Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 346 (2019) 3–14

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

A Branch-and-Price Algorithm for the Ring-Tree Facility Location Problem

Fabio H. N. Abe[a,1], Edna A. Hoshino[b,2,5], Alessandro Hill[c,3] and Roberto Baldacci[d,4]

[a] *Instituto Federal de Mato Grosso do Sul*
Dourados, Brazil

[b] *Faculdade de Computação*
Universidade Federal de Mato Grosso do Sul
Campo Grande, Brazil

[c] *Department of Industrial and Manufacturing Engineering*
California Polytechnic State University
San Luis Obispo, USA

[d] *Department of Electrical, Electronic, and Information Engineering "Guglielmo Marconi"*
University of Bologna
Bologna, Italy

Abstract

The ring-tree facility location problem is a generalization of the capacitated ring-tree problem in which additional cost and capacity related to facilities are considered. Applications of this problem arise in the strategic design of bi-level telecommunication networks. We investigate an extended integer programming formulation for the problem and different approaches to deal with the NP-hardness of the pricing problem that appears in a branch-and-price algorithm to solve it. Computational experiments show how heuristics and relaxations improved the performance of a branch-and-price algorithm.

Keywords: capacitated ring-tree problem, network design, integer programming, column generation, q-routes.

1 Introduction

The ring-tree facility location problem (RTFLP) is related to the capacitated ring-tree problem (CRTP), that was first described in [13]. The goal of both problems consists in designing a centralized two-level network of minimal cost in order to

[1] Email: fabio.abe@ifms.edu.br
[2] Email: eah@facom.ufms.br
[3] Email: ahill29@calpoly.edu
[4] Email: r.baldacci@unibo.it
[5] Partially supported by Fundect, Brazil.

https://doi.org/10.1016/j.entcs.2019.08.002

connect customers to a central depot. Customers are connected to the central depot by **rings** (or cycles), on the inner level, and by **trees** that intercept the depot or one node of a ring, on the outer level. The structure on the inner level provides redundancy and, usually, is more expensive. Customers are also classified into two types, depending on the required level of connection. Customers of Type 1 can be connected by rings or trees whereas customers of Type 2 must be connected by rings. Additionally, Steiner nodes can be used as optional intermediate connections to reduce the overall network cost and they can be used in rings or trees. A **ring-tree** is a tree intercepting the depot or a ring passing throught the depot plus all trees intercepting its ring nodes. Motivated by real applications, the RTFLP and the CRTP consider a capacity for the total number of customers in each ring-tree and also for the number of ring-trees used to connect all customers to the depot. In [1], the RTFLP was introduced to deal with a broader class of applications. It generalizes CRTP by considering two different costs to each connection (ring and tree connection costs), an additional facility installation cost that must be payed when a tree is connected to a ring node, and also a tree capacity that defines an upper bound to the number of customers that can be served by each facility.

There are a few works in the literature directly related to the CRTP and the RTFLP. A multi-exchange local search heuristic and a matheuristic based on generalized local branching for CRTP were described in [10,14]. Exact approaches for CRTP were proposed in [13,16]. In [12], different bi-objective functions for CRTP are discussed to minimize the risk of service loss due to single-edge failure. Profit-oriented versions of the CRTP were recently introduced in [11], in which models, heuristic, valid inequalities, and exact algorithms were proposed. The RTFLP is also related to the capacitated ring-star problem (CRSP). Basically, a ring-star refers to a ring-tree, whose trees have depth one. In the CRSP, there are no required levels for customers, tree capacity and facility installation costs. Notice that a CRSP solution could not be feasible for a RTFLP and vice versa. Thus, the RTFLP and the CRSP are similar but each one does not generalize the other. Exact algorithms for the CRSP were proposed in [3,15]. The CRSP has many applications and it is related to school bus routing problems, whose review can be found in [18]. The multiple vehicle traveling purchaser problem (MV-TPP) is a school bus routing problem very close to the CRSP. In the MV-TPP, bus stops are selected simultaneously with the generation of routes. Branch-and-cut and branch-and-cut-and-price algorithms were proposed for the MV-TPP in [19] and [20], respectively. A review of other integrated two-level network design problems is given in [17]. Motivated by applications of hub location problems in [17], the first level is called backbone network and the second level tributary network. Considering 2-edge connected networks and rings as structures in both levels, a branch-and-cut algorithm is proposed in [21] to design a two-level survivable network. In [1], two integer linear programming (ILP) formulations and preliminary results of a branch-and-price (B&P) algorithm for the RTFLP are presented. These preliminary results show that the set covering formulation provides very tight lower bounds but at very expansive processing time.

In this work, we investigate an extended ILP formulation for the RTFLP and dif-

ferent approaches to deal with the NP-hardness of the pricing problem that appears in a branch-and-price algorithm to solve it. To accelerate the exact approach for the pricing problem, we consider a relaxation that allows vertex repetition inside the ring-trees. This relaxation is similar to the idea of ng-routes and q-arbs which were successfully used in variants of vehicle routing problems [4,5,22]. We also propose a primal heuristic to populate the initial basis and a heuristic for the pricing problem. The remainder of this paper is organized as follows. In Section 2 we provide a formal description of the problem, additional definitions and notation. Section 3 is devoted to our algorithmic approaches, including a primal heuristic, heuristics and relaxations for the pricing problem. These methods are analyzed on instances from the literature and the computational results are reported in Section 4. Finally, we discuss some conclusions and address future work in Section 5.

