	14	18	2	4760	0.0	0.0	0.0	0.1	1	1	4760.0	4760.0
eil26.tsp	3	25	10	2422	0.0	0.0	4.2	3.9	49	252	2379.0	2355.5
eil26.tsp	4	25	7	2653	0.0	0.0	0.2	2.0	2	62	2647.4	2597.0
eil26.tsp	5	25	6	2786	0.0	0.0	0.1	1.8	1	35	2786.0	2732.6
eil26.tsp	7	25	4	3430	0.0	0.0	0.1	13.1	3	252	3423.0	3311.5
eil26.tsp	10	25	3	4207	0.0	0.0	0.0	3.8	1	64	4207.0	4029.8
eil26.tsp	14	25	2	5397	0.0	0.0	0.0	0.1	1	1	5397.0	5397.0
eil51.tsp	3	12	5	1778	0.0	0.0	0.1	59.4	3	359	1759.8	1634.5
eil51.tsp	4	12	4	1897	0.0	0.0	0.1	5.7	1	23	1897.0	1783.3
eil51.tsp	5	12	3	2121	0.0	0.0	0.1	4.8	1	15	2121.0	1978.7
eil51.tsp	7	12	2	2611	0.0	0.0	0.1	2.3	1	7	2611.0	2367.5
eil51.tsp	10	12	2	2940	0.0	0.0	0.1	2.6	1	9	2940.0	2789.5
eil51.tsp	3	25	10	2354	0.0	0.0	7.8	35.3	35	181	2329.6	2256.0
eil51.tsp	4	25	7	2606	0.0	0.0	0.8	47.6	9	110	2594.1	2448.7
eil51.tsp	5	25	6	2718	0.0	0.0	0.3	3.2	1	15	2718.0	2560.6
eil51.tsp	7	25	4	3400	0.0	0.0	0.3	380.7	6	1366	3390.3	3061.9
eil51.tsp	10	25	3	4111	0.0	0.0	0.1	31.3	1	155	4111.0	3786.2
eil51.tsp	14	25	2	5353	0.0	0.0	0.1	6.4	1	27	5353.0	5103.4
eil51.tsp	3	37	14	2795	0.0	0.0	157.9	15.8	77	62	2757.8	2706.4
eil51.tsp	4	37	11	3022	0.0	0.0	53.2	222.3	89	1440	2958.2	2836.7
eil51.tsp	5	37	9	3236	0.0	0.3	296.7	1801.2	997	7345	3162.6	2980.7
eil51.tsp	7	37	6	3775	0.0	1.3	3.2	1801.1	41	7237	3751.5	3428.4
eil51.tsp	10	37	5	4335	0.0	0.9	8.1	1801.1	157	7979	4295.0	4054.4
eil51.tsp	14	37	3	5759	0.0	0.0	1.6	990.9	71	3930	5726.6	5386.7
eil51.tsp	3	50	19	3393	0.5	0.0	1801.6	138.3	471	1230	3326.0	3286.0
eil51.tsp	4	50	14	3624	0.0	0.0	529.5	1729.1	273	10422	3583.5	3442.0
eil51.tsp	5	50	12	3853	0.0	1.4	365.0	1801.1	479	7465	3801.6	3567.1
eil51.tsp	7	50	8	4474	0.0	2.8	279.7	1801.0	977	5417	4431.8	4069.3
eil51.tsp	10	50	6	5216	0.0	3.0	84.3	1801.1	760	5979	5166.2	4707.1
eil51.tsp	14	50	4	6612	0.0	1.6	45.7	1800.0	922	2343	6574.7	6049.1

