A Robust Branch-Cut-and-Price Algorithm for the Heterogeneous Fleet Vehicle Routing Problem

Artur Pessoa, Eduardo Uchoa
{artur,uchoa}@producao.uff.br
Departamento de Engenharia de Produção
Universidade Federal Fluminense

Marcus Poggi de Aragão
poggi@inf.puc-rio.br
Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro

Abstract

This paper presents a robust branch-cut-and-price algorithm for the Heterogeneous Fleet Vehicle Routing Problem (HFVRP), vehicles may have distinct capacities and costs. The columns in the formulation are associated to q-routes, a relaxation of capacitated elementary routes that makes the pricing problem solvable in pseudo-polynomial time. Powerful new families of cuts are also proposed, which are expressed over a very large set of variables. Those cuts do not increase the complexity of the pricing subproblem. Experiments are reported where instances with up to 75 vertices were solved to optimality, a major improvement with respect to previous algorithms.

keywords: Extended Formulations, Column generation, Cutting planes.

1 Introduction

This work considers a direct generalization of the classical Capacitated Vehicle Routing Problem (CVRP): the *Heterogeneous Fleet Vehicle Routing Problem* (HFVRP). Instead of assuming that all vehicles are identical, there is an availability of several vehicle types, with different characteristics. This generalization is very important to the operations research practice, since most actual vehicle routing applications deal with heterogeneous fleets. This problem has mostly been tackled by heuristics, some recent references include [5, 10, 14, 15, 17]. Several authors explicitly commented about the difficulty of solving HFVRP instances to optimality or even of finding strong lower bounds. We could not find any work claiming optimal solutions on any benchmark instance, even those with just 20 clients.

We define formally the problem. Let $G = (V, A)$ be a directed graph with vertices $V = \{0, 1, \ldots, n\}$ and $m = |A|$ arcs. Vertex 0 is the *depot*. The *clients* are the remaining vertices. Each client vertex i is associated with a positive integer demand $d(i)$. Depot demand $d(0)$ is defined as zero. There exists a fleet of K vehicle types, with integral capacities $\{C_1, \ldots, C_K\}$ and fixed costs $\{f_1, \ldots, f_K\}$. For each arc $a \in A$ and each vehicle type k there is a nonnegative traveling cost c_a^k. The HFVRP consists of finding a set of vehicle routes satisfying the following

constraints: (i) each route, starting and ending at the depot, is assigned to a vehicle type k, (ii) the total demand of all clients in a route is at most the capacity C_k of the chosen type, and (iii) each client is included in a single route. The goal is to minimize the sum of traveling and fixed costs, both depending on the vehicle type. We assume that $C_1 \leq \ldots \leq C_K$. For ease of notation, C will be used as a shorthand for C_K, the largest capacity.

The first lower bounds for the HFVRP were introduced by Golden et al. [6]. Stronger bounding schemes were proposed recently. Westerlund et al. [18] give an extended formulation and some valid cuts. This was only tested on a small-sized instance with 12 clients. Choi and Tcha [2] produced lower bounds by using column generation. They are not good enough to build an effective branch-and-price algorithm. However, the approach of solving the restricted MIP provided by the columns generated at the root node proved to be a successful heuristic, obtaining the best known upper bounds on most benchmark instances from the literature. Yaman [19] performed a deep theoretical analysis of several different formulations and valid cuts. Practical experiments showed that this can lead to good lower bounds, but they are still not good enough for allowing the exact solution of the test instances.

The present work departs from an extended HFVRP formulation to generate valid cuts. It is shown that these cuts can be incorporated into a *Branch-Cut-and-Price* (BCP) algorithm in a *robust* way, i.e., without increasing the complexity of the pricing, as defined by Poggi de Aragão and Uchoa [13]. For ease of presentation, as done in a previous version of this work [11], all discussions about formulations and cuts in Section 2 will assume that the traveling costs are identical for all vehicles, i.e., for each $a \in A$ and for every vehicle type k, c_a^k is equal to a value c_a. Vehicles types still have distinct capacities and fixed costs. However this article shows (Section 3) that addressing the more general case where traveling costs may also differ only requires a slight change in the column generation part. The computational experiments described in Section 4 show that strong lower bounds are obtained in both cases. In particular, most instances with up to 75 clients can now be solved to optimality in reasonable times. This article also differs from [11] by the more detailed descriptions of the main algorithms.

2 Formulations and Valid Cuts

We give an extended formulation for the HFVRP (assuming identical traveling costs) similar to the one given by Picard and Queyranne [12] for the time-dependent TSP. Define $V_+ = \{1, \ldots, n\}$ as the set of all clients. Let binary variables x_a^d indicate that arc $a = (i, j)$ belongs to a route, for some vehicle type, and that the total demand of the remaining vertices in the route, including j, is exactly d. The *capacity-indexed* formulation follows:

$$\text{Minimize} \quad \sum_{a \in A} \sum_{d=0}^{C} \hat{c}_a^d x_a^d \tag{1a}$$

S.t.

$$\sum_{a \in \delta^-(i)} \sum_{d=1}^{C} x_a^d = 1 \qquad (\forall\, i \in V_+), \tag{1b}$$

$$\sum_{a \in \delta^-(i)} x_a^d - \sum_{a \in \delta^+(i)} x_a^{d-d(i)} = 0 \quad (\forall\, i \in V_+;\ d = d(i), \ldots, C), \tag{1c}$$

$$x_a^d \in \{0, 1\} \qquad (\forall\, a \in A;\ d = 0, \ldots, C), \tag{1d}$$

$$x_{(i,0)}^d = 0 \qquad (\forall\, i \in V_+;\ d = 1, \ldots, C). \tag{1e}$$

Equations (1b) are in-degree constraints. Equations (1c) state that if an arc with index d enters vertex i then an arc with index $d - d(i)$ must leave i. Costs $\hat{c}_a^d$, $a = (i, j)$, are equal to c_a if $i \neq 0$. Otherwise they are defined as $c_a + f_k$, where k is the index of the cheaper vehicle (in terms of fixed cost) with capacity $C_k \geq d$. Variables to the depot with index distinct from 0 can be removed. Note that variables $x_{(i,j)}^d$ with $d > C - d(i)$ can also be removed. To provide a simpler and precise notation when using the capacity-indexed variables, we define a directed multigraph $G_C = (V, A_C)$, where A_C contains arcs $(i, j)^d$, for each $(i, j) \in A$, $d = 1, \ldots, C - d(i)$, and $(i, 0)^0$, $i \in V_+$. In this context, it is assumed that $\delta^-(S)$ and $\delta^+(S)$ are the subsets of arcs in A_C, with any index, entering and leaving S, respectively. Denote by $\delta_d^-(S)$ and $\delta_d^+(S)$ the sets of arcs with index d entering and leaving S. For example, equations (1b) will be written as