2 Problem Definition

Let $G = (V, E)$ be a simple undirected graph and $V = U_1 \cup U_2 \cup W \cup \{d\}$, where d denotes the central depot, U_2 is the set of customers of Type 2, U_1 the set of customers of Type 1 and W the set of Steiner nodes. We denote the set of all customers $U_1 \cup U_2$ by U and the set of vertices induced by an edge set $X \subseteq E$ by $V[X]$.

We define a **ring-tree** as a pair (R, T), where $R \subseteq E$ induces a cycle in G and $T \subseteq E$ induces a forest in G, such that either $R = \emptyset$ and T is a tree in which d is a leaf, or; $R \neq \emptyset$ is a cycle including d and each maximally connected component induced by T has exactly one vertex $v \neq d$ in $V[R]$. In the latter case, we assume that certain equipament must be installed in each vertex v to provide service to all customers of the corresponding connected component. In this case, we say that the vertex v has a **facility** associated to it and those customers in the connected component are **served** by this facility. Finally, we say that a customer u is **covered** by a ring-tree (R, T) if $u \in V[R] \cup V[T]$.

Consider a ring edge cost function $c^{\mathtt{r}} : E \mapsto \mathbb{N}$, a tree edge cost function $c^{\mathtt{t}} : E \mapsto \mathbb{N}$ and a facility installation cost function $c^{\mathtt{f}} : V \mapsto \mathbb{N}$. Moreover, let m, Q and Q_t be integers that represent the maximal number of ring-trees, the ring-tree capacity, and tree capacity, respectively. Then a ring-tree $p = (R, T)$ is (Q, Q_t)-**capacitated** if $|V[R \cup T] \cap U| \leq Q$ and the number of customers served by each facility is less than or equal to Q_t. The **ring-tree cost** of p is given by $c_p = \sum_{e \in R} c^{\mathtt{r}}_e + \sum_{e \in T} c^{\mathtt{t}}_e + \sum_{v \in V[R] \cap V[T]} c^{\mathtt{f}}_v$.

The **ring-tree facility location problem** consists in finding at most m (Q, Q_t)-capacitated ring-trees $p_1, \ldots, p_k$, such that (i) each customer of Type 1 is covered by one ring-tree; (ii) each customer of Type 2 belongs to the cycle of one ring-tree; and (iii) the overall network cost $\sum_{i=1}^{k} c_{p_i}$ is minimized.

RTFLP is NP-hard since it generalizes the traveling salesman problem when $m = 1$, $Q = |V|$, and $U_2 = V \setminus \{d\}$. It is also related to other well-known optimization problems such as vehicle routing problems, Steiner tree problems, and ring-star

problems. For more relationships regarding ring-tree topology, we refer to [13].

3 Algorithmic Approaches

Two ILP formulations for the RTFLP were proposed in [1]. The first one is a compact formulation based on multi-commodity flows (MCF) and the second one considers (Q, Q_t)-capacitated ring-trees as columns in a set partitioning model (SPF). The lower bounds provided by the linear relaxation (LR) of (SPF) were 51% tighter than those obtained by the compact formulation [1]. Since (SPF) has an exponential number of variables, usually the column generation (CG) approach is used to solve its LR. In this approach, two subproblems must be solved: the restricted master problem (RMP) and the pricing problem. The RMP is the linear relaxation restricted to a small subset of the columns and the pricing problem consists in identifying a column, with negative reduced cost to be added to the RMP, which is re-optimized. This procedure repeats until no column with negative reduced cost can be found. The pricing problem that arises when solving the LR of (SPF) by CG consists of finding a ring-tree with minimum reduced cost. In [1], (MCF) and (SPF) consider an auxiliary oriented graph $G' = (V, A)$ with $A = \{(i,j),(j,i) : ij \in E\}$ to model a directed ring-tree. Notice that a directed ring-tree uniquely induces an undirected counterpart. Hereafter, G' will be used as G. Thus, edges and trees can be also mentioned as arcs and arborescences. For the sake of notation, an arc (i,j) will be denoted by ij.

Let $\mathcal{P}$ be the set of all (Q, Q_t)-capacitated ring-trees for an instance of the RTFLP. A ring-tree $p = (R, T)$ can be represented by two characteristic vectors r and t of dimension $(|V| + |A|) \times 1$. The components of r are such that r_i^p equals to one if and only if vertex i belongs to $V[R]$ and r_{ij} is one if and only if arc ij is in R. Similarly, the components of t are such that t_i^p equals to one if and only if vertex i belongs to $V[T] \setminus V[R]$ and t_{ij} is one if and only if arc ij is in T. We associate a variable λ_p to each ring-tree $p \in \mathcal{P}$ to decide if p is part of the solution. We also use binary variables y_i and x_e for each vertex $i \in W$ and each arc e that assume value 1 if and only if they belong to the solution. A set partition formulation for the RTFLP is described next.