Table 3: Results obtained by (1) BCP and (2) BC for small instances in class B

inst	m	U	Q	z*	gap[1]	gap[2]	time[1]	time[2]	nodes[1]	nodes[2]	root[1]	root[2]
eil76.tsp	3	18	7	2319	0.0	3.2	357.5	1800.0	909	661	2242.3	1773.0
eil76.tsp	4	18	5	2729	0.0	3.1	22.6	1800.0	215	555	2687.7	2002.2
eil76.tsp	5	18	4	3172	0.0	4.5	20.5	1801.1	321	1225	3115.0	2209.8
eil76.tsp	7	18	3	3824	0.0	3.5	0.4	1801.1	3	1333	3822.2	2732.1
eil76.tsp	10	18	2	5137	0.0	2.1	0.3	1801.1	1	1503	5137.0	3806.5
eil76.tsp	14	18	2	5613	0.0	1.1	0.2	1801.3	1	1285	5613.0	4467.9
eil76.tsp	3	37	14	3242	10.3	11.8	1800.2	1801.2	549	2341	2893.8	2705.5
eil76.tsp	4	37	11	3807	16.6	21.3	1800.2	1801.2	1161	2133	3226.9	2850.0
eil76.tsp	5	37	9	3495	0.0	2.5	4.9	1801.2	5	2075	3494.9	2976.7
eil76.tsp	7	37	6	4451	0.0	6.0	1.0	1800.0	1	1691	4451.0	3440.9
eil76.tsp	10	37	5	5177	0.0	6.4	16.6	1801.2	147	1717	5157.0	3945.6
eil76.tsp	14	37	3	7235	0.0	4.5	1.9	1801.0	32	1661	7198.0	5520.1
eil76.tsp	3	56	21	3698	12.2	11.1	1800.5	1801.1	91	4733	3258.7	3182.2
eil76.tsp	4	56	16	4310	19.1	20.1	1800.0	1801.2	205	2471	3579.4	3328.8
eil76.tsp	5	56	13	4848	24.6	29.0	1801.0	1801.1	561	2187	3847.4	3439.1
eil76.tsp	7	56	9	5159	9.5	16.6	1800.7	1801.2	2369	2023	4665.5	3937.8
eil76.tsp	10	56	7	5541	0.0	5.7	183.1	1801.0	545	1973	5512.4	4464.7
eil76.tsp	14	56	5	7032	0.5	5.4	1800.2	1800.0	12030	908	6942.5	5769.6
eil76.tsp	3	75	28	4010	1.5	0.6	1800.9	1801.2	13	4831	3947.5	3849.7
eil76.tsp	4	75	21	4921	16.2	17.3	1800.9	1801.1	77	2513	4213.6	3992.4
eil76.tsp	5	75	17	5423	19.3	22.9	1800.3	1801.1	261	2107	4528.8	4137.0
eil76.tsp	7	75	12	5953	12.9	19.7	1800.2	1800.0	1071	2722	5241.1	4530.8
eil76.tsp	10	75	9	6930	11.8	18.5	1800.4	1801.2	2629	1773	6158.0	5176.7