$$\sum_{a^d \in \delta^-(i)} x_a^d = 1, \ (\forall\, i \in V_+).$$

Working directly with this formulation is only practical for small values of capacity, as there are $O(mC)$ variables and $O(nC)$ constraints. The capacity-indexed formulation can be naturally rewritten in terms of q-routes. A q-route [3] is a walk that starts at the depot vertex, traverses a sequence of client vertices with total demand at most equal to C, and returns to the depot. Some vertices may be visited more than once, therefore the set of q-routes strictly contains the set of actual routes. A q-route also defines as a sequence of capacity-indexed arcs as follows. The sequence starts with the arc $(0, j)^D$ where j is the first visited vertex (after the vehicle leaves the depot) and D is the sum of the demands of all visited vertices (if a given vertex is visited t times, then its demand is counted t times). Then, each arc $(i, j)^d$ in the sequence is followed by the arc $(j, \ell)^{d-d(j)}$, where ℓ is the vertex visited immediately after j. The sequence ends necessarily with the arc $(i, 0)^0$, where i is the last visited vertex before the vehicle returns to the depot. For example, if $C = 5$ and all vertex demands are unitary, a q-route that visits the vertices 1,2,3,2 in this order (denote by $0 \to 1 \to 2 \to 3 \to 2 \to 0$) would correspond to the sequence of arcs $((0, 1)^4, (1, 2)^3, (2, 3)^2, (3, 2)^1, (2, 0)^0)$.

Suppose we have p possible q-routes. Define q_{aj}^d as one if the arc a^d is used by the j-th such q-route, and zero otherwise. The reformulation follows:

$$\text{Minimize} \quad \sum_{a^d \in A_C} \hat{c}_a^d x_a^d \tag{2a}$$

S.t.

$$\sum_{j=1}^{p} q_{aj}^d \lambda_j - x_a^d = 0 \quad (\forall\, a^d \in A_C), \tag{2b}$$

$$\sum_{a^d \in \delta^-(i)} x_a^d = 1 \quad (\forall\, i \in V_+), \tag{2c}$$

$$\lambda_j \geq 0 \quad (j = 1, \ldots, p_k), \tag{2d}$$

$$x_a^d \in \{0, 1\} \quad (\forall\, a^d \in A_C). \tag{2e}$$

It can be noted that equalities (1c) are implied by the definition of q-routes together with (2b). In fact, (1) and (2) are equivalent in terms of their linear relaxation bounds. Eliminating the x variables and relaxing the integrality constraints, we get the Dantzig-Wolfe Master (DWM)

relaxation:

$$\text{Minimize} \quad \sum_{j=1}^{p} \Big(\sum_{a^d \in A_C} q_{aj}^d \hat{c}_a^d \Big) \lambda_j \tag{3a}$$

$$\text{S.t.}$$

$$\sum_{j=1}^{p} \Big(\sum_{a^d \in \delta^-(i)} q_{aj}^d \Big) \lambda_j = 1 \quad (\forall\, i \in V_+), \tag{3b}$$

$$\lambda_j \geq 0 \qquad (j = 1, \ldots, p). \tag{3c}$$

This LP has an exponential number of variables, but can be efficiently solved by column generation. The pricing subproblem amounts to finding minimum reduced cost q-routes with capacity C. This can be done by a single call to a dynamic programming procedure with the pseudo-polynomial complexity of $O(mC)$, as described in section 3.1.1. The pricing complexity does not depend on K.

In order to obtain stronger lower bounds one may also add cuts. A generic cut i over the extended variables

$$\sum_{a^d \in A_C} \alpha_{ai}^d x_a^d \geq b_i \tag{4}$$

can be also included in the DWM LP as

$$\sum_{j=1}^{p} \Big(\sum_{a^d \in A_C} \alpha_{ai}^d q_{aj}^d \Big) \lambda_j \geq b_i. \tag{5}$$

Suppose that, at a given instant, we have a total of r cuts in the DWM, the i-th constraint with dual variable β_i. The in-degree constraints have dual variables π. We define the *reduced cost of arc $a = (j, i)$ with index d* as:

$$\bar{c}_a^d = \hat{c}_a^d - \pi_i - \sum_{i=1}^{r} \alpha_{ai}^d \beta_i, \tag{6}$$

where $\pi_0 = 0$. The resulting pricing subproblem after adding cuts, finding minimum cost q-routes with respect to capacity-indexed reduced costs $\bar{c}_a^d$ can still be solved in $O(mC)$ time, basically by the same dynamic programming algorithm. The above reformulation presents remarkable features. It allows the introduction of new cuts over the capacity-indexed variables, even for large values of C, without having to explicitly introduce any new variables and without changing the pricing subproblem.

2.1 Extended Capacity Cuts

We introduce a family of cuts over the capacity-indexed variables. For each vertex $i \in V_+$ the following balance equation is valid:

$$\sum_{a^d \in \delta^-(i)} d x_a^d - \sum_{a^d \in \delta^+(i)} d x_a^d = d(i). \tag{7}$$

Let $S \subseteq V_+$ be a set of vertices. Summing the equalities (7) corresponding to each $i \in S$, we get the *capacity-balance equation over S*:

$$\sum_{a^d \in \delta^-(S)} d x_a^d - \sum_{a^d \in \delta^+(S)} d x_a^d = d(S). \tag{8}$$

It can be noted that those equations are always satisfied by the solutions of (3) (translated to the x^d space by (2b)). Nevertheless, they can be viewed as the source of a rich family of cuts.

Definition 1 *An* Extended Capacity Cut *(ECC) over S is any inequality valid for $P(S)$, the polyhedron given by the convex hull of the 0-1 solutions of* (8)*.*

2.1.1 Strengthened rounded capacity cuts

The rounded capacity cuts over the traditional arc variables is

$$\sum_{a \in \delta^-(S)} x_a \geq k(S) := \lceil d(S)/C \rceil. \tag{9}$$

Noting that $x_a = \sum_{d=0}^{C} x_a^d$, this cut can be derived only from the above definition: for a given S relax (8) to $\geq$, divide both sides by C and round coefficients up. Remember that $\delta^+(S)$ contains no arc with capacity C, so all such coefficients are rounded to zero. All coefficients corresponding to $\delta^-(S)$ are rounded to one. Therefore rounded capacity cuts are ECCs. However, a stronger family of cuts can be defined over the extended space as shown next.

For any $S \subseteq V_+$, the following inequality is clearly valid for $P(S)$:

$$\sum_{a^d \in \delta^-(S)} d x_a^d \geq d(S). \tag{10}$$

Define $d^* = d(S) - C(k(S) - 1) - 1$. If at least one variable x_a^d with $d \leq d^*$ is set to one, we still need $k(S)$ additional variables set to one to satisfy (10). Otherwise, if all variables set to one have $d > d^*$, we need at least $k(S)$ such variables to satisfy (10). Hence

$$\frac{k(S)+1}{k(S)} \sum_{a^d \in \delta^-(S) \,:\, d > d^*} x_a^d + \sum_{a^d \in \delta^-(S) \,:\, d \leq d^*} x_a^d \geq k(S) + 1 \tag{11}$$

is also valid for $P(S)$ and dominates (9). In [16], it is shown that (11) also dominates root cut-sets inequalities. An even stronger inequality can be obtained by considering that some arcs leaving S with a sufficiently large demand index may receive a negative coefficient:

$$\begin{aligned} \frac{k(S)+1}{k(S)} \sum_{a^d \in \delta^-(S) \,:\, d > d^*} x_a^d + \sum_{a^d \in \delta^-(S) \,:\, d \leq d^*} x_a^d & \\ - \frac{1}{k(S)} \sum_{a^d \in \delta^+(S) \,:\, d \geq d'} x_a^d & \geq k(S) + 1, \end{aligned} \tag{12}$$

where d' is the smallest integer such that

$$d' \geq d^* + 1 - \min_{i \in S} \{d(i)\} \tag{13}$$

and

$$d' \geq C - d^*. \tag{14}$$

Proposition 1 *Inequality* (12) *is valid for* (1)*.*

Proof: Given a feasible solution $\bar{x}$ for (1), associate each arc $a_1^{d_1} \in \delta^-(S)$ to the next arc $a_2^{d_2} \in \delta^+(S)$ in the same route. We divide into two cases: (i) all pairs of associated arcs have $d_1 - d_2 > d^*$, and (ii) at least one pair of associated arcs has $d_1 - d_2 \leq d^*$. In the case (i), (14) ensures that all variables x_a^d with negative coefficients in (12) will be zero. Then (12) is satisfied because, by the definition of $k(S)$, we have at least $k(S)$ arcs of $\bar{x}$ entering S. In the case (ii), by the definition of d^*, we need at least $k(S)+1$ arcs entering S. Moreover, (13) ensures that every leaving arc $a_2^{d_2}$ of $\bar{x}$ with $d_2 \geq d'$ has its associated entering arc $a_1^{d_1}$ with $d_1 > d^*$. Hence, the total contribution of each pair of associated arcs to the left-hand side of (12) is at least $(k(S)+1)/k(S) - 1/k(S) = 1$. ■

This particular ECC (12) will be referred to as a *strengthened rounded capacity cut.*

2.1.2 Homogeneous Extended Capacity Cuts

The *Homogeneous Extended Capacity Cuts* (HECCs) are a subset of the ECCs where all entering variables with the same capacity have the same coefficients, the same happening with the leaving variables. For a given set S, define aggregated variables y^d and z^d as follows:

$$y^d = \sum_{a^d \in \delta_d^-(S)} x_a^d \qquad (d = 1, \ldots, C), \tag{15}$$

$$z^d = \sum_{a^d \in \delta_d^+(S)} x_a^d \qquad (d = 0, \ldots, C). \tag{16}$$

The capacity-balance equation over those variables is:

$$\sum_{d=1}^{C} dy^d - \sum_{d=0}^{C} dz^d = d(S) \quad . \tag{17}$$

For each possible pair of values of C and $D = d(S)$, we may define the polyhedron $P(C, D)$ induced by the integral solutions of (17). The inequalities that are valid for those polyhedra are HECCs. In Subsection 3.2 we illustrate how valid cuts can be derived and separated from that equality. The HECCs have been firstly proposed in [16] for the CMST problem.

2.2 Triangle Clique Cuts

Let $S \subseteq V_+$ be a set of exactly three vertices. The triangle clique cuts are a family of cuts defined over the variables x_a^d, with $a^d = (i,j)^d \in A_C$ and $i, j \in S$. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the compatibility graph where each vertex of $\mathcal{V}$ is a capacity-indexed arc $a^d = (i,j)^d \in A_C$ with $i, j \in S$. In this case, an edge $e = (a_1^{d_1}, a_2^{d_2})$ belongs to $\mathcal{E}$ if and only if $a_1^{d_1}$ and $a_2^{d_2}$ are *compatible.* For all distinct $i, j, k \in S$, there are four cases:

Case 1: if $e = ((i,j)^{d_1}, (i,k)^{d_2})$, then $e \notin \mathcal{E}$;

Case 2: if $e = ((i,j)^{d_1}, (k,j)^{d_2})$, then $e \notin \mathcal{E}$;

Case 3: if $e = ((i,j)^{d_1}, (j,k)^{d_2})$ and $d_1 \neq d_2 + d(j)$, then $e \notin \mathcal{E}$;

Case 4: if $e = ((i,j)^{d_1}, (j,k)^{d_2})$ and $d_1 = d_2 + d(j)$, then $e \in \mathcal{E}$;

For any independent set $I \subset \mathcal{V}$, it is clear that the following inequality is valid

$$\sum_{a^d \in I} x_a^d \leq 1. \tag{18}$$

This is a well-known clique cut. However $\mathcal{G}$ has a nice structure that can be explored to build a very efficient separation procedure, as will be shown in Subsection 3.2.

3 A Robust Branch-Cut-and-Price Algorithm

3.1 Column generation

Recall that the reduced cost of a λ_j variable in (3) is the sum of the reduced costs $\bar{c}_a^d$ of the arcs in the corresponding q-route. Those reduced costs are calculated using equations (6). The pricing subproblem of finding the q-routes yielding a variable with minimum reduced cost is NP-hard, but can be solved in pseudo-polynomial $O(mC)$ time using the following dynamic programming procedure.

3.1.1 Basic pricing algorithm

Define a matrix R such that the entry $R(d, i)$ represents the minimum-reduced cost partial q-route that starts at vertex $i \in V_+$ with total demand exactly d and ends at the depot. The reduced costs of the arcs that leave the depot are not considered in R. The entry $R(d, i)$ contains a *label* consisting of the vertex i, the reduced cost of the q-route (denoted by $\bar{c}(R(d, i))$), and a pointer to a label representing the walk down to the next vertex. Initially, the entries $R(d(i), i)$ are initialized with labels corresponding to a single arc $(i, 0)^0$ to the depot, for all $i \in V_+$. All entries $R(d, i)$ such that $d < d(i)$ are also initialized with labels having infinity costs. The dynamic programming recursion given by

$$\bar{c}(R(d, i)) = \min_{(i,j)^d \in \delta^+(i)} \{\bar{c}_{(i,j)}^d + \bar{c}(R(d - d(i), j))\}$$

is used to fill the entries of matrix R. At the end, we will have the minimum-reduced cost q-route with total demand exactly d that starts at each vertex $i \in V_+$, for $d = d(i), \ldots, C$ (after the arcs that leaves the depot are inserted). For each vertex i, we choose the total demand d^* that minimizes the reduced cost of the corresponding q-route (given by $c_{(0,i)}^{d^*} + R(d^*, i)$) to insert in the LP (3). There are nC entries in the matrix. Moreover, the reduced cost of each capacity-indexed arc is accessed exactly once. For each accessed arc, the procedure takes $O(1)$ time to update the matrix. So the total running time is $O(|A_C|) = O(mC)$.

3.1.2 Dealing with undirected instances

Our HFVRP formulation is directed, but the benchmark instances from the literature are undirected. This should not be a problem, the directed problem is more general and contains the undirected case. However this leads to a symmetric cost structure, allowing two representations (the two possible arc orientations) for what is essentially the same q-route (or route). As a consequence, a convergence difficulty may appear. Cuts that are asymmetric regarding the arcs that enter or exit a subset of vertices may become not violated by simply changing the orientation of one or a few q-routes, without increasing the lower bound. This is the case for all the ECCs.

We can deal with this difficulty by requiring that, in undirected instances, the first client visited by a q-route must have a smaller index than the last client. The modified algorithm has a worst case complexity of $O(nmC \log n)$, but our experiments show that the average case performance is much smaller. In practice, this symmetry breaking strategy only introduces a small factor on the computing time.