$$
\begin{aligned}
(SPF) \quad \min \ & \sum_{p \in \mathcal{P}} c_p \lambda_p & & \\
\textit{subject to} \ & \sum_{p \in \mathcal{P}} \lambda_p \leq m & & (1)\\
& \sum_{p \in \mathcal{P}} (a_i^p) \lambda_p = 1 & \forall i \in U & (2)\\
& \sum_{p \in \mathcal{P}} (a_i^p) \lambda_p = y_i & \forall i \in W & (3)\\
& \sum_{p \in \mathcal{P}} (r_e^p + t_e^p) \lambda_p = x_e & \forall e \in A & (4)\\
& y \in \{0,1\}^{|W|}, x \in \{0,1\}^{|A|}, 0 \leq \lambda_p \leq 1 & \forall p \in P & (5)
\end{aligned}
$$

where $a_i^p = r_i^p$, if $i \in U_2$, otherwise, $a_i^p = r_i^p + t_i^p$.

Constraint (1) limits the number of ring-trees while Inequalities (2) ensure that each customer of Type 1 is contained in exactly one ring-tree and each customer

of Type 2 belongs to exactly one ring. Constraints (3), (4), and (5) forbid Steiner nodes and arcs to appear in two or more ring-trees. Let π, α, γ and β be the dual variables associated with constraints (1), (2), (3), and (4), respectively. The reduced cost of a ring arc is $\bar{c}^{\mathtt{r}}_{ij} = c^{\mathtt{r}}_{ij} - \beta_{ij} - k_j$ and the reduced cost of a tree arc is $\bar{c}^{\mathtt{t}}_{ij} = c^{\mathtt{t}}_{ij} - \beta_{ij} - k_j$, where k_j is 0, α_j, or γ_j depending on whether j is d, or j in U, or W, respectively. Thus, the **reduced cost of a ring-tree** $p = (R, T)$ is given by $\bar{c}_p = \sum_{ij \in A} \bar{c}^{\mathtt{r}}_{ij} r^p_{ij} + \sum_{ij \in A} \bar{c}^{\mathtt{t}}_{ij} t^p_{ij} + \sum_{v \in V} c^{\mathtt{f}}_v \theta_v - \pi$, where θ_v is equal to 1 if $v \in V[R] \cap V[T]$ and 0, otherwise.

The **pricing problem for (SPF)** consists in finding a ring-tree $p \in \mathcal{P}$ that minimizes $\bar{c}_p$, which is NP-hard since it generalizes the profitable tour problem [8] when $U_1 = W = \emptyset$ and $Q = |V|$. We discuss next our approaches to cope with the hardness of this subproblem.

3.1 The relaxed pricing problem

Instead of finding the ring-tree with minimal reduced cost, we consider a relaxation for the pricing problem that allows some vertex repetitions. Our idea consists of combining two successful relaxations: ng-route and qarb relaxations, proposed in [5] and [22] for the capacitated vehicle routing problem and capacitated minimum spanning tree problem, respectively. This approach was also successfully applied to the CRSP [4].

A **ng-route** consists of non-necessarily elementary routes (or rings) that consider a set of prohibited vertex. This set is dynamically built and is based on sets called **ngsets** that are associated with each vertex. Basically, the ngset of vertex i, denoted by $NG(i)$, contains i and a limited size subset of selected vertices, for example, Δ closest neighbor of vertex i. As higher as the parameters Δ, more elementary is the route. Usually, routes are built as paths starting from the depot d, ending in a vertex, say i, and including an arc (i, d). Thus, all routes can be obtained by dynamic programming (DP) just extending paths over all vertices that are not prohibited. Consider a path $P = (v_0 = d, v_1, \ldots, v_k)$ starting from the depot and ending at vertex v_k. The set of **prohibited vertices** to extend the path P is denoted by $\pi(P)$ and is given by $\pi(P) = \{v_r : v_r \in \bigcap_{i=r+1}^{k} NG(v_i), 1 \leq r \leq k-1\} \cup \{v_k\}$.

Now, a **qarb** rooted at vertex i, $i \neq d$, is the vertex i alone or the vertex i connected to other qarbs rooted at distinct vertices $v_1, \ldots, v_k$ by arcs (i, v_j), $j = 1, \ldots, k$. We combine ng-route and qarb to obtain a relaxation of the pricing problem. To explain this idea consider a few more definitions. A **relaxed qarb** is a qarb rooted at vertex i plus the arc (d, i). An **ng-path** is a non-necessarily elementary path $P = (v_0 = d, v_1, \ldots, v_k)$ starting from the depot in which $v_k \notin \pi(P')$ where $P' = (v_0, v_1, \ldots, v_{k-1})$ is also an ng-path. In other words, each vertex v_i in P is not a prohibited vertex to extend the path $(v_0, \ldots, v_{i-1})$. An **ng-pathtree** is an ng-path $P = (v_0 = d, v_1, \ldots, v_k)$, $k \geq 2$, plus qarbs rooted at vertices in P, except the depot. An **ng-ringtree** is an ng-pathtree ending at a vertex i plus the arc (i, d). We call both, a relaxed qarb and an ng-ringtree, as a **relaxed ring-tree**. Noticed that a ring-tree is also a relaxed ring-tree. The **relaxed pricing problem** consists of finding the relaxed ring-tree of minimal reduced cost.