eil76.tsp	14	75	6	9100	14.3	20.6	1800.3	1800.0	7189	1649	7904.9	6803.2
eil101.tsp	3	25	10	2629	0.0	3.2	71.6	1802.4	12	723	2608.4	2274.5
eil101.tsp	4	25	7	2972	0.0	5.1	72.4	1801.7	29	631	2951.4	2424.1
eil101.tsp	5	25	6	3237	0.0	5.8	67.7	1801.4	178	637	3184.2	2576.6
eil101.tsp	7	25	4	3986	0.0	6.3	416.8	1801.4	1578	509	3930.1	2954.2
eil101.tsp	10	25	3	4803	0.0	5.5	27.0	1801.5	193	513	4764.6	3630.9
eil101.tsp	14	25	2	6244	0.0	3.3	1.5	1801.4	4	555	6240.0	4886.1
eil101.tsp	3	50	19	4136	21.4	21.1	1800.1	1800.0	3	322	3405.7	3193.3
eil101.tsp	4	50	14	4432	18.4	22.5	1803.4	1801.3	9	667	3692.9	3290.0
eil101.tsp	5	50	12	4613	16.2	21.7	1800.5	1801.1	9	639	3968.9	3424.7
eil101.tsp	7	50	8	5351	13.6	21.4	1813.7	1801.8	123	523	4683.9	3815.3
eil101.tsp	10	50	6	6283	13.2	21.6	1800.3	1801.4	351	477	5521.1	4387.6
eil101.tsp	14	50	4	8180	15.3	24.1	1800.2	1802.6	1847	493	7056.2	5461.5
eil101.tsp	3	75	28	4707	21.0	16.0	1802.5	1801.2	7	957	3865.9	3843.9
eil101.tsp	4	75	21	5185	23.6	22.9	1803.2	1801.3	5	765	4156.7	3963.8
eil101.tsp	5	75	17	5424	20.5	23.6	1800.0	1801.2	7	701	4440.4	4096.9
eil101.tsp	7			6211	20.4	27.1	1800.5	1801.2	9	583	5072.1	4437.0
eil101.tsp	10	75	9	6926	16.5	22.9	1803.7	1801.2	231	521	5922.0	5011.7
eil101.tsp	14	75	6	8623	15.9	23.5	1800.6	1801.5	1403	477	7407.6	6018.1
eil101.tsp	3	100	38	5130	15.8	11.9	1802.2	1801.2	5	1159	4423.5	4375.5
eil101.tsp	4	100	28	5523	17.6	16.1	1803.4	1801.2	9	1057	4660.8	4487.0
eil101.tsp	5	100	23	5931	19.1	19.9	1800.6	1801.1	5	925	4922.1	4618.1
eil101.tsp	7	100	16	6677	18.9	23.3	1802.4	1801.5	71	689	5605.3	4973.9
eil101.tsp	10	100	12	7456	17.0	22.6	1800.4	1801.5	241	657	6357.9	5521.5
eil101.tsp	14	100	8	8871	13.0	19.2	1800.1	1801.9	565	519	7836.2	6443.4

