V Algorithmic Approach

In the previous chapter, a flow formulation (IV.3) for the VRPTWEOS was introduced. The scope of this chapter is to propose an algorithmic approach to solve that formulation. The suggested solution approach is based on a reformulation of the flow model, obtained by applying the Dantzig-Wolfe decomposition technique. To solve the decomposed flow model, a column generation algorithm is used along with a branch and bound algorithm. This technique is known as Branch-and-Price, and it has been widely used in Vehicle Routing Problems.

Firstly, we perform a brief review of the Dantzig-Wolfe decomposition method. Next, the decomposed flow model is presented and finally the column generation and branch-and-price algorithms are explained.

V.1 Dantzig-Wolfe Decomposition

In this section we give a brief review of the Dantzig-Wolfe decomposition, as presented by (ARAGAO; UCHOA, 2003). For a detailed explanation of this method please refer to (CASTILLO; GARCÍA-BERTRAND, 2006).

Consider the following integer program with n variables:

$$Z_{IP} = min \quad cx$$

s.t.
$$Ax = b$$
$$Dx \leq d$$
$$x \in \mathbb{Z}^{n}_{+}$$
(IP)

Let $\mathcal{Q} = \{x^1, \ldots, x^n\}$ be the *finite* set of extreme points of the polytope defined by $P := \{x \in \mathbb{Z}_+^n \mid Dx \leq d\}$. Let Q be a $n \times p$ matrix where each column corresponds to an extreme point $x \in \mathcal{Q}$.

There is a one-to-one correspondence between the elements of \mathcal{Q} and the solutions of

$$x = Q\lambda$$

s.t. $\mathbf{1}\lambda = 1$
 $\lambda \in \{0, 1\}^{p}$

The reformulation of the integer program IP follows by replacing the x variables in IP by the expression above. Relaxing the integrality constraints, the *Dantzig-Wolfe Master* linear program is obtained:

$$Z_{DWM} = min \, (cQ)\lambda \tag{DWM}$$

Subject to:

$$(AQ)\lambda = b \tag{V.1}$$

$$\mathbf{1}\lambda = 1 \tag{V.2}$$

$$\lambda \ge 0$$

Where (V.1) are known as the coupling constraints and (V.2) are the convexity constraints.

V.2 Solving the DWM: Column Generation

Notice that there are an exponential number of extreme points in the mentioned polytope P, and therefore an exponential number of λ variables in the DWM LP. It is impractical to solve such a huge model at once. To tackle this issue, a technique called column generation is used.

As a previous step for the column generation procedure, a first version of the DWM with only a small subset of the λ variables (columns), will be constructed. Actually, if artificial variables are included to assure feasibility of the model, a first version of this LP without any λ variables can be used.

The column generation algorithm will start by solving the initial version of the LP, which should be easy to solve. Next, the algorithm will try to find a profitable column to add to the model, this is, a column with a convenient reduced cost that may improve the value of the objective function. This process of finding a suitable column is known as *pricing*.

When a profitable column is found by the pricing algorithm, this column is added to the LP. The column generation algorithm will continue by solving the updated model, and then the pricing algorithm will be called again. The algorithm will continue to perform this solving / pricing loop until no suitable column exists.

Given the current dual multipliers π and π_0 associated to the coupling constraints V.1 and to the convexity constraints V.2 respectively, the pricing sub-problem can be written as the following integer program:

Minimize
$$(c - \pi A)x - \pi_0$$

s.t. $Dx \le d$ (Pricing)
 $x \in \mathbb{Z}^n_+$

As shown before, the basic idea behind column generation is that of using the pricing sub-problem to dynamically generate and add columns to the master problem, which is iteratively solved. This approach is only practical when the decomposed constraints $Dx \leq d$ have a "nice structure", which allows many sub-problems to be solved quickly. (ARAGAO; UCHOA, 2003).

In the next section, we present the Dantzig-Wolfe decomposition of the flow formulation suggested for solving the VRPTWEOS.

V.3 Time Indexed Flow Model Reformulation

The pseudo-polynomially large number of variables and constraints comprised in the time indexed flow formulation (IV.3) may turn its direct use prohibitive. By using the Dantzig-Wolfe decomposition method, it is possible to rewrite this model in terms of variables associated to all possible paths (routes) in the flow graph G (IV.2) that are associated to each vehicle type.

To begin the decomposition of the flow model, recall our assignment constraints (IV.6) and synchronization constraints (IV.7). These constraints will remain in the master as the so called coupling constraints.

The constraints to be decomposed are both the flow initialization constraints (IV.8) and the flow conservation constraints (IV.9). Notice that these constraints define the set Ω of all possible routes that traverse the flow graph IV.2 starting and ending at the depot. Each such route is associated to a vehicle type, thus we introduce λ_r^e variables for each possible route of each vehicle type.