The algorithm to solve the relaxed pricing problem consists of two DP procedures. Consider an order $v_1 < v_2 < \ldots < v_l$ to the vertex set $U_1 \cup W$ in which $l = |U_1 \cup W|$.

Let $g(i, j, q, s)$ be the cost of a minimum cost qarb rooted at vertex v_i in which the number of customers is equal to q, the number of Steiner nodes is exactly s, and each child v_k of the root node is such that $k \leq j$. Functions $g(i, j, q, s)$, for each $i \in V \setminus \{d\}$, $j \in U_1 \cup W$, $q \in \{0, \ldots, Q_t\}$, and $s \in \{0, \ldots, |W|\}$ can be computed in pseudo-polynomial time using the DP functions similar to the ones described in [22] as follows. $g(i, j, q, s) = \min\{g(i, j-1, q, s), \bar{c}^{\mathtt{t}}_{ij} + g(j, l, q-a, s-(1-a)), \min\{g(i, j-1, q-q', s-s') + \bar{c}^{\mathtt{t}}_{ij} + g(j, l, q', s') : a' \leq q' \leq q-a, 1-a' \leq s' \leq s-(1-a)\}$, where the constant $a = 1$, if $i \in U_1$, otherwise $a = 0$ and $a' = 1$, if $j \in U_1$, otherwise $a' = 0$. We initialize $g(i, 0, 0, 0) = \infty$ and $g(i, 0, a, 1-a) = 0$, for all vertex $i \in U_1 \cup W$ and compute $g(i, j, q, s)$ in increasing order of j, q and s. The cost of a minimum cost relaxed qarb can be computed as $\min_{i \in U_1 \cup W, a \leq q \leq Q_t, 1-a \leq s \leq |W|} \bar{c}^{\mathtt{t}}_{di} + g(i, l, q, s)$.

Let $f(\pi, i, q, s)$ be the cost of a ng-pathtree of minimal cost whose ng-path P ends at the vertex i and such that (i) $\pi = \pi(P)$; (ii) the number of customers in the ng-ringtree is equal to q, and (iii) the number of Steiner nodes is exactly s. The DP recursion for computing function $f(\pi, i, q, s)$, for each vertex $i \in V \setminus \{d\}$, $q \in \{0, \ldots, Q\}$, and $s \in \{0, \ldots, |W|\}$ is as follows. $f(\pi, i, q, s) = \min\{f(\pi', j, q', s') + \bar{c}^{\mathtt{r}}_{ji} + g(i, |U_1 \cup W|, q-q', s-s') + b : i \notin \pi', 0 \leq q' \leq q-a, 0 \leq s' \leq s-(1-a)\}$. where $a = 1$ if vertex $i \in U$, otherwise $a = 0$ and $b = c^{\mathtt{f}}_i$ if $q - q' + s - s' > 1$, otherwise $b = 0$. We initialize $f(\{0\}, 0, 0, 0) = 0$ and $f(\{0\}, 0, q, 0) = \infty$, $\forall 1 \leq q \leq Q$. Thus, the cost of a minimum cost relaxed ng-ringtree can be computed as $\min_{i \in V \setminus \{d\}, 0 \leq q \leq Q, 0 \leq s \leq |W|} f(\pi, i, q, s) + \bar{c}^{\mathtt{r}}_{id}$.

Consider two ng-ringtrees R_i, $i \in \{1, 2\}$, whose corresponding ng-path P_i ends at the same vertex. Let q_i and s_i the number of customers and Steiner nodes in R_i, respectively. Ng-ringtree R_1 **dominates** R_2 (and R_2 is dominated by R_1) if its cost is less than R_2's cost, $q_1 \leq q_2$, $s_1 \leq s_2$, and $\pi(P_1) \subseteq \pi(P_2)$. Since dominated ng-ringtrees are unnecessary to construct a relaxed ring-tree of minimal cost, we discard all of them.

3.2 Partial pricing

Partial pricing consists of using a heuristic to identify an approximate solution for the pricing problem quickly. Wherever the heuristic fails to find a column with negative reduced cost, an exact algorithm must be used.

Our heuristic consists of two layers. At the outer layer, a greedy randomized adaptive search procedure (GRASP) [9] is used to build a ring R passing through the depot and some customers. At the inner layer, arborescences connecting customers in $U_1 \cup W - V[R]$ are constructed, and further attached to the ring R to obtain a ring-tree. In each iteration of the construction phase at the outer layer, we increase the ring $R = \{(v_0, v_1), (v_1, v_2), \ldots, (v_k, v_{k+1} = v_0)\}$ by randomly selecting an element of a list of restricted candidates (RCL) and including one vertex in $V[R]$. Each element in RCL consists of a pair (u, i) where u is a vertex in $V - V[R]$ and $i \in [0, k]$ is a position in R. A greedy adaptive function f provides the contribution of

each candidate (u, i) for the reduced cost of R and it is given by $f(v, i) = \bar{c}^{\mathrm{r}}_{v_i u} + \bar{c}^{\mathrm{r}}_{u v_{i+1}} - \bar{c}^{\mathrm{r}}_{v_i v_{i+1}}$. This function is used to construct the RCL with the most promising candidates based on a greedy parameter α.