Table 4: Results obtained by (1) BCP and (2) BC for large instances in class B

inst	m	U	Q	z*	gap[1]	gap[2]	time[1]	time[2]	nodes[1]	nodes[2]	root[1]	root[2]
eil26.tsp	3	6	3	159	0.0	0.0	0.0	0.0	1	1	159.0	159.0
eil26.tsp	4	6	2	177	0.0	0.0	0.0	0.0	1	1	177.0	177.0
eil26.tsp	5	6	2	193	0.0	0.0	0.0	0.0	1	1	193.0	193.0
eil26.tsp	3	12	5	226	0.0	0.0	0.1	0.6	1	9	226.0	223.2
eil26.tsp	4	12	4	243	0.0	0.0	0.1	0.5	1	11	243.0	239.7
eil26.tsp	5	12	3	270	0.0	0.0	0.0	0.1	1	3	269.3	269.3
eil26.tsp	7	12	2	324	0.0	0.0	0.0	0.0	1	1	324.0	324.0
eil26.tsp	10	12	2	406	0.0	0.0	0.0	0.0	1	1	406.0	406.0
eil26.tsp	3	18	7	289	0.0	0.0	3.4	14.8	19	300	282.3	269.6
eil26.tsp	4	18	5	324	0.0	0.0	1.1	6.3	38	106	320.8	308.5
eil26.tsp	5	18	4	353	0.0	0.0	0.1	1.7	1	31	352.8	340.1
eil26.tsp	7	18	3	414	0.0	0.0	0.1	1.8	12	45	412.0	400.9
eil26.tsp	10	18	2	500	0.0	0.0	0.0	0.0	1	1	500.0	500.0
eil26.tsp	14	18	2	625	0.0	0.0	0.0	0.0	1	1	625.0	625.0
eil26.tsp	3	25	10	327	0.0	0.0	3.1	3.8	9	75	323.9	312.5
eil26.tsp	4	25	7	362	0.0	0.0	5.4	75.7	49	1070	358.6	341.4
eil26.tsp	5	25	6	385	0.0	0.0	0.3	26.5	5	416	383.7	370.2
eil26.tsp	7	25	4	460	0.0	0.0	0.1	13.6	1	232	460.0	445.6
eil26.tsp	10	25	3	547	0.0	0.0	0.0	0.9	1	13	546.5	540.1
eil26.tsp	14	25	2	683	0.0	0.0	0.0	0.0	1	1	683.0	683.0
eil51.tsp	3	12	5	226	0.0	0.0	2.9	6.6	3	17	225.6	216.8
eil51.tsp	4	12	4	241	0.0	0.0	0.5	3.8	1	11	241.0	232.7
eil51.tsp	5	12	3	270	0.0	0.0	0.2	4.2	2	15	269.0	256.7
eil51.tsp	7	12	2	319	0.0	0.0	0.0	3.0	1	29	319.0	307.0
eil51.tsp	10	12	2	373	0.0	0.0	0.1	0.9	1	6	373.0	372.1
eil51.tsp	3	25	10	325	0.0	0.0	372.5	45.8	53	138	322.1	304.0
eil51.tsp	4	25	7	359	0.0	0.0	34.9	220.3	11	593	355.3	329.5
eil51.tsp	5	25	6	383	0.0	0.0	3.3	343.8	5	1316	381.2	354.8
eil51.tsp	7	25	4	457	0.0	0.0	0.4	161.5	1	332	457.0	422.1
eil51.tsp	10	25	3	539	0.0	0.0	0.3	41.8	2	231	538.5	513.3
eil51.tsp	14	25	2	669	0.0	0.0	0.2	2.6	1	7	669.0	655.0
eil51.tsp	3	37	14	397	0.0	0.0	1083.0	275.1	93	1148	392.8	379.0
eil51.tsp	4	37	11	427	0.5	0.0	1800.1	1799.0	295	6053	421.1	400.9
eil51.tsp	5	37	9	446	0.0	0.0	10.6	572.6	7	1702	444.0	425.8
eil51.tsp	7	37	6	530	0.0	1.1	39.3	1801.1	172	5549	525.1	481.3
eil51.tsp	10	37	5	598	0.0	0.2	8.6	1801.1	61	8023	594.5	551.8
eil51.tsp	14	37	3	765	0.0	0.0	1.1	137.6	28	446	762.7	727.1
eil51.tsp	3	50	19	648	37.6	36.7	1800.1	1801.1	157	13359	466.0	455.0
eil51.tsp	4	50	14	505	0.0	0.6	751.5	1801.1	239	6275	501.0	470.0
eil51.tsp	5	50	12	688	29.6	31.3	1800.7	1801.1	1017	5321	525.7	489.1
eil51.tsp	7	50	8	623	0.6	3.3	1800.0	1801.1	3725	5167	611.5	561.3
eil51.tsp	10	50	6	721	0.0	2.1	34.1	1801.1	199	5079	716.5	635.9
eil51.tsp	14	50	4	905	0.0	1.7	3.5	1801.1	75	4601	903.3	821.9