3.1.3 Pricing with Cycle Elimination and Symmetry Breaking

Houck et al. [7] and [3] already noted that one can find minimum reduced cost q-routes without 2-cycles (subpaths $i \to j \to i$, $i, j \neq 0$) without changing the time complexity of the princing algorithm. This immediately leads to a stronger formulation.

Our pricing algorithm combines the 2-cycle-elimination of Houck et al. with the symmetry breaking strategy described in 3.1.2. The algorithm operates as the basic dynamic programming introduced in 3.1.1, filling a $C \times n$ matrix with partial q-routes. Labels retain the exact same meaning as before, but now each entry in the matrix contains no longer a single label, but a *bucket* of labels. Therefore, bucket $R(d, i)$ represents not only the cheapest 2-cycle-free partial q-route with total demand d that starts in i, but also alternative partial q-routes that ensure that all possible extensions from from the depot to i are considered. When considering the possible extensions of a partial q-route, only the first (due to the symmetry breaking criterion) and the last but one (due to 2-cycle elimination) vertices visited by the extension are important. For example, the partial q-route $1 \to 2 \to 4 \to 0$ can have any extension of type $0 \to 3 \to \cdots \to 5 \to 1$ because the index of the first visited vertex (3) is smaller than that of the last visited vertex (4) and $5 \neq 2$, which ensures that no 2-cycle is formed when the extension and the partial q-route are connected by the vertex 1.

We limit the number of labels in each bucket by eliminating all dominated labels. A label ℓ is *dominated* by a set of labels $\mathcal{L}$ if all labels in $\mathcal{L}$ have reduced costs smaller than or equal to that of ℓ and every possible extension for ℓ can also legally extend some label of $\mathcal{L}$. The set of all extensions that can legally extend at least one partial q-route of $\mathcal{L}$ can be compactly represented by a quadruple $(i_0, i_1, i_2, i_3) \in V_+^4$. A general extension $0 \to j_1 \to \cdots \to j_2 \to j_3$ belongs to the set represented by the quadruple (i_0, i_1, i_2, i_3) when $j_3 = i_0$ and either $j_1 < i_1$ or both $j_2 \neq i_2$ and $j_1 < i_3$. Note that i_0 is just the vertex where the extensions are connected to the partial q-routes, i_1 is a limit for the index of the first vertex that ensures that a compatible q-route will be found, and i_3 is another (less restrictive) limit for this index that also requires a condition on the last but one visited vertex (of being different from i_2) to avoid forming a 2-cycle. For example, consider the following set of partial q-routes:

$$2 \to 4 \to 7 \to 0$$
$$2 \to 3 \to 6 \to 4 \to 0$$
$$2 \to 1 \to 5 \to 0$$

In this case, the corresponding set of extensions is represented by the quadruple $(2, 5, 4, 7)$ because the first and the third partial q-routes allows any extension whose index of the first vertex is smaller than 5, and the first partial q-route allows any extension whose index of the first vertex is smaller than 7 that do not form a 2-cycle with it (e.g. the last but one vertex is not 4).

In order to remove the dominated labels from a bucket, one must sort the q-routes in a non-decreasing order by their reduced costs and sequentially update the quadruple that corresponds to all possible extensions. Whenever the quadruple does not change, the corresponding q-route is

dominated by the previous ones. Since any change in the quadruple (i_0, i_1, i_2, i_3) increases either i_1 or i_3, we must have at most $2n$ non-dominated q-routes. However, having $\theta(n)$ non-dominated q-routes requires that such routes, when sorted by reduced costs are also roughly sorted by the index of the last vertex, which is very unlikely. As a result, the average bucket size is much smaller than the worst case, which explains the better performance observed experimentally.

3.1.4 Dealing with traveling costs depending on the vehicle type

The more general HFVRP may have costs c_a^k depending on the vehicle type. Dealing with this case requires solving K distinct subproblems. Subproblem k amounts to finding a minimum cost q-route with capacity up to C_k and with respect to the following set of reduced costs:

$$\bar{c}_a^d(k) = \hat{c}_a^d(k) - \sum_{i=1}^{m} \alpha_{ai}^d \beta_i, \tag{19}$$

where costs $\hat{c}_a^d(k)$, $a = (i, j)$, are equal to c_a^d if $i \neq 0$ and $c_a^d + f_k$ otherwise. Remark that, as before, columns corresponding to q-routes of any vehicle type are introduced in (3) without any special distinction. Only the column cost identifies the vehicle type.

3.2 Separation routines

Let $\bar{\lambda}$ be a fractional solution of (3). This solution can be converted into a $\bar{x}$ solution over the capacity-indexed arc space using equations (2b). Violated cuts of form (4) can be separated and added to (3) as (5).

Extended Capacity Cuts Our procedure starts by choosing candidate sets S. Those candidates include:

- All sets S up to cardinality 6 which are connected in the support graph of the fractional solution $\bar{x}$, i.e. the subgraph of G containing only the arcs a where some value $\bar{x}_a^d$ is positive. This connectivity restriction prevents an explosion on the number of enumerated sets. As proved in [16], if an ECC is violated over a set S composed of two or more disconnected components, there exists another violated ECC over one of those smaller components.
- The sets with cardinality larger than 6 that are inspected in the heuristic separation of rounded capacity cuts presented in [9]. The rationale is that if the rounded capacity cut is almost violated for a given set S, it is plausible that an extended capacity can be violated over that set. In particular, if the rounded capacity cut is violated, the ECC (12) will be certainly violated. The use of ECCs over large sets S is tested for the first time in this paper.

So, for each candidate set S, we first check if the strengthened rounded capacity cut (12) is violated. Then we try to separate HECCs from the equation (17) over S. In particular, we look for inequalities of the following form:

$$\sum_{d=1}^{C} \lceil rd \rceil y^d - \sum_{d=1}^{C-1} \lfloor rd \rfloor z^d \geq \lceil rd(S) \rceil \quad , \tag{20}$$

where $0 < r \leq 1$. As discussed in [16], at most $0.3C^2$ rational multipliers r need to be tried in this integer rounding procedure.

We illustrate the procedure. Figure 1 displays part of a fractional x_a^d solution of a unitary demand instance with $C = 4$, over a set $S = \{1, 2, 3\}$. The set S is being covered by 3 different q-routes with no cycles, each one with associated λ variable equal to 1/2. The first q-route enters S at vertex 1 with index 4 (arc a) and leaves the set at vertex 2 (arc d) with index 2. The second q-route enters at vertex 1 (arc b) with index 2 and leaves the set directly to the depot (arc h). The third q-route enters at vertex 3 (arc c) with index 2 and also leaves the set to the depot (arc i). Relaxing equation (17) to $\geq$, multiplying by 1/2 and performing integer rounding, a violated HECC is found:

$$2y^4 + 2y^3 + y^2 + y^1 - z^3 - z^2 \geq 2.$$

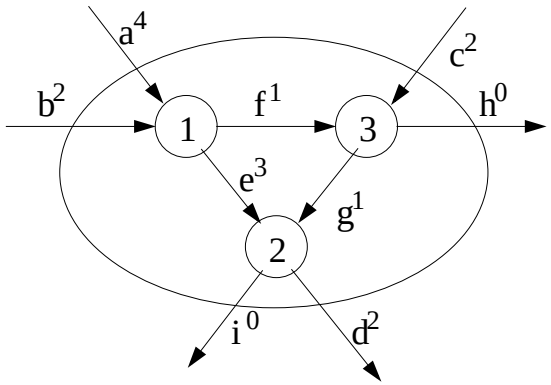

Figure 1: Part of a fractional solution over the extended variables.