We now establish the following relation between the λ variables and the x variables of the original flow formulation:

$$x_{ijte} = \sum_{r \in \Omega} a_{ijr}^{te} \lambda_r^{\epsilon}$$

$$s.t.\sum_{r\in\Omega}\lambda_r^e = k_e \tag{V.3}$$

$$\lambda^e \in \{0,1\}^r \tag{V.4}$$

Where coefficients a_{ijr}^{te} are defined as

$$a_{ijr}^{te} = \begin{cases} 1 & \text{if variable } x_{ijte} \text{ belongs to the route associated to } \lambda_r^e \\ 0 & \text{otherwise} \end{cases}$$

We now define the cost associated to each λ variable as

$$c_r^e = \sum_{ijte \in A_r} s_{ij}$$

By expressing the x variables of the original flow formulation in terms of the λ variables, and relaxing the integrality constraints we obtain the Dantzig-Wolfe Master:

$$\text{Minimize} \sum_{e \in E} \sum_{r \in \Omega_e} c_r^e \lambda_r^e \tag{V.5}$$

Subject to:

$$\sum_{t \in S_j} y_{jt} = 1 \quad \forall j \in J \setminus \{0\}$$
(V.6)

$$\sum_{i \in J} \sum_{r \in \Omega} a_{ijr}^{te} \lambda_r^e - y_{jt} = 0 \quad \forall j \in J, t \in S_j, e \in E_j$$
(V.7)

$$\sum_{r\in\Omega}\lambda_r^e = k_e \quad \forall e \in E \tag{V.8}$$

$$\lambda \ge 0 \tag{V.9}$$

This reformulation of the flow model has an exponential number of λ variables (routes), meaning that it has to be solved by column generation. Considering the dual multipliers π_{ite} and π_0 associated to the coupling constraints (V.7) and convexity constraints (V.8) respectively, for each vehicle type there will be a pricing sub-problem defined by

Minimize
$$\sum_{i \in J} \sum_{j \in J} \sum_{t \in S_i} (s_{ij} - \pi_{ite}) x_{ijt} - \sum_{j \in J} (s_{ij} - \pi_0)$$
 (V.10)

$$\sum_{i \in J \setminus \{0\}} x_{0i0e} = k_e \forall e \in E \tag{V.11}$$

$$\sum_{i \in J} x_{jite} + w_{jte} - \sum_{i \in J} x_{i,j,t-p_i-s_{ij},e} - w_{j,t-1,e} = 0 \ \forall j \in J, t \in S_j, e \in E_j$$
(V.12)

As already mentioned, constraints (V.11) and constraints (V.12) define the set of all possible paths in the flow graph shown in Figure IV.2 starting and ending at the depot. By solving the above IP formulation, suitable routes to be added to the master problem can be found. Notice though, that this includes even routes that may visit the same customer more than once.

(a) q-Route Pricing

By definition, a feasible solution to the VRPTWEOS will not contain routes with cycles, i.e. routes that visit the same customer more than once. Because of this, it is desirable to have a pricing algorithm that generates only profitable routes without cycles (see Figure V.1). This problem is known as the Elementary Shortest Path Problem with Resource Constraints (ESPPRC), which is known to be \mathcal{NP} -hard (DESAULNIERS *et al.*, 2010).



Figure V.1: Elementary route on the flow graph.

A way of not handling that level of complexity inside the pricing sub-problem is to relax the routes allowing them to be non-elementary, this is, allowing routes with cycles, more specifically routes that visit the same customer more than once. It is feasible to do so, because when such a route is added to the master problem, the coupling constraints will take care of not adding cycles to the final solution. It becomes clear that, the more cycling routes the sub-problem generates, the more extra work the master problem will have disregarding infeasible scenarios, and hence, the less information each route will provide to the final solution. This issue contributes negatively to the convergence of the column generation method but at the same time maintains the complexity of the pricing sub-problem tractable, which is a necessary condition for the column generation to be effective.

This type of route is known in the literature as q-Routes, and the problem of generating such routes is known as the Shortest Path Problem with Resource Constraints (SPPRC). The SPPRC is known to be \mathcal{NP} -hard in the weak sense. (CHRISTOFIDES *et al.*, 1981) describe pseudo-polynomial time algorithms to tackle this problem. Figure V.2 shows an example q-route. Notice how this route cycles through customers 2 and 3 over time.



Figure V.2: Example of a q-route on the flow graph.