Table 5: Results obtained by (1) BCP and (2) BC for small instances in class C

inst	m	U	Q	z*	gap[1]	gap[2]	time[1]	time[2]	nodes[1]	nodes[2]	root[1]	root[2]
eil76.tsp	3	18	7	404	25.1	26.3	1801.6	1801.2	29	1533	319.1	255.9
eil76.tsp	4	18	5	385	0.0	2.9	429.9	1801.9	45	1211	380.8	280.1
eil76.tsp	5	18	4	439	0.0	2.1	258.8	1802.0	83	1069	432.5	320.5
eil76.tsp	7	18	3	527	0.0	1.3	4.2	1801.1	27	1447	523.3	371.9
eil76.tsp	10	18	2	676	0.0	0.0	0.6	1113.6	4	1176	675.5	499.0
eil76.tsp	14	18	2	745	0.0	0.0	0.7	687.8	14	1123	743.5	610.8
eil76.tsp	3	37	14	623	51.6	50.8	1801.3	1801.1	7	2157	404.1	380.0
eil76.tsp	4	37	11	676	48.9	51.2	1801.3	1801.2	25	1539	451.8	396.9
eil76.tsp	5	37		491	0.0	1.7	25.2	1800.0	3	1213	490.9	418.2
eil76.tsp	7	37	6	626	0.0	4.9	586.8	1801.0	419	1521	620.0	468.4
eil76.tsp	10	37	5	723	0.0	3.7	824.6	1801.0	1059	1553	716.0	565.6
eil76.tsp	14	37	3	969	0.0	0.3	1.5	1801.2	3	1375	968.0	771.7
eil76.tsp	3	56	21	488	0.8	0.0	1800.6	650.9	35	1078	482.0	459.0
eil76.tsp	4	56	16	775	47.3	47.3	1801.0	1801.4	69	2015	522.1	474.3
eil76.tsp	5	56		566	0.0	1.6	599.1	1800.0	100	1235	562.3	498.4
eil76.tsp	7	56	9	926	35.6	42.2	1800.2	1800.0	925	849	676.5	553.2
eil76.tsp	10	56	7	1058	32.3	38.3	1800.4	1801.3	2213	1935	794.3	628.6
eil76.tsp	14	56	5	1273	27.7	32.9	1800.4	1801.1	4805	1767	988.8	845.1
eil76.tsp	3	75	28	878	51.4	51.1	1800.9	1801.1	33	4915	578.0	564.2
eil76.tsp	4	75	21	922	49.2	50.9	1800.1	1801.1	99	2221	614.4	579.5
eil76.tsp	5	75	17	966	45.9	49.3	1801.1	1800.0	201	1311	657.3	594.0
eil76.tsp	7	75	12	1001	31.2	37.7	1800.1	1801.2	497	1849	756.9	657.3
eil76.tsp	10	75	9	1114	23.6	31.5	1800.5	1801.2	2007	1827	893.4	770.8
eil76.tsp	14	75	6	1380	20.7	26.8	1800.2	1801.1	2281	1649	1139.5	947.9
eil101.tsp	3	25	10	504	41.6	38.1	1802.0	1801.2	3	673	355.2	316.9
eil101.tsp	4	25	7	485	20.9	21.3	1801.0	1801.4	13	613	398.4	338.6
eil101.tsp	5	25	6	554	28.5	29.4	1800.5	1801.2	11	571	429.0	361.9
eil101.tsp	7	25	4	654	21.1	26.7	1842.3	1801.4	41	553	538.1	423.4
eil101.tsp	10	25	3	631	0.0	1.9	9.2	1801.7	1	601	630.8	510.8
eil101.tsp	14	25	2	789	0.0	0.0	1.7	1144.2	1	426	789.0	669.5
eil101.tsp	3	50	19	783	57.5	53.2	1802.0	1801.5	4	891	490.8	483.4
eil101.tsp	4	50	14	830	54.6	53.7	1801.1	1801.1	6	865	534.9	497.3
eil101.tsp	5	50	12	803	41.1	42.9	1801.1	1801.5	10	795	563.9	515.8
eil101.tsp	7	50	8	944	41.5	46.8	1802.1	1801.2	9	691	665.7	563.8
eil101.tsp	10	50	6	1054	33.4	40.9	1835.4	1801.3	123	739	786.9	644.6
eil101.tsp	14	50	4	993	0.0	4.4	17.3	1801.2	1	659	993.0	783.5
eil101.tsp	3	75	28	995	66.4	60.0	1801.9	1802.5	3	1127	596.3	602.4
eil101.tsp	4	75	21	1047	66.7	62.3	1800.1	1801.5	3	1185	627.6	616.2
eil101.tsp	5	75	17	1082	61.0	61.3	1801.8	1801.1	3	993	671.3	634.3
eil101.tsp	7	75	12	1157	54.1	57.4	1803.8	1801.6	5	883	750.8	678.2
eil101.tsp	10	75	9	1243	42.5	48.7	1800.6	1801.3	107	889	870.0	747.1
eil101.tsp	14	75	6	1468	34.7	42.5	1803.7	1801.5	369	763	1086.1	883.5
eil101.tsp	3	100	38	1094	60.6	58.8	1804.4	1801.2	2	1271	680.9	667.7
eil101.tsp	4	100	28	1140	59.9	59.7	1800.9	1801.2	3	1871	712.5	685.8
eil101.tsp	5	100	23	1154	55.3	55.9	1826.9	1801.2	11	1143	742.1	702.2
eil101.tsp	7	100	16	1236	50.4	53.0	1800.8	1801.6	93	991	820.2	742.7
eil101.tsp	10	100	12	1330	42.7	47.5	1800.0	1801.4	351	981	928.8	818.6
eil101.tsp	14	100	8	1527	33.8	40.2	1800.0	1801.5	1173	791	1138.3	945.9