Triangle Clique Cuts The separation procedure for the triangle clique cuts finds the independent set $I \subset \mathcal{V}$ in $\mathcal{G}$ that maximizes $\sum_{a^d \in I} \bar{x}_a^d$. Although the problem of finding a maximum-weight independent set is strongly NP-hard for general graphs, such an independent set can be found for $\mathcal{G}$ in linear time by exploiting its specific structure.

Proposition 2 *The graph $\mathcal{G}$, for sets S of cardinality three, is a set of chains.*

Proof: By the definition given in 2.2, each arc that corresponds to a vertex of $\mathcal{G}$ is compatible with at most one arc with higher demand and at most one with lower demand. Hence, the vertex degree of $\mathcal{G}$ is at most two and there is no cycle. ■

Figure 2 shows that the compatibility graph $\mathcal{G}$ for the example of Figure 1 (considering demand indices up to 3) is a set of 6 chains. For instance, the arc $(1,2)^3$ is only compatible with $(2,3)^2$ in the triangle $\{1,2,3\}$. The arc $(2,3)^2$ is compatible with both $(1,2)^3$ and $(3,1)^1$, and $(3,1)^1$ is only compatible with $(2,3)^2$. Observe that $(1,2)^3$ and $(3,1)^1$ are not compatible. A violated triangle clique cut corresponding to a maximal independent set in $\mathcal{G}$ is:

$$x_e^3 + x_{(2,3)}^3 + x_{(3,1)}^3 + x_f^3 + x_g^3 + x_{(2,1)}^3 + x_e^1 + x_{(2,3)}^1 + x_{(3,1)}^1 + x_f^1 + x_g^1 + x_{(2,1)}^1 \leq 1.$$

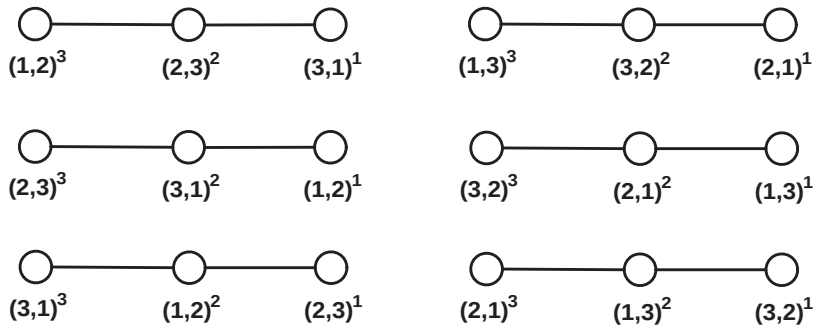

Figure 2: The compatibility graph $\mathcal{G}$ for the example of Figure 1.

A set I is a maximum-weight independent set for a set of chains if and only if it is the union of maximum-weight independent sets for each single chain. The maximum weight independent set for a single chain H can be obtained in a linear time through a simple dynamic programming procedure. Let $a_i^{d_i}$ be the ith vertex in H, numbered from 1 to $|H|$ from one extreme to the other of the chain (path). Also, let us define $I^*(i,1)$ (resp. $I^*(i,0)$) as the maximum independent set for the subchain containing the first i vertices of H that does (resp. does not) use the ith vertex. Finally, let $c(I) = \sum_{a^d \in I} \bar{x}_a^d$. We have the following recurrence:

$$c(I^*(i,1)) = \bar{x}_{a_i}^{d_i} + c(I^*(i-1,0));$$
$$c(I^*(i,0)) = \max\{c(I^*(i-1,0)), c(I^*(i-1,1))\}.$$

It is interesting to add suitable positive perturbations to the values of $\bar{x}_a^d$ that are zero, in order to generate lifted inequalities with as many non-zero coefficients as possible.

3.3 Branching with route enumeration

We branch over the edges of the undirected graph associated to G. We choose the pair $\{i,j\}$ such that the value $\bar{x}_{\{i,j\}} = \sum_{d=0}^{C}(\bar{x}_{(i,j)}^d + \bar{x}_{(j,i)}^d)$ is closer to 0.65. On the left branch node we require that $\bar{x}_{\{i,j\}}$ must be 0, on the right branch node this must be greater than or equal to 1. Next, we give an argument that this is a valid branching strategy. Suppose every pair of vertices is either fixed to zero or set to greater than or equal to one and the DWM LP relaxation remains feasible. Let $G' = (V,E)$ be the undirected graph such that E contains the edges set to be greater than or equal to one. Clearly, this graph can be decomposed into cycles and single edges, each one containing the root vertex, where each vertex of V_+ belongs to exactly one cycle or single edge. For each single edge, only the route that goes from the root to one vertex and back to the root is possible. On the other hand, for each of the cycles, two routes are possible, with opposite senses. However, in any basic feasible solution, only one of these routes are non-zero.

However, in order to improve the performance of our algorithm, we combine this traditional branching with a route enumeration technique inspired by the one described by Baldacci et al. [1] that we describe later. If the integrality gap, the difference between the best known feasible solution value and the current LP relaxation optimal value is sufficiently small, they found that it may be practical to enumerate all possible relevant elementary q-routes, i.e. all routes that have a chance of being part of the optimal solution. A route is non-relevant if at least one of the following two conditions hold:

(i) its reduced cost (with respect to the current values of (6)) is greater than the gap, or

(ii) there exists another route visiting the same set of clients with smaller cost (with respect to the original arc costs $\hat{c}_a^d$).

If the number of relevant routes is not too large (say, in the range of thousands), the overall problem may be solved by feeding a general MIP solver with a set-partitioning formulation containing only those routes. If this set-partitioning can be solved, the optimal solution will be found and no branch will be necessary. Sometimes this leads to very significant speedups when compared to traditional branching strategies. However, it should be remarked that such route enumeration is an inherently exponential procedure. Its practical performance depends crucially on the gap value and it is also very sensitive to the characteristics of the instance that is being solved. There is no guarantee that a combinatorial explosion will not happen, even on small sized instances.

Our hybrid strategy, devised to provide a robust approach, is to perform limited route enumerations (one for each vehicle type) after each branch-and-bound node is solved. This means that the enumeration is aborted if more than 30,000 relevant routes for some type k were already generated or if more than 200,000 states (partial non-dominated routes) are being kept by our dynamic programming algorithm (see 3.3.1). If those limits are not reached, a set-partitioning containing all relevant routes for each vehicle type is given to a MIP solver. After the set-partitioning is solved, this node is declared as *solved* and no branching will occur. Otherwise, the edge branching is performed and two more nodes must be solved. Since deeper nodes will have smaller gaps, at some point the enumeration will work. The overall effect may be a substantially smaller branch-and-bound tree. In some cases where the traditional branching would need to reach depth 10, the hybrid strategy does not go beyond depth 4.