After each change in R at the outer layer, the procedure at the inner layer is called to construct arborescences and join those with negative reduced cost to the current R. Notice that each arborescence T has a reduced cost that we will denote by $\bar{c}(T)$ and it is given by the sum of reduced cost of their arcs and an additional facility installation cost, if needed. To construct the set of arborescence, we used Edmond's algorithm [7] by successively selecting arcs (i, j), in non-decreasing order of reduced cost, with additional condition $|V[T_i \cup T_j] \cap U_1| \leq \min\{Q_t, Q - |V[R] \cap U|\}$, where T_i (T_j) is the arborescence that contains vertex i (j). We consider three ways to join an arborescence rooted at node k to R: (1) directly connecting the root k at depot node, (2) connecting it to a ring node of $V[R]$; or (3) including k as a ring node in R.

Considering all arborescences built, we construct another RCL to select arborescences to be joint to R. Each element of this RCL consists of a triple (T, i, l), where T is an arborescence, i refers to the position in R where T should be attached and l defines one of ways, $1, 2$, or 3, of join T to R. We compute the contribution of each candidate (T, i, l) for the reduced cost of the ring-tree which is given by $\bar{c}(T) + g$, where $g = \bar{c}^{\mathrm{t}}_{dk}$, $g = \bar{c}^{\mathrm{t}}_{v_i k} + c^{\mathrm{f}}_{v_i}$, if no facility was installed in v_i, or $g = \bar{c}^{\mathrm{r}}_{v_i k} + \bar{c}^{\mathrm{r}}_{k v_{i+1}} - \bar{c}^{\mathrm{r}}_{v_i v_{i+1}} + c^{\mathrm{f}}_{k}$, depending on $l = 1, 2$, or 3. To reduce the number of candidates, we only consider (T, i, l) that minimizes $\bar{c}(T) + g$ and whose contribution is non-negative. The inner layer interactively and randomly selects a triple in RCL and finishes when no more arborescence can be joint to R. On the other hand, the outer layer finishes when the number of customers in $V[R]$ reaches the limit Q. At the local search phase, we apply the well-known *2opt* [6] to improve the reduced cost of the ring-tree. All ring-trees with negative reduced cost found are included as columns into the RMP.

3.3 A primal heuristic

We proposed a primal heuristic to construct a set of solutions for the RTFLP. These solutions are used to populate the initial basis of the RMP and the best one is used to define the first primal bound.

Our heuristic is based on GRASP without the local search phase. Firstly, we construct a set $\mathcal{S} = \{(R_1, T_1), (R_2, T_2), \ldots, (R_s, T_s)\}$ of empty ring-trees to cover all customers. Then, we iteratively include each customer U_2 in the ring of some ring-tree in $\mathcal{S}$. After that, each customer U_1 is included in some ring-tree, and, finally, we try to include Steiner node of W in some ring-tree, only if the overall cost is reduced. To select a vertex to be included in a ring-tree, we construct a RCL in the same fashion used in the heuristic of pricing. For the first step, each element in the RCL consists of a triple (u, i, l) where u is a vertex in $U_2 - V[R_l]$, l is the index of one ring-tree (R_l, T_l) of $\mathcal{S}$, and $i \in [0, k]$ is a position in R_l. In this case, the function f that provides the contribution of (u, i, l) for the cost of S is given by $f(v, i, l) = c^{\mathrm{r}}_{v_i u} + c^{\mathrm{r}}_{u v_{i+1}} - c^{\mathrm{r}}_{v_i v_{i+1}}$. We consider four ways to include a vertex

k of U_1 into a ring-tree: (1) inserting into a cycle by removing ring arc (i, j) and adding (i, k) and (k, j) , (2) appending to a ring node j, (3) inserting into a tree by removing tree arc (i, j) and adding (i, k) and (k, j), or (4) appending to a vertex j of an existing tree. In the last cases, the tree capacity has to be checked and the facility installation cost has to be considered in the second case.

4 Computational Experiments

All experiments were carried out on a desktop computer with an Intel i7-4790 3.6GHz processor, 32GB RAM, under Linux. The algorithms were implemented in C and compiled with gcc 4.9.2. We used the SCIP (v3.1.0) framework [2] and Cplex (v12.6.1.0) as the linear programming solver. [6] For the primal heuristic described in Section 3.3, we fixed $\alpha = 0.6$. In all experiments, we fixed the time limit to 1800 seconds and considered the following configurations: best node selection in the branch-and-bound (B&B) procedures, branching on variables, and all preprocessing, heuristic, cutting, and multithread of SCIP and Cplex were disabled.