Table 6: Results obtained by (1) BCP and (2) BC for large instances in class C

CLASS	BEST BOUND (%)	UNSOLVED			BEST GAP (%)
		BOTH	BCP	BC	
A	100	26	28	54	81
B	100	31	32	55	77
C	99	37	39	52	73

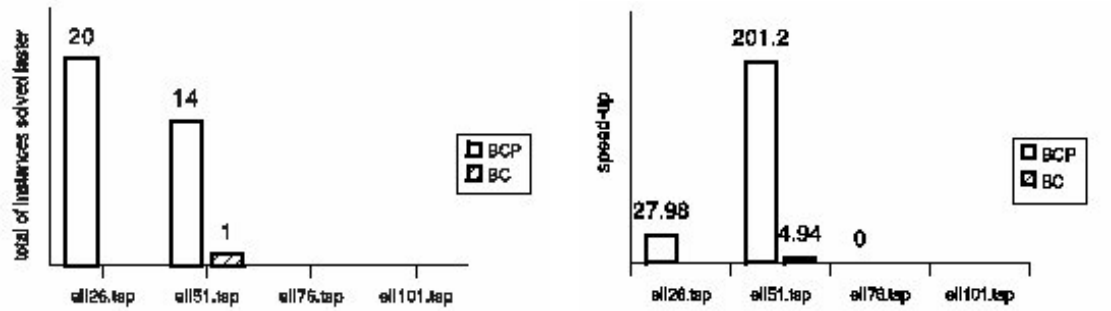
Table 7: Summary of the results of algorithms BCP and BC for classes A, B and C.

	Class A			Class B			Class C		
inst	GAP	TIME	OPT	GAP	TIME	OPT	GAP	TIME	OPT
eil26	*	6.8	0/20	*	6.6	0/20	*	10.5	0/20
eil51	*	5.9	6/22	*	4.7	6/22	1.2	1.2	4/19
eil76	2.0	*	14/14	3.5	*	11/11	3.5	1418.5	7/10
eil101	2.7	*	6/7	3.6	*	6/6	1.5	669.1	2/3

Table 8: Comparative BC × BCP in each instance class.

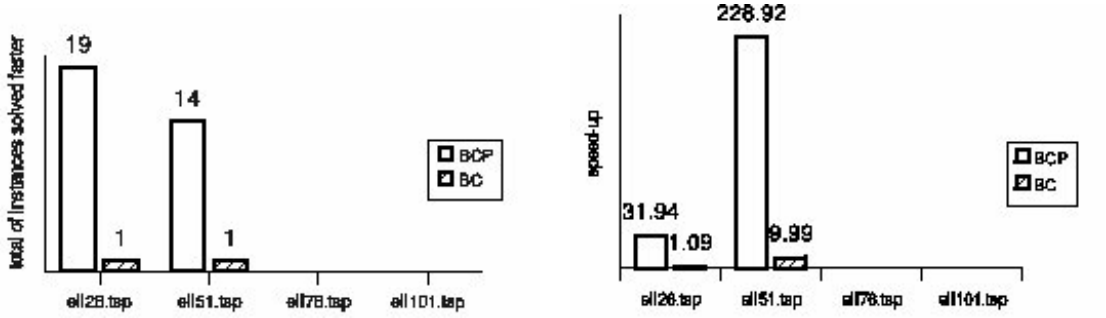
The performances of the algorithms in each instance class is also illustrated in Figures 3, 4 and 5. On the left side in all of these figures, we present the total of instances solved faster by each algorithm. The right side shows the speed-up average given by the processor time of the slower algorithm divided by that of the faster. Besides solving more instances to optimality than BC (as already showed in the previous tables), BCP computed many instances faster than BC as we can see in these figures.