Our BCP also uses the route enumeration as a heuristic. If the actual gap g of a node is still too large and the limits are reached, we try the enumeration with a dummy gap of $g/2$. If this is still not enough, we try with $g/4$ and so on. If the enumeration now succeeds, we try an increased dummy gap of $(g/2+g/4)/2$. In short, we perform a sort of binary search to determine a dummy gap that will yield a set-partitioning of reasonable size. The dummy gaps are used only in the root node of the BCP to search for improved upper bounds. In our experiments, the best known upper bounds for some instances have been improved in this way. The MIPs constructed with dummy gaps are aborted after 10,000 branch-and-bound nodes are explored.

3.3.1 The route enumeration algorithm

Our route enumeration algorithm uses dynamic programming procedures that generate partial q-routes without s-cycles[1], for different values of s. For that, we use the general algorithm of [8, 4]. Similarly to the algorithm proposed in 3.1.3, this algorithm stores a bucket of labels in each matrix entry and uses a domination criterion to reduce the bucket sizes. The symmetry breaking strategy is not applied in this context.

Our enumeration procedure performs the following three basic steps:

Step 1: Use the basic dynamic programming procedure described in 3.1.1 enhanced with the elimination of 3-cycles to obtain lower bounds on the reduced cost of partial q-routes that finish at the depot.

[1] In a partial q-route without s-cycles, each two visits to the same vertex must have more than s arcs between them.

Step 2: Use an analogous dynamic programming procedure in the reverse direction with the elimination of 5-cycles to obtain lower bounds on the reduced cost of partial q-routes that start at the depot. The lower bounds obtained in Step 1 are used to improve the performance of this step.

Step 3: Use the lower bound obtained in Step 2 to do the enumeration.

The value of $\bar{c}(R(d,i))$ calculated in Step 1 is a lower bound on the reduced cost of any 3-cycle-free partial q-route that starts at vertex i with total demand exactly d and ends at the depot.

The *latent demand* of a partial q-route that starts at the depot and ends at the vertex i is the capacity-index of its last arc. This means that a partial q-route with latent demand d can only be extended toward the depot by partial q-routes with total demand exactly d. Let us define $R'(d,i)$ as the minimum-reduced cost partial q-route that starts at the depot and ends at the vertex i with latent demand d. Note that the concatenation of $R'(d,i)$ and $R(d,i)$ gives a q-route that starts and ends at the depot, and visits the vertex i with remaining total demand (the sum of demands of all vertex visited in the walk from i toward the depot) exactly d. The values of $\bar{c}(R'(d,i))$ can be computed through a dynamic programming procedure that uses the following recursion:

$$\bar{c}(R'(d,i)) = \min_{(j,i)^d \in \delta^-(\{i\})} \{\bar{c}(R'(d+d(j),j)) + \bar{c}^d_{(j,i)}\}.$$

Initially, only the values of $\bar{c}(R'(d,0))$ are set to zero, for $d = 1,\ldots,C_k$. Similarly to the column generation procedure, the basic algorithm uses a matrix with nC_k entries and runs in $O(mC_k)$.

When applying the s-cycle elimination technique of [8, 4], each entry contains a bucket of labels and the overall time complexity becomes $O(mC_k \log n)$ for fixed values of s. The $\log n$ factor in the time complexity is due to sorting the routes in each bucket before eliminating the dominated ones. This factor could be avoided by keeping the buckets always small, but it is not worth doing that in practice because it requires several rounds of eliminations of dominated routes for each bucket. Whenever $\bar{c}(R'(d,i)) + \bar{c}(R(d,i))$ is greater than or equal to the gap, we set the matrix entry corresponding to $\bar{c}(R'(d,i))$ as infinity. This avoids extending expensive the routes in the bucket of $R'(d,i)$ to other buckets, improving the performance of the procedure.

Step 3 also uses a dynamic programming approach to generate elementary q-routes. In this case, since the matrix size is potentially exponential on n, its entries must be dynamically allocated on demand. For each $S \subseteq V^+$ with $d(S) \leq C_k$ and $i \in S$, define the matrix entry $R^e(S,i)$ as the minimum-cost (considering the original costs) elementary partial q-route that starts at the vertex i, ends at the depot, and visits exactly the vertices of S. By the domination condition (ii), at most one partial q-route need to be stored in each matrix entry. We use $c(R^e(S,i))$ and $\bar{c}(R^e(S,i))$ to denote the cost and the reduced cost of the partial elementary q-route stored in the matrix entry $R^e(S,i)$, respectively. Initially, we set only the matrix entries such that $|S| = 1$. Then, we use the following recursion:

$$c(R^e(S,i)) = \min_{(i,j)^{d(S\setminus\{i\})} \in \delta^+(\{i\})} \{c^{d(S\setminus\{i\})}_{(i,j)} + c(R^e(S\setminus\{i\},j))\}.$$

The matrix size is limited through the elimination of any path $R^e(S,v)$ such that $\bar{c}(R'(d(S),i)) + \bar{c}(R^e(S,i))$ is greater than or equal to the gap. Note that the elimination of an optimal path

$R^e(S, v)$ by reduced cost does not imply that the corresponding matrix entry is filled with another path with higher (original) cost. In fact, the combination the two domination conditions is much more effective than each condition alone.

4 Experiments

We tested the resulting algorithm for the HFVRP on the set of instances given in Golden et al. [6], which is the same used in the experiments reported by Yaman [19] and by Choi and Tcha [2]. These instances are identified as "cXX-N" where XX represents its number of clients and N identifies the instance. For each instance there are three problem variants: (FIX) vehicles differ only in their fixed costs; (VAR) vehicles differ in their traveling (variable) costs; and (FIX+VAR) vehicles differ in both fixed and traveling costs. Of course, vehicles differ in their capacities in all such variants. The computational results from Yaman [19] (available only for variant FIX) only provide lower bounds and correspond to experiments running on a Sun Ultra 12×400 MHz. Those from Choi and Tcha [2] (for the three variants) are on a Pentium IV 2.6 GHz and provide valid lower bounds and upper bounds. Our code was executed on a Core 2 Duo running at 2.13 GHz with 2 GB of RAM. Linear programs and set-partitioning MIPs were solved by CPLEX 10.0. In all runs of our branch-cut-and-price, we set the initial upper bound to a value 0.1 units higher than those reported in [2]. We did that because our research was focused on lower bounds and on branching strategies, trying to solve instances to optimality. Instead of investing time implementing a good heuristic (which would also be important to the overall performance our method), we decided "to borrow" from the published results from Choi and Tcha. Of course, a fair comparison of our methods with others should add the running times of [2] to our running times.