We derived four instance classes [7] A, B, C, and D from the TSPlib-based CRTP instances proposed in [13]. All CRTP instances (called Q-1, Q-2, ..., Q-45) and the type 1 customer rates in $\{0, 0.25, 0.5, 0.75, 1\}$ were considered. Each class differs in tree edge costs, facility installation costs, and tree capacities. Let w_e be the cost for an edge $e \in E$ in the CRTP instance. For each edge e, we set $c_e^{\mathtt{r}} = w_e$ and $c_e^{\mathtt{t}} = \lfloor w_e/2 \rfloor$. In instance classes A and B $c^{\mathtt{f}} = 0$, while in classes C and D, $c^{\mathtt{f}} = \lceil (\min_{e \in E} w_e + \max_{e \in E} w_e)/2 \rceil$. $Q_t = 1$ in classes A and C while, in other classes, $Q_t = 2 \lfloor Q/3 \rfloor$. The capacities m and Q, and the sets U_1, U_2, and W are the same as in the CRTP. We consider small instances with 26 vertices and large instances with 51, 76, and 101 vertices.

We analyzed the performance of different B&P implementations. Consider (SPFr) a model similar to (SPF) whose columns are relaxed ring-trees. `SPF-`, `SPF+`, and `SPF*` are B&P implementations to solve (SPF) using a B&B to solve the exact pricing whilst `SPFr+` and `SPFr*` solve (SPFr) using the DP described in Section 3.1. We also consider implementations (`MCF-` and `MCF+`) of a B&B for the (MCF). The signal "`-`" is used to indicate that no heuristics are enabled while "`+`" denotes that the primal heuristic described in Section 3.3 is used, and "`*`" indicates that the heuristic of pricing described in Section 3.2 is used in addition to the primal heuristic.

First, we analyzed the quality of primal solutions found by the primal heuristic in small instances. In Table 1, column `BestUB` presents the percentage of heuristic solutions that were equal to the best upper bound found by B&P and B&B implementations. Column `Optimal` shows the percentage of these solutions that were also optimal and column `Gap` provides the relative objective function gap of the primal solutions compared to the best known solution. We also analyzed which procedure generated the columns in the best solution found by `SPF*` code. Columns `Primal`,

[6] The authors thank IBM and SCIP for the academic licenses.

[7] The set of test instances is available upon request from the corresponding author.

`ExactPric`, and `HeurPric` show the percentage of columns in the best solution that were generated by primal heuristic, exact pricing procedure, and pricing heuristic, respectively. As shown in Table 1, the primal heuristic provided the best upper bound in many instances and also was responsible for 86% of columns in the best solution.

Table 1
Results for Primal Heuristics

Class	`BestUB` (%)	`Optimal` (%)	`Gap`(%)	Primal(%)	ExactPric(%)	HeurPric(%)
A	42.2	15.8	4.5	89.7	5.2	5.2
B	31.1	35.7	8.7	82.0	8.4	9.6
C	48.9	22.7	3.8	93.3	0.0	6.7
D	44.4	50.0	11.0	82.6	7.9	9.6
Total	41.7	31.1	7.0	86.9	5.4	7.8

Next, we analyzed the performance of all implementations. Table 2 summarizes the results for each class of instances for $|V| = 26$ and $|V| = 51$. Line `optimal` gives the percentage of instances solved to optimality and `time` shows the average time spent to reach it. Line `solved` shows the percentage of instances which were not solved to optimality, but whose primal and dual bounds could be calculated, and `gap` presents the final duality gap, on average, restricted to these instances. Finally, line `root lb` shows the average duality gap reached at the root node, defined as $(ub - lb0)/ub$, where ub refers to the best upper bound found and $lb0$ to the root lower bound (LB). The symbol $*$ is used to indicate that the time limit was reached.

		$\|V\| = 26$							$\|V\| = 51$		
Class		SPF-	SPF+	SPF*	MCF-	MCF+	SPFr+	SPFr*	MCF+	SPFr+	SPFr*
A	optimal (%)	4.4	8.9	6.7	0	0	13.3	11.1	0	0	0
	time (sec.)	859.8	550.6	476.1	*	*	61.3	87.7	*	*	*
	gap (%)	1.1	6.1	5.7	51.4	47.8	10.9	10.6	47.7	15.9	15.3
	solved (%)	2.2	26.7	42.2	71.1	100	86.7	88.9	78.3	93.3	71.7
	root lb (%)	3.5	3.5	3.5	33.7	33.7	7.3	7.3	29.3	13.4	13.4
B	optimal (%)	11.1	17.8	17.8	0	2.2	33.3	37.8	0	5	5
	time (sec.)	1029.6	366.5	389.9	*	1167.2	2.8	25.6	*	123.1	431.7
	gap (%)	*	6	3.2	50.1	37.8	10.9	11.3	39.8	15.3	14.3
	solved (%)	0	20	22.2	97.8	97.8	66.7	62.2	98.3	85	70
	root lb (%)	1.8	1.8	1.8	28.3	28.3	5.2	5.2	25.7	12.5	12.5
C	optimal (%)	4.4	8.9	11.1	0	0	22.2	24.4	0	5	5
	time (sec.)	855.8	903.7	825.9	*	*	13.4	215.9	*	81.5	302.1
	gap (%)	2.5	4.1	5.8	108.9	63.7	7.8	7.9	64.4	9.4	10.3
	solved (%)	6.7	28.9	42.2	60	100	77.8	75.6	73.3	95	78.3
	root lb (%)	3.4	3.4	3.4	38.7	38.7	5.8	5.8	34.1	8.5	8.5
D	optimal (%)	15.6	15.6	24.4	2.2	4.4	44.4	35.6	0	3.3	1.7
	time (sec.)	1200.9	559.3	719.8	1721.1	1428.4	94	133.9	*	373	324.5
	gap (%)	1.4	2.8	3.2	82.8	59.5	11.7	10.6	62	15.4	15.3
	solved (%)	2.2	15.6	20	82.2	95.6	55.6	64.4	93.3	93.3	83.3
	root lb (%)	1.4	1.4	1.4	33.8	33.8	5.3	5.3	33.1	12.6	12.6