We also noticed that as the number of vehicles m increases, the performance of BC decreases while that of BCP gets better. The difference of behavior becomes more evident when the data displayed in Table 9 are analyzed. There, among all instances of the benchmark for each value of the parameter m corresponding to the fleet size (number of rings), we show the percentage of instances that were solved to optimality by each algorithm. Before doing our analysis, recall that, according to the procedure that generated the instances,



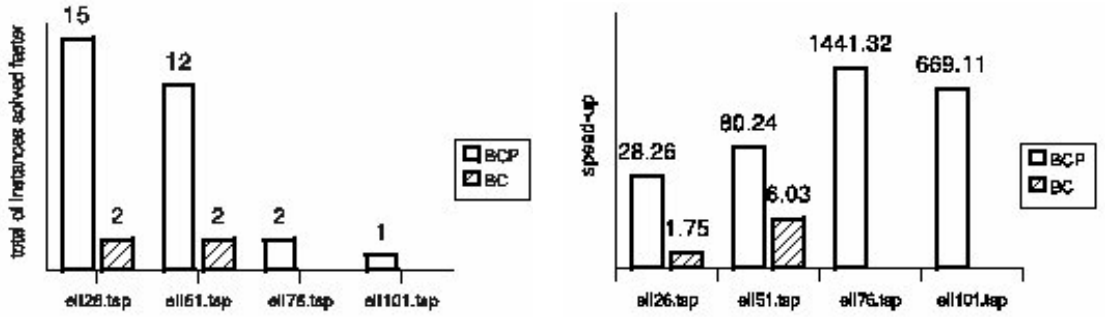
(a) total of instances solved faster (b) speed-up rate

Figure 3: Comparison between BCP and BC for instances in class A.



(a) total of instances solved faster (b) speed-up rate

Figure 4: Comparison between BCP and BC for instances in class B.



(a) total of instances solved faster (b) speed-up rate

Figure 5: Comparison between BCP and BC for instances in class C.

ALGORITHM	$m = 3$	$m = 4$	$m = 5$	$m = 7$	$m = 10$	$m = 14$
BCP	56%	63%	69%	67%	73%	62%
BC	50%	50%	38%	33%	33%	31%

Table 9: Total of instances solved at optimality.

the vehicle capacity (Q) is strictly related to the parameter m. Thus, relative to capacity constraints, these instances can be think of having the same degree of difficulty. In this context, it is reasonable to assume that the size of m is more likely to make an instance harder to solve than the value of Q. In fact, as can be seen in Table 9, both algorithms solved the same number of instances to optimality when $m = 3$. However BCP solved two (six) instances more than BC for $m = 4$ ($m = 5$). Moreover, for $m > 5$, BC solved 14 instances whereas BCP solved 32 instances to optimality.

As said before, we extended our experiments to include instances with costs computed with the weights EUC_2D as in [1] in order to compare our method with the branch-and-cut implemented in that work. In the analysis that follows, the times reported in [1] were multiplied by 0.65 to reflect the difference between the CPU clocks of the machines used in the two experiments. We should notice that one has to be very careful with this sort of comparison since variations in performance are due not only to hardware aspects. In this case, the implementations used different linear programming solvers, different programming language compilers and were executed under distinct operating systems. The implementation of BC in [1] also makes use of a more aggressive branching rule and of an upper bound computed by a primal heuristic in a preprocessing phase. None of these features were implemented in our algorithm. Nevertheless, for completeness, we decide to report also on this additional test.

The results for classes A and B are summarized in Table 10. Column FAST shows in how many instances the algorithm was faster and column SPEED-UP represents the average in speed-up for each algorithm where it ran faster. The total of instances solved to optimality is provided in column OPT. The number of times each algorithm got a better lower bound and the final gap average are given in columns LB and GAP, respectively.