The comparison of root lower bounds for variant FIX are presented in Table 1. The columns with header K and C contain the number of different types of vehicles and the largest capacity, respectively. The bounds and CPU time from Yaman [19], obtained with a branch-and-cut on a flow formulation, are given in the two columns labeled **Yam.**. Likewise, the two columns labeled **Choi** present lower bounds and times in Choi and Tcha [2]. They are obtained using a column generation algorithm on q-routes with 2-cycle elimination. The two columns labeled **New** present the results of our algorithm running with 2-cycle elimination and the route symmetry breaking and the cut separation described above. The Time column represents the CPU time to obtain the indicated lower bound. The last two columns in Table 1 report the previous best known upper bounds and the upper bounds by our algorithm. Values proved to be optimal are printed bold. Instances c75-17, c100-19 and c100-20 could not be solved in the time limit of 10,000 seconds. However, the enumeration heuristic could find feasible solutions of value equal (c100-c19) or better than the previous best (c75-17) in two out of those three instances.

Additional BCP statistics on FIX variant are presented in Table 2, where the 2nd, 3rd and 4th columns contain the number of cuts of each type inserted in the root node. The headers **SRCC**, **Rd ECC** and **Clique** mean strengthened rounded capacity cuts, HECCs obtained by integer rounding and triangle clique cuts, respectively. The following four columns contain the execution times spent in the root node with column generation, LP solving, cut separation, and route enumeration + set-partitioning solving, respectively. Note that the time to obtain the root lower bound (reported in Table 1) does not include the route enumeration + set-partitioning time. The last three columns show the total number of open nodes in the BCP, the total number of nodes CPLEX required to solve all the set-partitioning problems, and the overall execution

time by the BCP. No branching was required on instances with up to 50 clients, the number of relevant routes enumerated is small enough to solve the instance by solving a single set-partitioning problem. Instance c75-18 could be solved by the hybrid strategy, branching until the gap is small enough for the enumeration.

Tables 3 and 5 are analogous to Table 1 except for having only the bounds presented in Choi and Tcha [2]. The tables present the results obtained for variants VAR and VAR+FIX, respectively. The statistics regarding cuts inserted and CPU times for variants VAR and VAR+FIX are presented in Tables 4 and 6, respectively, with columns analogous to the ones of Table 2. It can be noted that the FIX+VAR variant is apparently the easiest. Smaller average integrality gap and less branches in the search tree for every instance, together with smaller computing times for almost all solved instances, suggest that. Despite the larger average gaps, VAR instances appears to be less difficult than FIX instances, which seem to be the more demanding variant of the HFVRP.

5 Comments

This article presented a BCP for the HFVRP, featuring the use of cuts defined over the extended formulation. The approach seems very promising and deserves further development. All the new introduced families of cuts were found to be crucial in the consistency of the resulting algorithm. For example, if only the rounded HECC separation is disabled, instance c20-4 takes more than 1 hour to be solved by the same BCP code. Also, the enumeration technique has shown to be an effective heuristic when the current LB is sufficiently close to the optimal solution.

Acknowledgments

AP, MPA and EU were partially financed by CNPq grants 301175/2006-3, 311997/06-6 and 304533/02-5. AP and EU received support from Mestrado em Engenharia de Produção-UFF. The authors also thank the technical support by GAPSO Inc., Brazil.

References

[1] R. Baldacci, L. Bodin, and A. Mingozzi. The multiple disposal facilities and multiple inventory locations rollon-rolloff vehicle routing problem. *Computers and Operations Research*, 33:2667–2702, 2006.

[2] E. Choi and D-W Tcha. A column generation approach to the heterogeneous fleet vehicle routing problem. *Computers and Operations Research*, 34:2080–2095, 2007.

[3] N. Christofides, A. Mingozzi, and P. Toth. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical Programming*, 20:255–282, 1981.

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

Table 1: Comparison of bounds and times on the FIX variant.

Instance	K	C	Yam. LB	Yam. Time(s)	Choi LB	Choi Time(s)	New LB	New Time(s)	Prev UB	New UB
c20-3	5	120	912.40	–	951.61	0.4	961.03	2.5	961.03	**961.03**
c20-4	3	150	6369.51	–	6369.15	0.7	6437.33	5.2	6437.33	**6437.33**
c20-5	5	120	959.29	–	988.01	0.8	1001.12	7.2	1007.05	**1007.05**
c20-6	3	150	6468.44	–	6451.62	0.4	6515.82	4.7	6516.47	**6516.47**
c50-13	6	200	2365.78	397.1	2392.77	10.1	2400.93	107.0	2406.36	**2406.36**
c50-14	3	300	8943.94	175.6	8748.57	50.8	9111.81	277.2	9119.03	**9119.03**
c50-15	3	160	2503.61	142.8	2544.84	10.0	2572.99	72.3	2586.37	**2586.37**
c50-16	3	140	2650.76	142.1	2685.92	11.3	2705.35	48.8	2720.43	**2720.43**
c75-17	4	350	1689.93	1344.8	1709.85	206.9	1717.37	456.9	1744.83	1734.52
c75-18	6	400	2276.31	1922.8	2342.84	70.1	2351.07	347.5	2371.49	**2369.64**
c100-19	3	300	8574.33	1721.2	8431.87	1178.9	8648.68	1017.3	8661.81	8661.81
c100-20	3	200	3931.79	2904.0	3995.16	264.0	4005.29	407.1	4039.49	–
avg gap			2.63%		1.60%		0.44%			

Table 2: BCP statistics on the FIX variants.

Instance	# Root Cuts SRCC	Rd ECC	Clique	Root Times Col Gen	LP	Cut Sep	Enum +SP	# BCP Nodes	# SP Nodes	Total Time
c20-3	13	50	15	1.4	0.4	0.3	0.2	1	1	2.7
c20-4	6	169	14	2.6	0.6	1.1	0.3	1	1	5.5
c20-5	34	81	19	3.5	0.8	1.2	1.0	1	1	8.2
c20-6	1	103	3	2.8	0.4	0.9	0.3	1	1	5.0
c50-13	37	250	38	49.1	3.5	20.8	14.3	1	1	121.3
c50-14	13	1148	51	96.5	27.5	82.9	28.6	1	5	305.8
c50-15	16	475	40	30.8	6.9	17.2	282.9	1	238	355.2
c50-16	21	254	51	23.3	2.4	10.6	211.4	1	91	260.2
c75-17	15	497	87	236.5	20.5	109.8	6271.9	–	0	>10000
c75-18	27	296	55	179.4	3.8	80.2	3326.4	79	475	41607.1
c100-19	41	830	109	598.0	45.2	202.5	1063.8	–	–	>10000
c100-20	21	265	87	236.3	6.4	84.3	6265.9	–	–	>10000

Table 3: Comparison of bounds and times on the VAR variant.

Instance	K	C	Choi LB	Choi Time(s)	New LB	New Time(s)	Prev UB	New UB
c20-3	5	120	616.38	0.19	621.94	2.3	623.22	**623.22**
c20-4	3	150	376.94	0.44	382.61	8.9	387.18	**387.18**
c20-5	5	120	739.72	0.23	742.87	0.6	742.87	**742.87**
c20-6	3	150	406.82	0.92	414.31	16.5	415.03	**415.03**
c50-13	6	200	1469.41	4.11	1487.84	71.5	1491.86	**1491.86**
c50-14	3	300	582.25	20.41	592.71	220.9	603.21	**603.21**
c50-15	3	160	978.04	4.61	989.42	54.8	999.82	**999.82**
c50-16	3	140	1106.12	3.36	1118.16	42.8	1131.00	**1131.00**
c75-17	4	350	1021.86	69.38	1026.87	539.9	1038.60	**1038.60**
c75-18	6	400	1779.41	48.06	1786.59	884.6	1801.40	**1800.80**
c100-19	3	300	1080.68	182.86	1092.43	1344.2	1105.44	1105.43
c100-20	3	200	1501.67	98.14	1511.36	779.0	1530.43	1530.43
avg gap			1.87%		0.84%			

Table 4: BCP statistics on the VAR variants.