Table 2
B&P and B&B performance for instances with $|V| = 26$ and $|V| = 51$

As shown in Table 2, `SPFr+` and `SPFr*` presented better performance than other implementations. For example, on small instances, `SPFr+` solved 28% of instances at optimality and reached a final duality gap of around 10%, on average. Moreover, the processing time spent by both codes was smaller than the others. With respect

to the root LB, the usage of the relaxed pricing problem decreased the initial LB quality only by 3% whilst (MCF) provided LB around 32% worse. Based on these results, we decided to analyze the performance to solve large instances ($|V| = 51$) only for three implementations (`MCF+`, `SPFr+`, and `SPFr*`). On large instances, `MCF+` could not solve any instance at optimality while `SPFr+` solved 3.3%. Both codes could compute a final duality gap for almost of instances, however the final duality gap reached by `MCF+` was around 45%, on average, while `SPFr+` reached around 14%.

We made a pairwise comparison to analyze the results only on instances solved by both codes. We compare `SPF+` × `SPF*` (Table 3) to analyze the B&P performance when the partial pricing is used. Table 4 reports the comparison between `SPF+` and `SPFr+` to analyze the usage of the relaxed pricing problem. Finally, Table 5 compares B&B and B&P performances. In these tables, columns `solved` and `optimality` show values in the form XX/YY(ZZ) where XX indicates the amount of instances solved by the first code (or solved to optimality, in the second case) whereas YY refers the same for second code, and ZZ gives the amount of instances solved by both codes (or equivalently, solved to optimality by both codes). Columns `gap` and `time` give the final duality gap and the processing time, on average, for each code, restricted to the instances solved by both codes.

Table 3
Comparative `SPF+` × `SPF*`

	solved	gap		optimality	time	
Class	`SPF+` / `SPF*` (=)	`SPF+`	`SPF*`	`SPF+` / `SPF*` (=)	`SPF+`	`SPF*`
A	16 / 22 (16)	4.6	4.2	4 / 3 (3)	488.3	476.1
B	17 / 18 (14)	2.3	1.5	8 / 8 (7)	395.7	298.0
C	17 / 24 (17)	3.1	3.2	4 / 5 (4)	903.7	620.5
D	14 / 20 (14)	1.4	1.4	7 / 11 (6)	405.5	325.1
Total	64 / 84 (61)			23 / 27 (20)		
Avg		2.9	2.6		548.3	429.9

Concerning partial pricing, the pricing heuristic proposed in Section 3.2 could not improve the B&P performance as expected. Although no improvements of using partial pricing on (SPFr) could be verified, `SPFr+` and `SPFr*` solved different instances at optimality. In the other hand, as shown in Table 3, the heuristic of pricing improved the B&P performance for (SPF).

Considering the relaxation of the pricing problem, Table 4 shows that the relaxed pricing proposed in Section 3.1 was effective to deal with the NP-hardness of the pricing problem. `SPFr+` doubled the total of instances solved to optimality and also speeded up the B&P. Although `SPF+` provides better duality gaps than `SPFr+`, the relaxed pricing procedure could provide duality gaps for all instances. `SPFr+` finished with a larger duality gap than `SPF+`, possibly due to the lower bound quality provided by the relaxed model.

Table 4
Comparative SPF+ × SPFr+

	solved	gap		optimality	time	
Class	SPF+ / SPFr+ (=)	SPF+	SPFr+	SPF+ / SPFr+ (=)	SPF+	SPFr+
A	16 / 45 (16)	4.6	8.5	4 / 6 (2)	192.2	1.2
B	17 / 45 (17)	3.2	6.5	8 / 15 (5)	292.0	0.7
C	17 / 45 (17)	3.1	5.1	4 / 10 (4)	903.7	0.7
D	14 / 45 (14)	1.4	5.5	7 / 20 (3)	370.6	0.9
Total	64 / 180 (64)			23 / 51 (14)		
Avg		3.1	6.4		439.6	0.9

Table 5
Comparative MCF+ × SPFr+

	solved	gap		optimality	time	
Class	MCF+ / SPFr+ (=)	MCF+	SPFr+	MCF+ / SPFr+ (=)	MCF+	SPFr+
A	45 / 45 (45)	47.8	9.4	0 / 6 (0)	*	*
B	45 / 45 (45)	36.9	7.3	1 / 15 (0)	*	*
C	45 / 45 (45)	63.7	6.1	0 / 10 (0)	*	*
D	45 / 45 (45)	56.9	6.5	2 / 20 (0)	*	*
Total	180 / 180 (180)			3 / 51 (0)		
Avg		51.3	7.3		*	*

Finally, we compare the B&P and B&B codes. As shown in Table 5, SPFr+ could solve significantly more instances at optimality than MCF+. Since none of the instances was solved to optimality simultaneously by both codes, processing time could not be verified. However, the duality gap was reduced by a factor 7, on average. These results are due to the tight LBs provided by (SPFr).