Until now, we have been talking of cycles in the sense of a route visiting the same customer multiple times. Nevertheless, as traveling back in time is impossible, it is easy to see that the flow graph we are working on is an acyclic graph. Moreover, a straightforward topological order of the vertices in G = (V, A) can be achieved by ordering them by time index. The costs c_{ij}^{te} associated to the arcs of this graph correspond to the reduced costs described in (V.10). To solve a shortest path problem in this graph, a reaching algorithm described by (AHUJA *et al.*, 1993) can be used.

The above can be achieved in $\mathcal{O}(|A|)$ as follows. Let $F(j,t)_e$ denote the minimum reduced cost sub-path that starts at the depot and finishes with an arc $(i, j)_e^t$. We start the reaching algorithm by setting distance labels

$$F(j,t)_e = \begin{cases} 0 & \text{if } j=0 \text{ and } t=0 \\ +\infty & \text{otherwise} \end{cases}$$

We begin to process all vertices of G in the topological order, naturally starting with vertex (0,0), the depot. For each vertex i being processed we scan vertices in its adjacency list A(i). If for any arc $(i,j)_e^t \in A(i)$ we find that $F(j,t+p_i+s_{ij})_e > F(i,t)_e + c_{ij}^{te}$ then we set $F(j,t+p_i+s_{ij}) = F(i,t)_e + c_{ij}^{te}$. Before processing all nodes once in this order, the distance labels are optimal.

Notice that for our problem $|A| = n^2T$ so this pseudo-polynomial algorithm has complexity $\mathcal{O}(n^2T)$. Solving this pricing algorithm can be quite consuming. Furthermore, this algorithm will produce routes with cycles. (CHRISTOFIDES *et al.*, 1981) describe a simple way to extend this algorithm to avoid 2-cycles, which are cycles of the form $(i \to j \to i)$. This extension does not change the complexity of the pricing algorithm but gives routes of better quality which helps the convergence of the column generation approach. An example of a q-route with 2-cycle elimination can be seen in Figure V.3.



Figure V.3: Example of a q-route with 2-cycle elimination on the flow graph.

V.4 Column Generation for Extended Formulations

The column generation approach previously explained repeatedly adds promising extreme points of the polytope, described by the decomposed constraints, to the Dantzig-Wolfe Master until no such extreme point exists. Therefore, the problem is iteratively solved over an approximation of, or partially described, original polyhedron. One advantage of this method is that of working only with a small subset of an exponentially large set of variables. The column generation for extended formulations is an alternative to the standard column generation approach, and is described by (SADYKOV; VANDERBECK, 2011). Instead of working over the Dantzig-Wolfe Master, this approach considers the original variable formulation restricted to a subset

of variables. Nevertheless, in order to be able to use the column generation for extended formulations approach, the target model must have a decomposable structure that makes it suitable for Dantzig-Wolfe decomposition. This way, the pricing sub-problem corresponds to the pricing of the associated standard column generation approach. However, instead of adding the priced columns to the Dantzig-Wolfe Master, these columns are *lifted* by expressing them in terms of the original variables and then added to a restricted version of the original variable formulation. This approach should be considered as an alternative to the standard column generation approach when convergence problems occur. The main advantage of the column generation for extended formulations is that the lifted columns act as a stabilization technique for column generation, thus accelerating the convergence of the method. Recalling our original flow formulation IV.3, the column generation for extended formulations will begin with a restricted version of the model, in which initially no x variables exist and integrality constraints are relaxed. Without the x variables, this model is infeasible. To avoid infeasibility, artificial variables are introduced to the assignment constraints (IV.6) and to the flow initiation constraints (IV.8). The initial model is solved using a generic LP solver and then the pricing algorithm will try to find a profitable column. As mentioned before, this pricing sub-problem corresponds to the same pricing algorithm associated to the standard column generation approach. This time the dual multipliers π_{ite} and π_0 used to compute the reduced cost of the columns in the pricing sub-problem V.10 are now associated to the flow constraints (IV.9) and flow initialization constraints (IV.8) of the restricted original flow formulation, instead to the rows of the Dantzig-Wolfe Master. Whenever a column is generated by the pricing algorithm, the x variables associated to it are added to the model. Similarly to the standard column generation method, this process will continue until no profitable column is found.

V.5 A Branch-and-Price Algorithm

The column generation approaches explained before are intended to solve a linear relaxation of the flow formulation. It may be the case that the solution to this linear problem is integral, but if not, another technique must be used to achieve integral solutions. An algorithm commonly used for this purpose, is the well known branch-and-bound algorithm. This algorithm works on a search tree to make an implicit enumeration of the solution space, disregarding branches for which the best possible solution is worse than a previously known solution. When branch-and-bound is used with column generation, they receive the name of Branch-and-Price. Next, we describe our Branch-and-Price implementation, which can be found in Algorithm V.1.