Instance	# Root Cuts			Root Times				# BCP	# SP	Total
	SRCC	Rd ECC	Clique	Col Gen	LP	Cut Sep	Enum +SP	Nodes	Nodes	Time
c20-3	8	62	10	1.3	0.1	0.3	0.2	1	1	2.5
c20-4	9	95	20	5.8	0.7	0.8	0.5	1	1	9.4
c20-5	2	0	0	0.4	0.1	0.0	0.2	1	1	0.8
c20-6	58	71	19	11.3	1.2	1.5	0.4	1	1	16.9
c50-13	10	111	49	49.8	0.8	9.4	3.4	1	1	75.0
c50-14	23	202	52	137.4	3.8	31.2	217.2	1	1	438.1
c50-15	16	129	54	35.2	1.0	8.4	13.4	1	1	68.2
c50-16	4	156	47	24.6	0.9	7.2	15.6	1	1	58.4
c75-17	8	192	71	346.0	6.7	90.3	3913.8	13	7	14608.9
c75-18	17	213	58	585.0	7.5	132.4	2202.6	7	4	7811.9
c100-19	29	339	118	980.5	23.5	179.1	4380.4	–	–	>10000
c100-20	30	198	99	602.8	6.7	88.2	3297.8	–	–	>10000

Table 5: Comparison of bounds and times on the FIX-VAR variant.

Instance	K	C	Choi LB	Choi Time(s)	New LB	New Time(s)	Prev UB	New UB
c20-3	5	120	1138.58	0.25	1144.22	1.3	1144.22	**1144.22**
c20-4	3	150	6369.15	0.45	6437.34	4.5	6437.33	**6437.33**
c20-5	5	120	1307.74	0.19	1313.78	5.0	1322.26	**1322.26**
c20-6	3	150	6451.62	0.41	6515.98	5.2	6516.47	**6516.47**
c50-13	6	200	2959.60	3.95	2964.65	14.0	2964.65	**2964.65**
c50-14	3	300	8748.57	51.70	9121.29	265.8	9126.90	**9126.90**
c50-15	3	160	2597.22	4.36	2627.56	75.4	2634.96	**2634.96**
c50-16	3	140	3114.00	5.98	3153.93	95.3	3168.92	**3168.92**
c75-17	4	350	1979.87	68.11	1988.37	357.4	2023.61	**2004.48**
c75-18	6	400	3128.75	18.78	3143.38	476.2	3147.99	**3147.99**
c100-19	3	300	8431.87	905.20	8648.74	1099.8	8664.29	8661.81
c100-20	3	200	4082.25	53.02	4129.32	526.1	4154.49	–
avg gap			1.53%		0.34%			

[5] M. Gendreau, G. Laporte, C. Musaraganyi, and E. D. Taillard. A tabu search heuristic for the heterogeneous fleet vehicle routing problem. *Computers and Operations Research*, 26:1153–1173, 1999.

[6] B. Golden, A. Assad, L. Levy, and F. Gheysens. The fleet size and mix vehicle routing problem. *Computers and Operations Research*, 11:49–66, 1984.

[7] D. Houck, J. Picard, M. Queyranne, and R. Vegamundi. The travelling salesman problem as a constrained shortest path problem. *Opsearch*, 17:93–109, 1980.

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

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

[10] L. Ochi, D.Vianna, A. Filho, and L. Drummond. A parallel evolutionary algorithm for the vehicle routing problem with heterogeneous fleet. *Future Generation Computer Systems*, 14:285–292, 1998.

[11] A. Pessoa, M. Poggi de Aragão, and E. Uchoa. A robust branch-cut-and-price algorithm for the heterogeneous fleet vehicle routing problem. In Camil Demetrescu, editor, *WEA*, volume 4525 of *Lecture Notes in Computer Science*, pages 150–160. Springer, 2007.

[12] J. Picard and M. Queyranne. The time-dependant traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations Research*, 26:86–110, 1978.

[13] M. Poggi de Aragão and E. Uchoa. Integer program reformulation for robust branch-and-cut-and-price. In L. Wolsey, editor, *Annals of Mathematical Programming in Rio*, pages 56–61, Búzios, Brazil, 2003.

[14] J. Renaud and F. Boctor. A sweep-based algorithm for the fleet size and mix vehicle routing problem. *European Journal of Operational Research*, 140:618–628, 2002.

[15] E. D. Taillard. A heuristic column generation method for the heterogeneous fleet VRP. *RAIRO*, 33:1–14, 1999.

[16] E. Uchoa, R. Fukasawa, J. Lysgaard, A. Pessoa, M. Poggi de Aragão, and D. Andrade. Robust branch-cut-and-price for the capacitated minimum spanning tree problem over a large extended formulation. *Mathematical Programming*, 112:443–472, 2008.

[17] N. A. Wassan and I. H. Osman. Tabu search variants for the mix fleet vehicle routing problem. *Journal of the Operational Research Society*, 53:768–782, 2002.

[18] A. Westerlund, M. Göthe-Lundgren, and T. Larsson. Mathematical formulations of the heterogeneous fleet vehicle routing problem. In *ROUTE 2003, International Workshop on Vehicle Routing*, pages 1–2, 2003.

[19] H. Yaman. Formulations and valid inequalities for the heterogeneous vehicle routing problem. *Mathematical Programming*, 106:365–390, 2006.

Table 6: BCP statistics on the FIX-VAR variants.

Instance	# Root Cuts			Root Times				# BCP	# SP	Total
	SRCC	Rd ECC	Clique	Col Gen	LP	Cut Sep	Enum +SP	Nodes	Nodes	Time
c20-3	4	21	5	0.8	0.2	0.1	0.2	1	1	1.4
c20-4	6	164	14	2.2	0.5	1.0	0.3	1	1	4.8
c20-5	7	62	3	3.1	0.4	0.6	0.3	1	1	5.3
c20-6	1	102	3	3.0	0.5	1.0	0.3	1	1	5.4
c50-13	4	7	1	13.1	0.2	0.2	2.9	1	1	16.9
c50-14	13	1111	49	103.3	26.8	72.8	21.9	1	1	287.8
c50-15	26	488	16	43.8	3.0	15.6	4.3	1	27	79.7
c50-16	15	358	23	54.1	2.2	15.3	15.7	1	48	111.0
c75-17	19	139	64	236.8	2.7	53.9	6423.4	5	3	8615.9
c75-18	15	184	15	310.8	1.2	89.2	30.2	1	1	506.4
c100-19	43	937	107	623.6	71.9	212.3	2596.3	–	–	>10000
c100-20	42	435	45	356.2	7.1	90.8	5017.0	–	–	>10000