5 Conclusions

We analyzed different approaches to improve the performance of a branch-and-price algorithm to solve a set partitioning formulation for the ring-tree facility location problem. Computational experiments show that the improvements were due to the primal heuristic and the relaxation of the pricing problem. The primal heuristic provided good upper bounds and also columns to populate the initial basis, which were responsible for almost of columns in the best solution. In our experiments, only small instances (up to 51 vertices) could be treated by the current implementations. As further work, we intend to adapt valid inequalities from related problems and

analyze their efficacy in a branch-and-cut-and-price algorithm.

References

[1] Abe, F. H. N., E. A. Hoshino and A. Hill, *The ring tree facility location problem*, Electron Notes Discret Math **50** (2015), pp. 331 – 336, LAGOS'15 — VIII Latin-American Algorithms, Graphs and Optimization Symposium.

[2] Achterberg, T., *SCIP: Solving constraint integer programs*, Math Progr Comput **1** (2009), pp. 1–41.

[3] Baldacci, R., M. Dell'Amico and J. Salazar, *The capacitated m-ring star problem*, Oper Res **55** (2007), pp. 1147–1162.

[4] Baldacci, R., A. Hill, E. A. Hoshino and A. Lim, *Pricing strategies for capacitated ring-star problems based on dynamic programming algorithms*, Eur J Oper Res **262** (2017), pp. 879 – 893.

[5] Baldacci, R., A. Mingozzi and R. Roberti, *New route relaxation and pricing strategies for the vehicle routing problem*, Oper Res **59** (2011), pp. 1269–1283.

[6] Croes, G. A., *A method for solving traveling-salesman problems*, Oper Res **6** (1958), pp. 791–812.

[7] Edmonds, J., *Optimum branchings*, J Res National Bureau of Standards **71B** (1967), pp. 233–240.

[8] Feillet, D., P. Dejax and M. Gendreau, *Traveling salesman problems with profits*, Transp Sci **39** (2005), pp. 188–205.

[9] Feo, T. A. and M. G. Resende, *Greedy randomized adaptive search procedures*, J Global Optim **6** (1995), pp. 109–133.

[10] Hill, A., *Multi-exchange neighborhoods for the capacitated ring tree problem*, in: I. Dimov, S. Fidanova and I. Lirkov, editors, *Numerical Methods and Applications*, Lecture Notes in Computer Science, Springer International Publishing, 2015 pp. 85–94.

[11] Hill, A., R. Baldacci and E. A. Hoshino, *Capacitated ring arborescence problems with profits*, OR Spectrum (2018).

[12] Hill, A. and S. Schwarze, *Exact algorithms for bi-objective ring tree problems with reliability measures*, Comput Oper Res **94** (2018), pp. 38 – 51.

[13] Hill, A. and S. Voß, *Optimal capacitated ring trees*, EURO J Comput Optim **4** (2016), pp. 137–166.

[14] Hill, A. and S. Voß, *Generalized local branching heuristics and the capacitated ring tree problem*, Discret Appl Math **242** (2018), pp. 34 – 52.

[15] Hoshino, E. A. and C. C. De Souza, *A branch-and-cut-and-price approach for the capacitated m-ring–star problem*, Discret Appl Math **160** (2012), pp. 2728–2741.

[16] Hoshino, E. A. and A. Hill, *Column generation approach for the capacitated ring tree problem* (2014), ALIO/EURO 2014.

[17] Klincewicz, J. G., *Hub location in backbone/tributary network design: a review*, Locat Sci **6** (1998), pp. 307 – 335.

[18] Park, J. and B.-I. Kim, *The school bus routing problem: A review*, Eur J Oper Res **202** (2010), pp. 311 – 319.

[19] Riera-Ledesma, J. and J.-J. Salazar-González, *Solving school bus routing using the multiple vehicle traveling purchaser problem: A branch-and-cut approach*, Comput Oper Res **39** (2012), pp. 391 – 404.

[20] Riera-Ledesma, J. and J. J. Salazar-González, *A column generation approach for a school bus routing problem with resource constraints*, Comput Oper Res **40** (2013), pp. 566 – 583.

[21] Rodríguez-Martín, I., J.-J. Salazar-González and H. Yaman, *A branch-and-cut algorithm for two-level survivable network design problems*, Comput Oper Res **67** (2016), pp. 102 – 112.

[22] Uchoa, E., R. Fukasawa, J. Lysgaard, A. Pessoa, M. P. de Aragão and D. Andrade, *Robust branch-cut-and-price for the capacitated minimum spanning tree problem over a large extended formulation*, Math Progr **112** (2008), pp. 443–472.