(a) BaP Algorithm Description

Before the start of the branch-and-price algorithm, some global parameters are set. These parameters correspond to the best bound and the best current solution value, named as LB and UB respectively for our minimization problem. These parameters are initialized, to denote that no current best values exist.

Our implementation of the Branch-and-Price algorithm starts by creating the root node of the search tree. The root node will correspond to one of the relaxation problems described for the flow formulation. The so called search tree is implemented using a stack, therefore the nodes of the search tree are traversed by a DFS.

At each iteration, the algorithm selects the node on top of the stack and solves it using one of the column generation approaches previously explained. As soon as the column generation procedure ends, the branch-and-price algorithm gets the value of the current node solution Z_{LP} (if such a solution exists). Next the following checks are made:

- If the column generation model is infeasible, return - If $[Z_{LP}] \ge UB$, return

If any of those two cases proceed, then the loop-up procedure over the current branch of the search tree will be abandoned, in the first case by infeasibility and in the second case by bound.

If neither of the above conditions are met, the algorithm verifies if the solution of the current node is integral. If that is the case, the algorithm will no further explore that branch, as no better solution will exist in it. However, before continuing the algorithm checks if the value of the current node solution is better than the best known solution so far, i.e. if $Z_{LP} < UB$, then the global bound is updated by making $UB = Z_{LP}$ and the associated solution is stored.

If the solution is not integral and the current node is the root node, then the value of the best bound is updated: $LB = Z_{LP}$.

Next, the look-up procedure continues by opening two new branches from the current node. The branching procedure, in our implementation, comprehends the selection of the most fractional variable in the solution, i.e. the variable which value is closer to 0.5. However, it could be the case that such variable is a λ variable (in the Dantzig-Wolfe decomposition approach). Branching on a λ variables will modify the structure of the column generation pricing sub-problem, making it more complicated, eventually only being able to solve it by costly IP techniques (ARAGAO; UCHOA, 2003).

The alternative is to branch on the y and x original formulation variables. With this in mind, we give precedence to the y variables. Branching on the y variables seems "stronger" as they couple groups of x variables.

Let x^b be the selected variable to branch on. The branch-and-price algorithm continues by making two copies of the current node, corresponding to the left and right children of the current node in the search tree. Before adding such nodes to the stack, the branching restrictions are imposed as follows:

- For the left child of the current node, branching constraint $x^b \leq 0$ is added.
- For the right child of the current node, branching constraint $x^b \ge 1$ is added.

Next, the two children nodes are added to the stack. The Branch-and-price algorithm continues until there are still nodes in the stack, the integrality gap arrives at a globally defined tolerance value, or the algorithms achieve a given time limit.

Algorithm V.1 Branch and Price Algorithm	
1:	procedure BAP(rootNode)
2:	input: Root <i>node</i> of the search tree.
3:	output: Best solution found, if one exists.
4:	$LB \leftarrow -\infty$
5:	$UB \leftarrow \infty$
6:	$ZInc \leftarrow \infty$ //value of best current solution found
7:	$bestSolution \leftarrow \emptyset$
8:	$nodeCount \leftarrow 0$
9:	$stack \leftarrow emptyset$
10:	stack.push(node)
11:	$\mathbf{while} \ stack \neq \emptyset \ \mathbf{do}$
12:	$currentNode \leftarrow stack.pop()$
13:	$currentSolution \leftarrow solveLPByColumnGeneration(currentNode)$
14:	$Zlp \leftarrow \infty$
15:	$Zlp \leftarrow currentSolution.value$
16:	$doBranching \leftarrow true$
17:	if $Zlp = \infty$ then
18:	$doBranching \leftarrow false$
19:	else if $ Zlp \ge ZInc$ then
20:	$doBranching \leftarrow false$
21:	else
22:	If $nodeCount = 0$ then
23:	$LB \leftarrow Zlp$
24:	if isIntegerSolution(currentSolution) then
25:	if $Zlp < ZInc$ then
26:	$bestSolution \leftarrow currentSolution$
27:	$ZInc \leftarrow Zlp$
28:	$doBranching \leftarrow false$
29:	else if $\lceil Zlp \rceil \ge Zinc$ then
30:	$doBranching \leftarrow false$
31:	$x^b \leftarrow getMostFractionalVariable(currentSolution)$
32:	$leftNode \leftarrow currentNode$
33:	$rightNode \leftarrow currentNode$
34:	$leftNode.addBranchingConstraint(x^b \leq 0)$
35:	$rightNode.addBranchingConstraint(x^b \ge 1)$
36:	stack.push(leftNode)
37:	stack.push(rightNode)
38:	$nodeCount \leftarrow nodeCount + 1$
39:	return bestSolution