	Class A					Class B				
ALGORITHM	FAST	SPEED-UP	OPT	LB	GAP	FAST	SPEED-UP	OPT	LB	GAP
BC	10	16.8	35	25	0.5	8	20.4	28	28	1.3
BCP	18	16.0	28	20	0.7	16	43.6	29	17	2.4

Table 10: Comparative between results reported in [1] and BCP algorithm.

One can see that BCP solved more instances faster than BC and got a better speed-up. However, the total of instances solved to optimality by BC was bigger than BCP. The results also suggest that BCP is slightly more suited to handle instances in class B while the opposite seems to be true for class A. Although this last experiment was not very conclusive, from the material presented in this section, we can say that BCP provides a very effective way to solve the CmRSP exactly and is at least as good as BC.

4 Conclusions

In this paper we analyzed the performance of a branch-and-cut-and-price algorithm (BCP) for the CmRSP. We extended the branch-and-price algorithm previously proposed in [2] by adding a subset of cuts described in [1]. The BCP outperformed our implementation of the branch-and-cut algorithm presented in [1] in several aspects. It solved many instances more and provided a higher speed-up. In particular, our tests with instances having large fleet sizes indicate that, in this case, the branch-and-cut-and-price approach is far more adequate than the branch-and-cut one. Therefore, though no definitive conclusions can be drawn by comparing the results of our BCP algorithm with those reported in [1], our analysis suggests that branch-and-cut-and-price is a competitive and robust approach to tackle the CmRSP.

With respect to possible improvements to the BCP algorithm, one could devise the implementation of a strong branching strategy and of primal heuristics either to be called at each node of the enumeration tree or as a preprocessing phase. In [1], both these issues are reported to be vital to enhance the performance of the branch-and-cut algorithm.

References

[1] R. Baldacci, M. Dell'Amico, J. Salazar, The capacitated m-ring star problem, Operations Research 55 (2007) 1147–1162.

[2] E. A. Hoshino, C. C. de Souza, Column generation algorithms for the capacitated m-ring-star problem, in: 14th Annual International Computing and Combinatorics Conference (COCOON 2008), Vol. 5092 of LNCS, Springer, 2008, pp. 631–641.

[3] R. Fukasawa, H. Longo, J. Lysgaard, M. P. de Aragão, M. Reis, E. Uchoa, R. Werneck, Robust branch-and-cut-and-price for the capacitated vehicle routing problem, Mathematical Programming 106 (3) (2006) 491–511.

[4] L. A. Wolsey, Integer Programming, Wiley-Interscience, 1998.

[5] M. Dell'Amico, F. Maffioli, P. Varbrand, On prize-collecting tours and the asymmetric travelling salesman problem, International Transactions in Operational Research 2 (3) (1995) 297–308.

[6] S. Irnich, D. Villeneuve, The shortest path problem with resource constraints and k-cycle elimination for $k \geq 3$, INFORMS Journal on Computing 18 (3) (2006) 391–406.

[7] M. G. C. Resende, C. C. Ribeiro, Greedy radomized adaptive search procedures, in: F. Glover, G. Kochenberger (Eds.), Handbook of Metaheuristics, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002, pp. 219–249.

[8] L. Lasdon, Optimization Theory for Large Systems, MacMillan, 1970.

[9] J. Lysgaard, A. Letchford, R. Eglese, A new branch-and-cut algorithm for the capacitated vehicle routing problem, Mathematical Programming 100 (2) (2004) 423–445.

[10] Dash Optimization, XPRESS-Optimizer Manual (2007).

[11] G. Reinelt, A traveling salesman problem library, ORSA Journal on Computing 3 (1991) 376–384.

[12] TSPLIB: a library of instances for the TSP and other related problems, `www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/`.

[13] E. A. Hoshino, C. C. de Souza, Instances for the capacitated m-ring-star problem, `www.ic.unicamp.br/~cid/Problem-instances/CmRSP/`.